D5.1 ICOS Alpha Release

Document Identification			
Status	Final	**Due Date**	30/11/2023
Version	1.0	**Submission Date**	30/11/2023

Related WP	WP5	Document Reference	D5.1
Related Deliverable(s)	D2.1, D2.2, D3.1, D4.1	Dissemination Level (*)	PU
Lead Participant	TUBS	Lead Author	Marc Michalke
Contributors	TUBS, NKUA, ENG, ZSCALE, ATOS, NCSRD, SIXSQ, UPC, XLAB	Reviewers	Lara López (ATOS)
			Ivan Paes (ZSCALE)

Keywords:
System integration, CI/CD, Release

Document Information

List of Contributors	
Name	**Partner**
Marc Michalke	TUBS
Francisco Carpio	TUBS
Admela Jukan	TUBS
Fin Gentzen	TUBS
Anastasios Giannopoulos	NKUA
Gabriele Giammatteo	ENG
Panagiotis Gkonis	NKUA
Ivan Paez	ZSCALE
Alex Volkov	ATOS
George Xylouris	NCSRD
Dimitris Santorinaios	NCSRD
Andreas Oikonomakis	NCSRD
Nikos Dimitriou	NCSRD
Konstantin Skaburskas	SIXSQ
Nabil Abdennadher	SIXSQ
John White	SIXSQ
Jordi Garcia	UPC
Xavi Masip-Bruin	UPC
Montse Farreras	UPC
Hrvoje Ratkajec	XLAB

Document History			
Version	**Date**	**Change editors**	**Changes**
0.1	26/10/2023	Marc Michalke (TUBS)	Initial table of contents.
0.2	15/11/2023	Marc Michalke (TUBS)	Partner contributed content for all sections
0.3	16/11/2023	Marc Michalke (TUBS)	Harmonization for internal review
0.4	27/11/2023	Marc Michalke (TUBS)	Reviewers inputs included
	29/11/2023	Carmen San Roman (ATOS)	Quality assurance review
1.0	30/11/2023	Francesco D'Andria (ATOS	Final version to be submitted

Quality Control		
Role	**Who (Partner short name)**	**Approval Date**
Deliverable leader	Marc Michalke (TUBS)	27/11/2023
Quality manager	Carmen San Roman (ATOS)	29/11/2023
Project Coordinator	Francesco D'Andria (ATOS	30/11/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Delivery
CLI	Command Line Interface
CPU	Central Processing Unit
CSR	Certificate Signing Request
DM	Deployment Manager
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
IT-X	Iteration X of the project
ML	Machine Learning
NFS	Network File System
OCI	Open Container Initiative
OCM	Open Cluster Management
POSIX	Portable Operating System Interface
QA	Quality Assurance
QoE	Quality of Experience
QoS	Quality of Service
TlS	Transport Layer Security
UI	User Interface
WP	Work Package

Executive Summary

The ICOS project aims to cover the range of challenges that arise when addressing the Cloud-Edge-IoT continuum paradigm, proposing an approach that embeds a well-defined set of functionalities, culminating in the definition of an IoT2cloud operating system. ICOS's mission is to design, develop and validate a meta operating system for the Cloud-Edge-IoT continuum, by addressing the following challenges:

- devices volatility and heterogeneity
- continuum infrastructure virtualization and diverse network connectivity
- optimized and scalable service execution and performance
- resources consumptions, including power consumption; guaranteed trust, security and privacy
- reduction of integration costs and effective mitigation of cloud provider lock-in effects

ICOS realizes this within a data-driven system built upon the principles of openness, adaptability, data sharing and a future edge market scenario for services and data.

This document describes and accompanies the first release of the ICOS software; the ICOS Alpha release. The release is based on the work carried out by the consortium to implement the architecture for the ICOS system as defined in the document "D2.2 - ICOS Architecture Design (IT-1)" [1] for the first iteration of the project. This document will outline the functionalities already implemented to achieve the first subset of the project's objectives and of the use cases' needs. This first software release has alpha status, meaning that it consists of first implementations of core functionalities, many of them in the proof-of-concept stage which enables not only integration testing but also the identification of gaps within the previous architecture design and their respective solutions.

The document includes details on the current extent of the integration suites, the status of the implementation which also covers the deviation from the previous architecture and the plans for the two future beta and final releases. Furthermore, the document describes the development process as well as the testbed and workflow that is used for the integration tests. Another aspect addressed in this deliverable is the availability of the code as well as some very basic description of the installation process and references to the more in-depth installation instructions.

The project has set up an integrated development environment to allow for continuous development, integration, and testing throughout the rest of the project. Leveraging this approach, the ICOS alpha release has been created which demonstrates the basic functionality of ICOS and serves as a proof-of-concept for most parts of the architecture.

1 Introduction

1.1 Purpose of the document

This document lays out details about the ICOS alpha release; the result of joint work under the umbrella of work package 5 which also allows for the presentation of the work that happened so far within the technical work packages 2-4. The main goal is to describe the software that is released for this first iteration, how it can be deployed, how the components interact with each other, and which functionalities are currently included as well as the resulting plans for the remainder of the project. This provides the reader with an overview on what can be expected from the alpha release and how.

The deliverable elaborates upon changes to the architecture as well as the three ICOS suites and the currently included functionalities, a subset of which is demonstrated through a basic hello-world workflow. It then describes the tools and requirements needed to deploy the suites and points out where complete installation instructions as well as the artifacts can be found. Furthermore, the testing strategy and the alpha testbed are described, based on which the technical performance results are obtained. An additional aspect of this deliverable is the description of the development process and how the project follows a continuous integration and delivery strategy.

1.2 Relation to other project work

This deliverable presents the results of the architecture definition as well as changes to the design presented previously in "D2.2 ICOS Architecture Design (IT-1)" [1]. It presents the first of a total of three software releases throughout the project and therefore precedes the beta release D5.2 and the complete release D5.3 in month 22 and 32 of the project respectively.

Since the ICOS alpha release shows results of all technical work packages, this document is closely related to deliverables D3.1 [2] and D4.1 [3] which describe the ICOS components of the meta kernel and the intelligence layers respectively in detail. To avoid redundancies, the descriptions in this document have been kept high-level. For more detail the reader is kindly asked to refer to these documents.

1.3 Structure of the document

This document is structured in 6 major chapters:

1. "Introduction": (this section) presents the objectives of the document and introduces its content, structure and relationships with the other project's documents
2. "ICOS suites": describes the three different ICOS suites and their components implemented in the alpha release
3. " Implementation and plans": provides an overview about the current implementation status, the changes made to the architecture during implementation and the plans for currently unimplemented features
4. "Alpha testbed": elaborates upon the testbed that is currently used to perform integration testing and performance evaluation
5. " Development process": explains the used tools and the implemented strategy for the development process
6. "Conclusions": summarizes the content of the document and provides final considerations on how the technical work will continue in the project.

2 ICOS Suites

2.1 Technologies and Strategy

The Logical view of the ICOS architecture, described in Section 4 of D2.2 [1], identifies two main components of the ICOS System: the ICOS Controller and the ICOS Agent. The ICOS Controller is responsible for managing the continuum (keeping track of the topology and availability of the current system) and the run-time (launching and monitoring the execution of services on-demand). The ICOS Agent is responsible for the execution of the services taking care of the code execution and the data accesses, for providing the telemetry of the execution of the service and their resource usage and, eventually, for runtime communication with other Agents.

ICOS Agents are distributed along the continuum and run in a wide range of heterogeneous devices from restricted nodes in the far edge to high-performance computing resources at Cloud level. Each ICOS Controller manages a set of Agents based on a proximity criterion; hence, ICOS Controllers are deployed on resource-rich computing facilities along the continuum to cover the whole geographical area.

Given the different purposes and requirements for the ICOS Agent and the ICOS Controller, ICOS delivers two complete packages (or suites) to facilitate the installation, deployment and update of these ICOS Agents and Controllers. Each suite comprises all the necessary software components and sets them up to deliver the expected functionalities. Given the wide adoption of containerization (Docker) and clustering (Kubernetes) technologies and their presence as a baseline in several technologies described in the state of the art and use cases, these suites will be delivered in the form of Helm[1] charts. This will support the integration of the ICOS system in multiple ways:

- support the achievement of some of the driving principles of the architecture like modularity, robustness and extensibility
- facilitate the integration of components at runtime providing common networking layer
- streamline the distribution, configuration, and deployment of the components
- support the compatibility and sustainability of the ICOS System with external systems and infrastructures

The usage of Helm charts requires a working installation of the Kubernetes container orchestration engine or a compatible distribution (e.g. k3s[2], k0s[3]). For devices that only offer a pure OCI-compatible container runtime (e.g. CRI-O[4], containerd[5]), there is no Helm support, which is usually only relevant for restricted devices. If such devices must be used to deploy an ICOS agent, alternative deployment tools must be leveraged. For the alpha release, this option has been dismissed but it will be taken into consideration for the beta and final releases of the project.

The ICOS Agent and ICOS Controller suites facilitate deployment of an ICOS system; however, the ICOS architecture also comprises a tool for allowing external users to interact with the system. This tool should be a standalone appliance that can be installed on machines that are not part of the ICOS system such as a personal laptop or a mobile phone. For that purpose, ICOS requires a third suite, the Client Suite, that facilitates the installation of this tool. In such devices, containerization and clusterization solutions cannot be assumed; therefore, the installation of the binary should be done in the host system.

The following sections delve into the details of each of these suites starting from the Client Suite, continuing with the Controller Suite, and finishing with Agent Suite. Table 1 summarizes which

[1] https://helm.sh/
[2] https://k3s.io/
[3] https://k0sproject.io/
[4] https://cri-o.io/
[5] https://containerd.io/

components described in the architecture belong to each of the suites for the alpha release, it will be extended in future releases. The components that have not been included in the table are not necessary to deliver the functionalities expected for this alpha release. Each suite's distribution is described under the respective subsection.

Table 1: ICOS Component-Suite Mapping

Suites composition			
Component	Client	Controller	Agent
Shell CLI	X		
Shell BackEnd		X	
Job Manager		X	
Aggregator		X	
Deployment Manager			X
Intelligence API		X	
Security Layer Coordination Module API		X	

2.2 Client Suite

In the first iteration of the ICOS project, the client suite only consists of the ICOS shell command line interface. In the two future iterations, the graphical user interface will be added to mirror at least the same functionality that the command line interface offers and the functionality of both will be continuously extended.

2.2.1 Prerequisites

The ICOS client suite currently only consists of a single binary resembling the command line interface. Therefore, the hardware limitations for this release are quite low. Currently, a single-core CPU of one of the supported architectures (AMD64 and ARM64) is recommended with a minimum of 512 MB of memory. Hardware below this recommendation might still be compatible, but has not been tested.

Regarding the software requirements, no extended prerequisites are necessary besides a current Linux/GNU-based operating system. Since the shell client is a command line application, the target host does not need to have any desktop environment but simply must run a POSIX-compatible shell and a working TCP/IP stack. The user is then expected to provide a configuration file at $XDG_CONFIG_HOME/icos-shell-client/config.yaml, where XDG_CONFIG_HOME defaults to the user's $HOME/.config directory if the variable is not defined. An example configuration file can be copied from the development Gitlab or public Github repository and then customized to the individual environment. Once done, the shell can be used to execute commands against an ICOS installation defined in the config file.

2.2.2 Contained modules

The ICOS client suite fulfills the role of a user interface within the ICOS ecosystem. It is used by the user to interact with the system, querying the current status of resources like running jobs, creating new resources and authenticating against the system.

For the alpha release, this suite consists only of the shell client binary, which is part of the ICOS shell layer, complemented by the ICOS shell backend which is part of the ICOS Controller suite and provides the shell client's point of communication.

2.2.3 Distribution

The shell client binary will be distributed as a single binary per compatible CPU architecture. These binaries and their source code are published internally via the development Gitlab instance and publicly through Github at https://github.com/tubskns/icos-shell/releases.

Copies of these binaries can simply be copied to and then run on any system that matches the minimum requirements without any complex installation process being required after the prerequisites are met. Once the client suite is extended, abstractions for more complex installation procedures are planned to allow for easy installation and access to not just the command line interface but also the graphical user interface. At this point, the client suite will also be released through the same release channel as the other suites; hosted the ICOS Gitlab repository[6].

2.3 Controller Suite

Like the client suite, the controller suite only provides a subset of components for this alpha release. In the two future iterations, the missing components will be added to reach the fully developed state of the project.

2.3.1 Prerequisites

Since the ICOS controller suite consists of several modules that constitute the most important part of the system, the minimum hardware requirements are a compatible host with Kubernetes installed, 2 GB of RAM and 2 CPU cores per machine as well as 40GB of storage. The installation furthermore requires full IP network connectivity between all machines that are part of the cluster (public or private network). For details on how to reach the required state of an installed Kubernetes distribution, the user can refer to the Kubernetes documentation:

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

It should be noted that while many distributions of Kubernetes, e.g. k3s, are compatible with ICOS, an exhaustive list cannot be formed yet at this stage of the project. Therefore, the current recommendation is limited to plain Kubernetes to host the controller suite with the following specifications:

- Kubernetes 1.19 or higher
- Kubectl[7]
- Kustomize[8]
- Clusteradm[9]
- CSR Authentication support
- Orchestrator's Control Plane

On top of this, the ICOS controller suite can then be deployed in form of a Helm chart.

2.3.2 Contained modules

For the alpha release, the Controller Suite consists of the following components, all packaged together into a single Helm chart:

- Shell Backend: Entry point to ICOS ecosystem from Client Suite.
- Job Manager: Scheduler that manages the actual work to be performed by ICOS Controller.
- Aggregator: Provides the current topology of ICOS Ecosystem for allocation purposes.

[6] https://production.eng.it/gitlab/icos
[7] https://kubernetes.io/docs/reference/kubectl/kubectl/
[8] https://kustomize.io/
[9] https://github.com/open-cluster-management-io/clusteradm

- Matchmaker: In charge of providing the target where the work must be executed.
- Intelligence API: Provides and interface with the Intelligence Layer
- Security Layer Coordination Module API: Provides and interface with the Security Layer

This list will be extended throughout subsequent releases as the functionality of ICOS expands.

2.3.3 Distribution

For the distribution of ICOS Controller Suite, the corresponding Helm Chart is provisioned. This chart embeds all previously mentioned components as sub-charts so they can all be installed at once and configured through a single configuration file. Extensive information about Helm Charts is available through the official website at https://helm.sh/docs/topics/charts/.

The ICOS Controller Helm Chart is available at the ICOS OCI Repository harbor.res.eng.it/icos/helm and can be downloaded from there through Helm. For development and unreleased versions, a separate repository is used at harbor.res.eng.it/icos-private/helm.

2.4 Agent Suite

In the Alpha release, the ICOS Agent will contain only a single component; the Deployment Manager. Its task is the management of deployments which includes their creation, reading, update and deletion as instructed by the ICOS controller. The component design is detailed in deliverable D3.1 [2].

2.4.1 Prerequisites

In its current form, the ICOS agent suite displays rather low hardware requirements for deployment since the Deployment Manager is a rather lightweight component consisting of a single binary that is not expected to be larger than 50MB, consume no more than 25MB of RAM and needs approximately 0.5 CPU cores. The binary can be delivered for ARMv6/7/8 and Linux/Windows x86-64. Since the Deployment Manager is packaged as a Docker container image, the installation furthermore requires the preexistence of an OCI-compatible container runtime on the host running the agent suite.

2.4.2 Contained modules

By the adopted design (see D3.1 [2]), the DM will connect various Edge-Cloud Orchestrators with the ICOS Controller for management of the user application deployments. For that reason, the Edge-Cloud Orchestrator specific implementations of DMs must be produced. For ICOS Alpha, the project integrated two Edge-Cloud Orchestrators: OCM[10] and Nuvla[11]. Hence, two DM implementations were produced, one for OCM and the other for Nuvla.

2.4.3 Distribution

For the distribution of the ICOS Agent Suite, the corresponding Helm Chart is provisioned. This chart embeds all related components as sub-charts so they can all be delivered at once. It has to be adapted to the specific hardware environment of the cluster or node that shall become part of ICOS by providing an individually tailored values.yml file upon installation.

The ICOS Agent Helm Chart is available at the ICOS OCI Repository harbor.res.eng.it/icos/helm and can be downloaded from there through Helm. For development and unreleased versions, a separate repository is used at harbor.res.eng.it/icos-private/helm.

[10] https://open-cluster-management.io/
[11] https://nuvla.io/ui/

3 Implementation and Plans

This section highlights the changes made to the architecture presented in D2.2 [1] and in the architecture whitepaper [5] as well as the reasons behind them. Furthermore, current shortcomings are pointed out. Next, an estimated roadmap by IT-2 and IT-3 for the services not considered in IT-1 is presented, along with the expected improvements for those already included in IT-1.

3.1 Updates and Changes to the architecture

3.1.1 Node Onboarding Service

The Node Onboarding Service is one of the most essential ICOS functionalities to constitute and maintain the continuum. In this service, an ICOS Agent node needs to join an active ICOS Controller node before it is exploitable by ICOS. It involves several components from the Agent, Controller, the Lighthouse, as well as the communication of multiple Controllers to exchange local views. Although this service has been deeply investigated and defined in the first iteration of the architecture definition, IT-1 is an early-stage development and integration work. Furthermore, bringing in cross ICOS Controller communication first requires basic ICOS Controller functionality to be in place; the work on cross ICOS Controller communication will be done after IT-1.

Most efforts in IT-1 have been focused on the runtime layer capabilities of an ICOS system, addressing all system components for a proper ICOS application deployment and execution in the continuum, and assuming that the ICOS system is already operative and stable.

3.1.2 Aggregator

The ICOS Meta-Kernel Layer has been designed with a component named Topology, which is responsible for collecting, storing, and maintaining updated data related to the continuum. These data are collected from the ICOS nodes that join the continuum, as well as from the Telemetry and Logging components, and will be mainly used by the Intelligence Layer to classify the nodes' capabilities and availability. In order to process the data from the Topology and provide infrastructure information to the Job Manager service, a service named Aggregator has been introduced in D3.1 [2]. This service is responsible for providing knowledge of the static and dynamic properties of a resource infrastructure associated with a particular ICOS ecosystem deployment. For IT-1, the aggregator component has been implemented as a web service that exposes a web API that is called by the matchmaking service and provides the infrastructure taxonomy that describes the characteristics of the ICOS infrastructure.

3.1.3 Execution Manager / Offloader

The purpose of the Execution Manager component is to provide a mechanism that allows service components to decompose application components into sub-components (or tasks) to enable massive/distributed parallel workload execution. Although this service is one core component in the ICOS Meta-Kernel architecture, in IT-1 the application deployment and execution has been focused on a basic single-component application, so the current delivery doesn't consider application parallelism exploitation (as noticed in D3.1 [2]).

3.2 Roadmap

3.2.1 Implementation planned for IT-2

- **Node onboarding service, basic functionality**. A new node will be dynamically added to an active ICOS Controller and, therefore, be onboarded on the corresponding ICOS infrastructure. The new node will be authenticated, provide the meta information about its features and capability, and will be configured to interface with the monitoring infrastructure. In IT-2 the node onboarding will be done independently at each Controller, this is, without cross-Controller communication to exchange local views and decide the best Controller to be assigned to (IT-3).
- **Enhanced job management and matchmaking services**. In IT-2 the Job Management component services will be enhanced to consider multi-component and massively parallel applications, and provide more sophisticated application multi-component deployments. This involves: *(1)* enhancing the application description model to include more elaborated constraints, providing application description information to feed the deployment decision process, and specifying the telemetry data to be considered during application monitoring; *(2)* improving the application deployment description model to consider more sophisticated deployment layouts for multi-component applications; and *(3)*, implementing a more effective matchmaking service to provide more appropriate solutions based on heuristics. AI-based solutions will be implemented in IT-3.
- **Dynamic adaptation feedback loop**. ICOS handles a variable workload running on a dynamic infrastructure. For ensuring that the services orchestrated on the Continuum meet the expected levels of QoE and QoS, it is necessary to establish a mechanism that provides information regarding the behavior of the service, alert about violations about service level agreements and provide a proper response to it. In IT-2, this feedback loop will be implemented and the policies to ensure the QoS will be used building on the metrics collected. For achieving a better response of the ICOS system, the Distributed and Parallel Execution Manager will be integrated in this feedback loop publishing information about the execution of the service to improve the estimations of the current workload of the system and its current ability to meet the service level expectations.
- **ICOS GUI shell**. The ICOS Shell suite will be improved in IT-2 with a graphic interface (GUI) and provide enhanced functionalities and users' services.
- **Integrated API with data management**. In IT-2 the data dispatching functionality will be added to an active ICOS Controller, and onboarded in the ICOS infrastructure. As presented in D4.1[3], the data management will expose a specific API that allows it to communicate with the Distributed & Parallel Execution module within the Distributed Meta-Kernel Layer, which coordinates the execution of processing tasks. This enables the data management module to control where the data is and schedule task execution (i.e. pub/sub/query/process) accordingly via a data-centric approach.
- **Intelligence Layer with basic functionality**. Towards IT-2, the Intelligence API will provide integration with the MLFlow MLOps framework to keep track of ML workloads. Additionally, it will add support to communicate with Dataclay[12], and Zenoh[13], which are part of the data management framework. In IT-2, the Intelligence layer will provide basic scripts to use cases to guide them on how to use the API.
- **Security Layer functionalities.** The security functionalities planned for IT-2 are the following. Active scanning for security issues within deployed ICOS resources and providing metrics for assessing resources' trustworthiness score which can be used by the Agreggator and Matchmaking components. Scanning and detection of anomalies in the ICOS ecosystem using log based anomaly detection tools. Basic management of security compliance policies for ICOS resources. Authentication and Authorisation of ICOS service-to-service and controller-to-controller calls in line

[12] https://dataclay.bsc.es/

[13] https://zenoh.io/

with the Zero-trust principle. And encryption of communications between ICOS services using TLS encryption mechanism, which will also provide trust between ICOS nodes.

3.2.2 Implementation planned for IT-3

- **Multi-controller cross communication**. In IT-3 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.
- **Intelligent application deployment**. The matchmaking process will be AI-driven in IT-3 in order to provide near-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 last release of ICOS, the dynamic adaptation loop will be improved in two directions. First, it will publish new metrics in the monitoring infrastructure to allow a better understanding of the service behavior and its current status/needs. Second, it will leverage on AI to make decisions on the current resource allocation and propose possible adaptations.
- **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 catalog).
- **Security Layer**. The Security Layer functionalities, planned for IT-3, 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.

4 Alpha Testbed

4.1 Infrastructure

The infrastructure environment provided by **NCSRD** as depicted in Figure 1, is designed to accommodate a wide range of computing and networking needs, including virtualization, container orchestration, and secure remote access. It consists of a Proxmox[14] cluster composed of three dedicated physical servers, leveraging the Proxmox Virtual Environment (Proxmox VE) for management of virtual machines and containers. These physical servers are interconnected through a Gigabit Ethernet network switch. A centralized 20TB Network File System (NFS) storage solution is integrated into the cluster. This storage is indispensable for data sharing and redundancy, allowing virtual machines and containers to access and store data across the network with high availability. Also, a Kubernetes cluster has been deployed, consisting of one master node and three worker nodes. This cluster further enhances the environment's capabilities, offering advanced container orchestration and scaling.

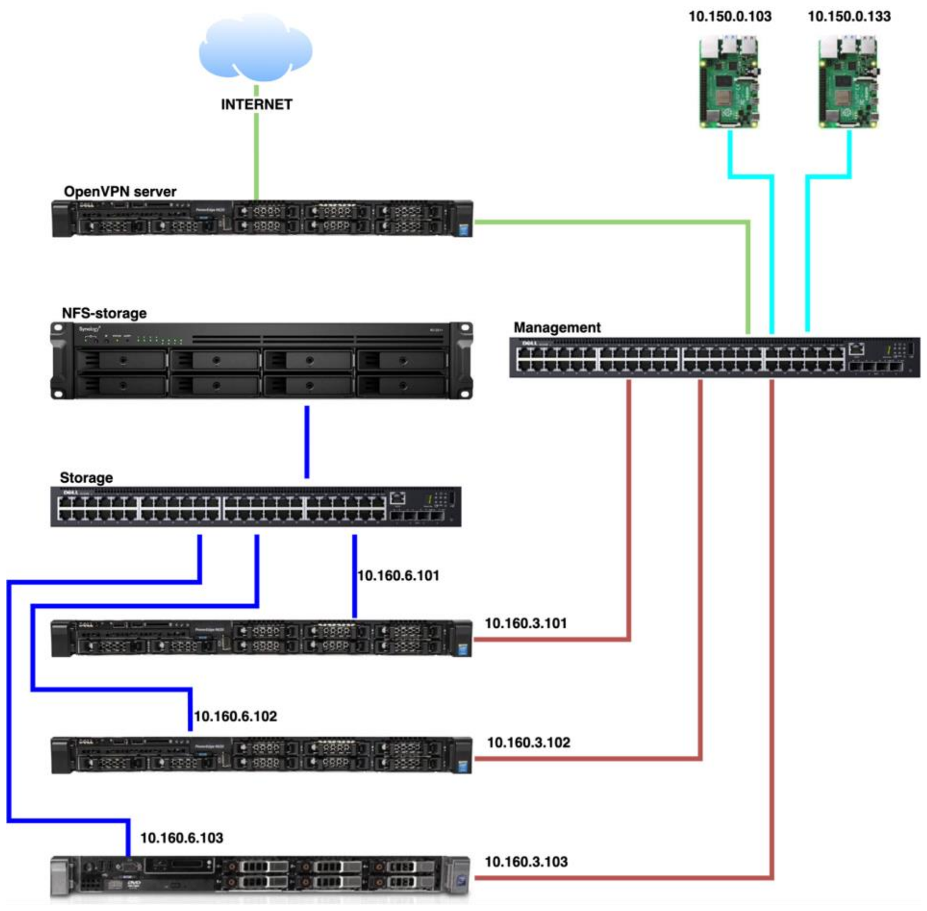

Figure 1 NCSRD Testbed

Additionally, two Raspberry Pi 4 devices are incorporated, each serving unique roles;

One Raspberry Pi runs k3s, a lightweight Kubernetes distribution, and is configured with the Cilium Container Network Interface[15] (CNI), the second Raspberry Pi is dedicated to Minikube[16], enabling the local deployment of a single-node Kubernetes cluster. OpenVPN has been used to establish secure and

[14] https://www.proxmox.com/en/
[15] https://cilium.io/
[16] https://minikube.sigs.k8s.io/docs/

encrypted connections for remote access. User accounts have been carefully configured to ensure secure remote administration, monitoring, and access to the environment.

TUBS provides a set of four office-class PCs that are connected with a Gigabit network switch. In addition to these devices, a set of four Raspberry Pi 3 can be deployed at the same switch to test ICOS against low-end hardware.

The infrastructure at **TUBS** premises on the other hand will be used to verify the reproducibility of the setup process once the first deployment at NCSRD succeeded. To achieve this, TUBS will work with the other partners to follow the documented deployment steps, identify gaps and forward this information to close them in the documentation. As a side effect, this process will also verify the compatibility with different architectures since TUBS will aim to involve Raspberry Pi 3 computers into the installation process.

The infrastructure environment provided by **UPC**, as depicted in Figure 2, is designed to accommodate a wide range of computing and networking needs, including virtualization, container orchestration, and secure remote access.

It consists of a Kubernetes cluster composed of three dedicated physical servers, consisting of one master node and two worker nodes. This cluster enhances the environment's capabilities, offering advanced container orchestration and scaling. Furthermore, we have one additional physical server dedicated to Proxmox, leveraging the Proxmox Virtual Environment (Proxmox VE) for the management of virtual machines and containers. In this server, new workers or even additional clusters can be deployed to the real cluster (a second cluster has been currently created). All of these servers are interconnected through a Gigabit Ethernet network switch. In addition, the Kubernetes Master has an NFS storage solution.

In the CRAAX lab, 4 real machines have been separated in a different network from our racks for the creation of the Kubernetes nodes.

- The network is 192.168.1.0/24.
- The router is 192.168.1.1.

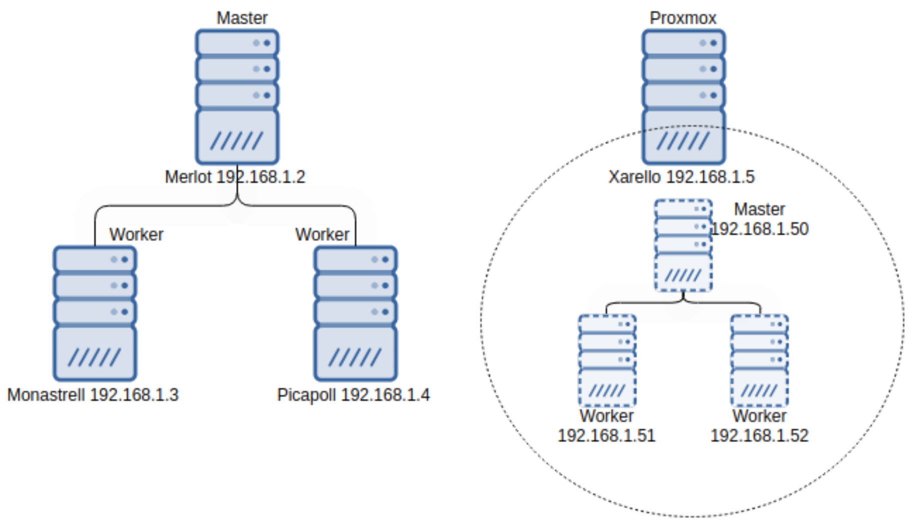

Figure 2 UPC Testbed

In the following table, we can see the characteristics of the UPC servers

Table 2: UPC Server Characteristics

Server characteristics		
	Resources	Operating system
CPU	2 CPU Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	Distributor ID: Ubuntu Description: Ubuntu 18.04.6 LTS Release: 18.04 Codename: bionic
RAM	125 Gb	
SSD	2Tb	

4.2 Hello World Workflow

To validate the functionality of the components deployed as part of this alpha release, the consortium designed a basic workflow that involves all functionalities that are already expected to work. This workflow also presents the basis for integration testing between the different components.

This workflow consists of the following steps to be performed by the different ICOS components:

- **Submission of the App for Instantiation:** The ICOS operator is able to launch the instantiation process by using The ICOS Shell. To do so the operator must provide a valid application description and send the deployment request to the shell backend.
- **Processing of Submitted App for Provisioning:** The Job Manager component receives the application description sent by the ICOS operator and validates such description. Once it is validated it is sent to the Matchmaking component for further processing. Also, the actual Job is created at this step, but this job is not yet eligible for execution.
- **Matchmaking Processing the Deployment Request:** Once the Matchmaking component has received the submitted application description, Matchmaking requests the topology from the Aggregator component. This topology is analyzed within the application requirements present in the mentioned application description to make the best placement decision.
- **Matchmaking Returns Deployment Targets:** Once the placement decision has been made, the Matchmaking component will return it to the Job manager. It is possible to have multiple placement decisions.
- **The Selected Orchestrator Pulls Proper Jobs that it's able to Offload & Execute:** When the Job is populated by the Matchmaking results, the job job is considered eligible for execution. Once that occurs the job is marked as valid for execution and goes to the pile of jobs to be executed. This pile is noticed by the Orchestrator and iterated continuously. When the Orchestrator takes a job to execute it, it will mark the mentioned job as not eligible for execution, but only for a limited amount of time. If such time passes the job is executable again. This prevents multiple Orchestrators from executing the same job and at the same moment gives enough time for such a job to be successfully executed. The Orchestrator that executed the job successfully must notify the Job Manager about the results of the execution within the mentioned amount of time.
- **Orchestrator Specific Deployment:** At this stage, the Job is taken by a specific Orchestrator. For this Orchestrator to be able to execute the deployment contained within the Job it is necessary to perform an adaptation process. The Deployment Manager component is in charge of the mentioned adaptation process, it will describe the deployment to the Orchestrator in a way that it will be able to finally execute it.
- **The Node Agent Pulls the Corresponding Job and Executes it:** Node's Agent is in continuous communication with its Orchestrator in order to pull and execute the deployments that the Orchestrator has taken. This execution is delegated to the specific Orchestration tool.

For more sequential view, Figure 3 presents a sequence diagram of this workflow.

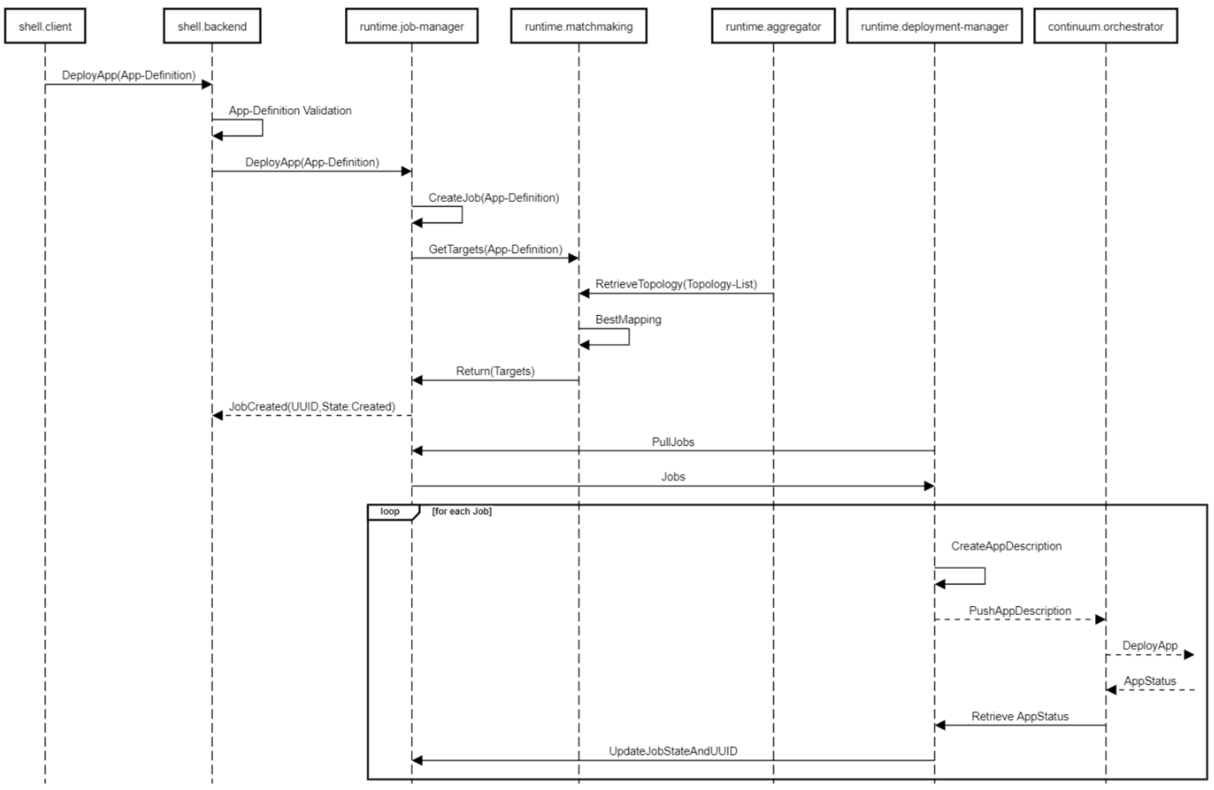

Figure 3 Hello-World workflow

4.3 Technical Performance Assessment and Refinement

During performance evaluation and overall assessment, two individual scenarios will be considered and implemented, one for optimum resource management and the other one for anomaly detection.

In the first case, user story 5 titled “Dynamic re-configuration / optimization of application resources” that has already been described in deliverables D2.1 [4] and D2.2 [1] will be selected for an initial technical performance assessment. This use case will be evaluated in one distinct phase for the alpha release. The participating modules are the following i) network telemetry where data are sent from specific ICOS agents to an ICOS controller. In particular, two agents and one controller will be considered for initial performance evaluation. ii) Data transfer to the data management layer for proper manipulation and pre-processing prior to the ML training, iii) ML model training in the intelligence layer monitoring and evaluation of the selected action.

The first step includes the secure onboarding of both nodes to the ICOS continuum.

In the next stage, simple data transfers will be evaluated in the context of IT-1. It is assumed at this point that no task offloading takes place. Data that are sent to the data management and intelligence layer include the energy consumption in each one of the involved ICOS nodes as well as CPU utilization. Derived performance metrics include i) ML model training time 2) energy consumption of the ICOS involved nodes and iii) average time per task execution. Derived output curves include the load forecasting in the involved nodes.

In the second case, the goal will be to define simple network anomalies in the alpha version from well known models. In this case, abnormal data will be inferred to the network. Predefined abnormal patterns will be included in the traffic time series of the data and which will be defined as unexpectedly high bursts of the load, relative to the average load course. In this case, each ICOS node is properly trained to detect such anomalies and mitigate them. In the Alpha release testing, two particular anomaly detection modules will be used and evaluated: i) decision tree and ii) random forest. Therefore, a full anomaly detection cycle will be implemented and evaluated: a) data transfer to the data management layer b) training the ML model in the intelligence layer c) inferring the trained model to the involved ICOS nodes d) evaluating the response of the specific nodes to the threat.

In this case, performance metrics include i) ML model training time ii) evaluation of positive mitigation scenarios.

Improvements during next iterations: At a later stage, a task offloading model will be deployed per ICOS agent able to select the optimum offloading strategy according to the computational load in the agent and the task complexity. In particular, a deep-Q learning algorithm will be employed with two potential actions: i) execute the task locally on the agent or ii) offload the task to the cloud. In the same context, at a later stage, additional ICOS nodes will be added and the subset of nodes will be selected that minimizes overall power consumption.

5 Development Process

The ICOS project follows a continuous development strategy that aims at optimizing the activities in the project by establishing a continuous development, integration, validation, feedback cycle that allows to run development activities in parallel with integration and validation activities, as well as to reduce the time for newly developed features to be available to use cases for validation. To support this strategy, Work Package 5 defined a set of activities that follows the Continuous Integration Continuous Delivery (CI/CD) practices and guidelines. The objective is to continuously integrate and test new developments without waiting for the ICOS releases (Alpha at M15, Beta at M22 and Final at M36). This will result in a faster and smoother preparation of the ICOS releases and a better software quality.

The overall process at project level is well explained in Figure 4. A continuous integration "loop" is established in technical Work Packages 3,4 and 5 (with the yellow background in the picture), while at each ICOS release a new cycle is triggered that involves the the deployment and validation of ICOS in the Work Package 6 in the project's use cases (with a purple background in the picture).

The remainder of this section focuses on how the first cycle is organized, which process and activities have been defined (section 5.1) and which supporting tools are used (Section 5.2).

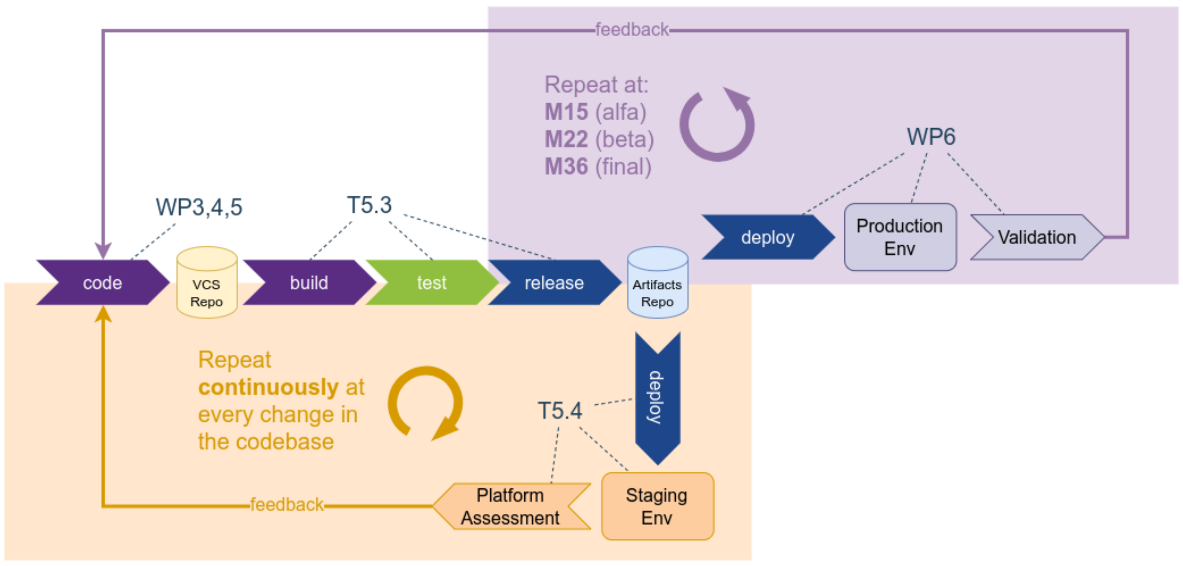

Figure 4 ICOS integration, release and validation processes overview

5.1 CI/CD Process

The integration process aims at i) verifying that the ICOS software produced in the project satisfies the functional and non-functional (i.e. quality) requirements, ii) package the software, iii) release it to WP6 for the deployment in use cases environments.

To achieve this, each ICOS software component developed in the project goes through a CI/CD pipeline: a sequence of different tasks, each of them specialized at a single job (e.g., compiling the code, executing functional tests).

Figure 5 shows the steps of the CI/CD pipeline adopted in ICOS. The single steps are briefly introduced in the remainder of the section.

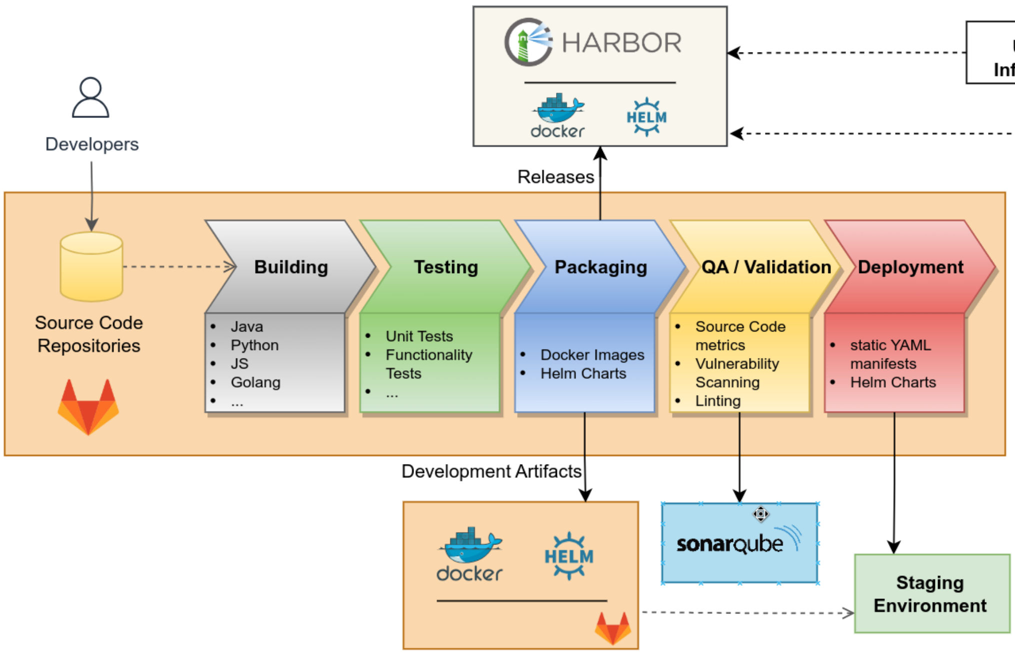

Figure 5 CI/CD Pipeline steps

The Source Code Repositories keep the code of the components developed in ICOS. It represents the synchronization point between the development activities and the integration activities, and the triggers for the CI/CD pipelines. In fact, every time new code for a component is available in the source code repository, a new pipeline is triggered for that component.

Each pipeline is made of the following steps:

1. the **Building** task compiles the source code of the component into binaries artifacts that can be executed on the target platforms;
2. the **Testing** task executes the automated component level tests (e.g., unit tests, functional tests) associated to the component;
3. the **Packaging** task packages the compiled binaries of the component into deployable packages and upload them into the artifact repository;
4. the **Quality Assurance** (**QA)** task ensures the quality of the software running multiple type of validation tasks on the component (e.g., security scanning, software quality measurement, license scanning, password detection, documentation scanning);
5. the **Deployment** task deploys the artifacts in a staging environment where additional tests are carried on.

The enabling factor for allowing the execution of the pipelines for all the components at every change in the code is the **automation**. The entire pipeline is executed automatically thanks to the support of specific tools (presented in section 5.2). This allows to a) reduce the time and the effort for the execution, b) run the pipeline very frequently (at every single change in the code), b) have an immediate feedback on the results.

The full documentation of the CI/CD pipeline is maintained in the WP5 Wiki at: https://production.eng.it/gitlab/icos/support with examples and guides on how to configure it.

5.2 Integration Infrastructure

To support the development and the integration activities, an infrastructure made of multiple resources and tools has been established in the project with the collaborative effort from different partners in the consortium. Figure 6 depicts the structure of this infrastructure as of November 2023. Additional tools and integrations will be evaluated and implemented in the future to satisfy the needs for integrating and testing the upcoming ICOS releases.

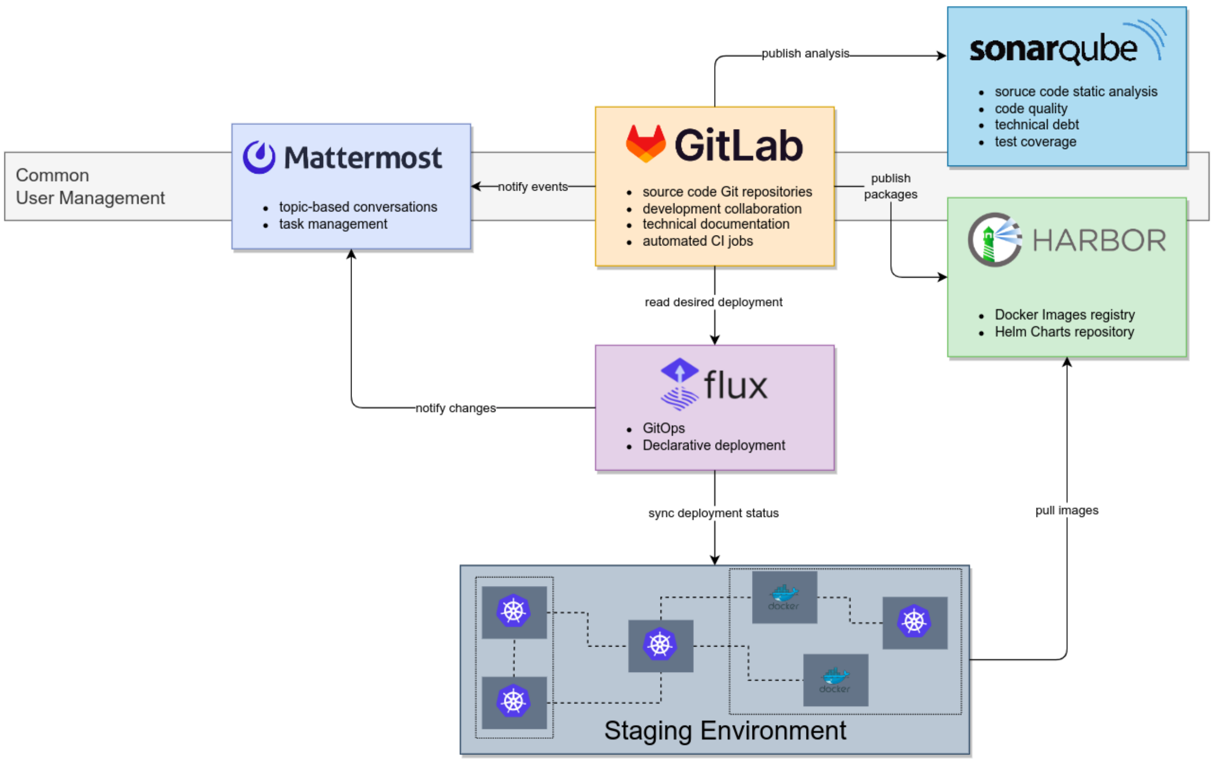

Figure 6 Integration infrastructure

In order to simplify the usage of the infrastructure, management of user identities and accounts is centralized. The registration of new users is a procedure that is defined here: https://production.eng.it/gitlab/icos/support/-/wikis/Users-Management. Once registered, new users will receive a username and password that can be used to access most of the tools in the infrastructure.

The next subsections briefly present the tools, explaining why and how the single tools are used in the project.

5.2.1 GitLab

GitLab is used as the main collaborative DevOps platform in the project. It is managed by ENG and it is available at https://production.eng.it/gitlab. GitLab offers multiple tools that are used within the project.

Source code repositories

GitLab is used to store the source code and technical documentation of the software components developed within the ICOS project. The source code is organized in a tree structure that reflects the ICOS MetaOS structure in layers and functionalities.

Collaborative development

GitLab offers multiple tools to effectively allow developers to write code in a collaborative manner:

- **WEB IDE** to change files directly from the GItLab UI;
- **Branches** and **Merge Requests** to allow collaborators to suggest modification to the source code in a tracked and controlled way;
- **Roles** to assign different permissions to different people in a granular way;
- **Issue Trackers** and **To Do Lists** to manage, assign and discuss technical tasks;
- **Tags** and **Releases** to create references to a specific version of the code;

These functionalities are described in details in the official GitLab documentation: https://docs.gitlab.com/ee/topics/manage_code.html

Technical documentation

GitLab is used to store the API definition for each component (typically in the formal OpenAPI format) as well as its technical documentation. This information is particularly useful to other project's developers that need to use and/or integrate with the component. Having all this information in the same tools and in a standard place and format greatly reduces the communication overhead and the time spent to find it. At the moment, no particular requirements have been imposed on the format and the type of documentation is required for each component, but better rules are expected to be defined for the next ICOS releases.

Continuous Integration server

GitLab offers a code build functionality that allows developers to create pipelines (a sequence of jobs) to automate the execution of different technical steps of the integration of ICOS components (e.g., compilation, packaging, release, distribution). A set of job templates for defining pipelines has been created and are maintained by ENG. The objective of these templates is to standardize the way ICOS components are built and tested, reduce the time and the effort for developers to set-up the CI pipeline for their components.

The templates are available at https://production.eng.it/gitlab/resengit/gitlab/pipeline-helpers, while guidelines on how to use them are available in the WP5 wiki at https://production.eng.it/gitlab/icos/support/-/wikis/Create%20CI%20Pipelines.

5.2.2 GOHarbor

GOHarbor is an open-source OCI registry. It is used in ICOS to store the Docker Images and the Helm Charts artifacts produced by the integration process. This tool is managed and maintained by ENG and it is available at https://produciton.eng.it:8433/icos.

The repository is private and access is possible only using robot accounts that are created by ENG upon request. Accounts are necessary for partners that wish to store/download the images and also for testbeds and automation tools that need to download the images to deploy the ICOS components. More information on the WP5 Wiki at https://production.eng.it/gitlab/icos/support/-/wikis/Harbor.

5.2.3 SonarQube

SonarQube is a tool that collects and analyzes quality related data in software components. ENG deployed and maintains an instance specific for ICOS on its premises at https://sonarqube.res.eng.it/.

SonarQube is integrated with GitLab through a job that runs the SonarQube Scanner[17] for each ICOS component. The scanner analyzes the source code of the component and sends the analysis report to the SonarQube server.

The SonarQube server provides an overview of the status of the project as well as the status of each component on different aspects: **Code Reliability**, **Security**, **Duplications** and **Test Coverage**. It also provides explanations and guidelines for developers to solve the quality issues and improve the quality of their components and, in general, of the ICOS software.

5.2.4 Mattermost

Mattermost is a communication and collaboration tool oriented to development teams. This tool streamline and standardize the technical communications within the project, making technical discussions quicker and open to all developers. It has been selected over alternative solutions (e.g., Microsoft Teams, Slack) for its privacy features as well as the availability of an open-source community edition. It is managed and maintained by ENG and it is available at https://mattermost.res.eng.it/.

The tool supports the one-to-one conversations between two members, as well as the creation of groups (between up to 8 members) or channels (unlimited members). By convention, channels should be created per-topic to keep all the conversations related to the same topic in the same place. A set of predefined channels have been created, one for each work package, to have work package-wide conversations. Additional channels can be created by any member to discuss more specific topics (e.g., testbeds, a specific system functionality, a specific use case).

Few additional plugins have been installed to integrate it with other project's tools:

- integration of the authentication with GitLab accounts;
- notifications for GitLab events;
- notifications for FluxCD events;
- email notifications (through a plugin developed ad-hoc[18]);
- creation of Kanban-like boards to visually organize activities and tasks;
- creation of reminder notifications for specific events.

Further documentation on the tool functionalities and usage can be found in the WP5 wiki at https://production.eng.it/gitlab/icos/support/-/wikis/Mattermost.

5.2.5 FluxCD

FluxCD is a GitOps solution that allows the deployment of software in Kubernetes clusters in a centralized and declarative manner. In WP5, this solution is adopted to streamline the management of the staging testbed. Benefits of this approach are multiple:

- Repeatable, self-documented deployments;
- avoids manual, undocumented changes directly in the testbeds;
- collaborative management of the staging testbed status;

The status of the staging testbed is maintained in GitLab in a specific repository: https://production.eng.it/gitlab/icos/gitops/staging-testbed. This repository contains FluxCD resources that describe the desired status of the staging Kubernetes clusters. FluxCD is constantly watching changs to this repository and everytime a change is detected it is applied to the staging environment. For the ICOS Alpha release only a subset of components in the staging environment are managed using FluxCD (the rest are managed manually), however it will be extended as much as possible in the next releases.

[17] https://docs.sonarsource.com/sonarqube/9.9/analyzing-source-code/scanners/sonarscanner/

[18] https://github.com/ggiammat/mattermost-missed-activity-notifier

6 Conclusions

This document presented the status of the ICOS suites for the alpha release. It described the functionalities and components that were determined to be within the scope of this first iteration, as well as enhancements to be expected for the second release in IT-2. It presented the outcome of all technical work packages so far and forms a starting point for subsequent releases.

To achieve this milestone, all partners collaborated throughout several months to define the realistic scope for the first iteration and to implement them in time. As a result, a limited set of functionalities was successfully integrated and documented for this release, forming a solid baseline not just for the remainder of the project but also allowing for first demonstrations of core functionalities of ICOS.

The tools used for the development strategy adopted within the project have been described as well as the testbed and workflow that is used for the integration tests. Another aspect addressed in this deliverable was the availability of the code as well as some very basic description of the installation process and references to the more in-depth installation instructions.

The project has set up an integrated development environment to allow for continuous development, integration and testing throughout the rest of the project. Leveraging this approach, the ICOS alpha release has been created, deployed and is tested through a Hello-World workflow which demonstrates the basic functionality of ICOS and serves as a proof of concept for most parts of the architecture. The described deviations from this architecture have been described and will further be refined in subsequent deliverables that target the architecture and the code base.

Furthermore, the analysis and design activities are expected to progress, resulting in the architecture of the system evolving and improving further. Therefore, a new iteration will start in the second project period and a new, final, version of the ICOS System architecture will be delivered by M22 and documented in deliverable "D2.4 - ICOS architectural design (IT-2)".

7 References

[1] ICOS. *D2.2 ICOS Architecture Design (IT-1)*. Giammatteo, Gabriele, 2023

[2] ICOS. *D3.1 - Meta-Kernel Layer Module Integrated (IT-1)*. Skaburskas, Konstantin, 2023

[3] ICOS. D4.1 - Data Management, Intelligence and Security Layers (IT-1). Ratkajec, Hrvoje, 2023

[4] ICOS. *D2.1 - ICOS ecosystem: Technologies, requirements and state of the art*. D'Andria, Francesco. 2023.

[5] ICOS Architecture white paper (2023) https://www.icos-project.eu/promotional-material.