D2.2 ICOS Architecture Design (IT-1)

Document Identification			
Status	Final	**Due Date**	31/05/2023
Version	1.0	**Submission Date**	31/05/2023

Related WP	WP2	Document Reference	D2.2
Related Deliverable(s)	-	Dissemination Level (*)	PU
Lead Participant	ENG	Lead Author	Gabriele Giammatteo (ENG)
Contributors	ATOS, NCSRD, PSNC, L-PIT, SUITE5, XLAB, ENG, UPC, SSEA ZETTA, RHT, BSC, CeADAR, SIXSQ, TUBS, NKUA, CRF	Reviewers	Francesc Lordan (BSC)
			Panagiotis Trakadas (NKUA)

Keywords:
System Architecture, Logical Decomposition, Functional Architecture, System Use Cases, Operational Architecture, Sequence Diagrams

Document Information

List of Contributors	
Name	Partner
Alex Volkov	ATOS
Francesco D'Andria	ATOS
Anna Queralt	BSC
Francesc Lordan	BSC
Andrés L Suárez-Cetrulo	CeADAR
Jaydeep Samanta	CeADAR
Ricardo Simón Carbajo	CeADAR
Sebastián Cajas Ordóñez	CeADAR
Gabriele Giammatteo	ENG
Maria Antonietta Di Girolamo	ENG
George Xylouris	NCSRD
Nikos Dimitriou	NCSRD
Anastasios Giannopoulos	NKUA
Konstantinos Skianis	NKUA
Panagiotis Gkonis	NKUA
Panagiotis Trakadas	NKUA
Marcin Plociennik	PSNC
Jose Castillo Lema	RHT
Konstantin Skaburskas	SIXSQ
Nabil Abdennadher	SIXSQ
Francisco Carpio	TUBS
Jordi Garcia	UPC
Sergi Sánchez-López	UPC
Xavier Masip-Bruin	UPC
Hrvoje Ratkajec	XLAB
Tomaz Martincic	XLAB
Ivan Paez	ZSCALE

Document History			
Version	Date	Change editors	Changes
0.1	20/03/2023	ENG	Draft TOC, Executive Summary
0.2	19/04/2023	ENG	First version for Introduction, Methodology and Development process sections
0.3	04/05/2023	ENG, UPC, SIXSQ, RHT, ATOS, XLAB, CeADAR, NKUA, NCSRD, ZSCALE, TUBS	Contributions integrated from partners for functional blocks (Section 4) and interactions analysis (Section 5)

Document History			
Version	Date	Change editors	Changes
0.4	17/05/2023	ENG, UPC, XLAB, CeADAR.	Improved System Overview, Introduction and Conclusions sections
0.5	18/05/2023	ENG	Final version ready for internal review
0.6	25/05/2023	ENG	Integrated feedback from internal reviewers
0.7	30/05/2023	ENG	Finalized last feedback from internal reviewers
1.0	31/05/2023	ATOS	Quality assurance Review. Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Gabriele Giammatteo (ENG)	17/05/2023
Quality manager	Carmen San Román (ATOS)	31/05/2023
Project Coordinator	Francesco D'Andria (ATOS)	31/05/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AD	Architecture Description
AI	Artificial Intelligence
API	Application Programming Interface
CAs	Certificate Authorities
CD	Continuous Delivery
CI	Continuous Integration
CIS	Center for Internet Security
CLI	Command Line Interface
COE	Container Orchestration Engines
CP	Cloud Provider
CRUD	Create, Reade, Update, Delete
DAG	Directed Acyclic Graph
DevOps	Development and Operations
DML	Data Management Layer
DPE	Distributed & Parallel Execution
DPM	Dynamic Policies Manager
DV	Development Viewpoint
DVC	Data Version Control
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
ECIP	Edge Computing Infrastructure Provider
EDA	Exploratory Data Analysis
FSTP	Financial Support to Third Parties
GDPR	General Data Protection Regulation
GPU	Graphic Processing Unit
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
IL	Intelligence Layer
IoT	Internet of Things
IoTP	IoT Provider
IPFS	InterPlanetary File System
JWT	JSON Web Token
LAN	Local Area Network
LSTM	Long Short-Term Memory
MAN	Metropolitan Area Network
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
mTLS	mutual TLS

Abbreviation / acronym	Description
Mx	Month number x
NP	Network Provider
OCM	Open Cluster Management
PMEM	Persistent Memory
RPC	Remote Procedure Call
SDK	Software Development Kit
SLA	Service Level Agreement
SSO	Single Sign-On
SUC	System Use Case
TLS	Transport Layer Security
TSN	Time-Sensitive-Networking
UML	Unified Modelling Language
VAT	Vulnerability Assessment Tool
WAN	Wide Area Network
WP	Work Package
WPx	Work Package number x

Executive Summary

This deliverable presents the initial ICOS System architecture based on the work carried out by the Consortium to identify an architecture for the ICOS System able to satisfy the requirements elicited in the document "D2.1 - ICOS ecosystem: Technologies, requirements and state of the art (IT-1)" and provide the functionalities needed to achieve the project's objectives and to address the project's use cases' needs.

The description of the architecture follows a standard framework based on the ISO/IEC/IEEE 42010:2022 standard to ensure that all the stakeholders and concerns are sufficiently addressed.

Specifically, this document presents the ICOS System architecture from five different points of view: i) a functional point of view where the basic functionalities offered by the system to the user are identified and formalised, ii) a logical view that shows the static structure of the system and decomposes it in functional components, iii) an operational view that analyses the behaviour of the system at runtime, iv) a deployment point of view that describe the distribution of the system in the infrastructures and its topology and v) an implementation point of view that focuses on how the system will be realized.

The document has also the objective of aligning all the partners on a common set of principles for the design of the system and a common vocabulary and methodology to describe the system. This allows to achieve a common understanding of the ICOS functionalities and a common way to design and implement the system throughout the whole Consortium. This will be especially important partners will work in separated teams in the technical Work Packages during the implementation of the system.

The architecture will be updated during the project lifetime according to the feedback received from the development and validation phases. The final version of the architecture will be reported in the deliverable "D2.4 - ICOS architectural design (IT-2)" at M22.

1 Introduction

1.1 Purpose of the document

This document is the main outcome of the work carried out by the project's task "T2.4 – Architectural Design". The main goal of this task is the definition of an architecture for the ICOS System based on the analysis of the requirements, use cases, and objectives expressed in the deliverable "D2.1 - ICOS ecosystem: Technologies, requirements and state of the art (IT-1) [1].

The deliverable defines the main architectural components of the ICOS architecture and describes their functionalities, properties, and interactions, as well as how they address all the functional and quality requirements. The document presents the architecture under five different point of views:

- **a functionality view** that focuses on the functionalities offered by the system from a user perspective,
- a **logical view** that shows the static structure of the system decomposed in architectural components each of them with well-defined functionalities and interfaces,
- an **operational view** that analyses the behaviour of the system at runtime capturing the interactions and the synchronization mechanisms between the architectural components and the messages and data that is exchanged,
- a **physical view** that describes the distribution of the system in the infrastructure nodes and
- an **implementation view** that provides information on how the software components will be realized.

The physical and implementation views will be further analysed and refined in the technical Work Packages when the system will be developed (WP3, WP4 and WP5). The architecture is presented in this document following as much as possible the structure and terminology defined by the **ISO/IEC/IEEE 42010:2022** [2] standard for architectural descriptions. Also views and diagramming languages are standardized (i.e., Unified Modelling Language (UML)). This avoids ambiguities and helps to correctly and immediately communicate the concepts internally and externally to the Consortium.

The document also prepares the future work in the project concerning the design of the architectural components and their implementation. In fact, in addition to the definition of the architecture, that will be used as a blueprint of the system in all technical Work Packages, the document also provides preliminary information to kick-off the work for the implementation of the system like: baseline technologies and assets that will be used, partners responsible for carrying out the implementation work, the integration process and an initial plan for the release of the different system functionalities.

All the Consortium collaborated to the definition of the architecture. This allowed all partners to agree on a common vocabulary, achieve a common understanding of the ICOS functionalities and expected results, as well as have common methodology to design and implement the system. This alignment will be especially important during the development of the system where partners will work in separated teams in the technical Work Packages.

1.2 Relation to other project work

This deliverable presents the first version of the ICOS System architecture. It received inputs from:

- the deliverable "D2.1 - ICOS ecosystem: Technologies, requirements, and state of the art (IT-1) [1] the summarizes the work conducted since the beginning of the project in the task "T2.1 – Ecosystem identification: Baseline technologies" concerning description of use cases, elicitation of requirements and the analysis of the State of the Art for baseline technologies;
- the tasks "T2.2 – Compute continuum requirements definition" and "T2.3 – AI, data management and trust/security requirements" concerning the analysis of the ICOS related technologies, the definition of the project's use cases, and the elicitation of the system requirements.

The document provides a set of Architectural Views required for the coordination of the partners for the design, implementation, and validation of the components of the ICOS system. As such, the primary **audience of this document** consists of the members of the Consortium that participate in: i) the design and development of the components in the technical Work Packages WP3, WP4 and WP5, ii) the integration and testing of the ICOS System in WP5 and iii) the set-up and deployment of the use-cases in WP6.

Additionally, the document is of wider interest to stakeholders external to the Consortium that are active in the domains of Cloud and Edge Computing, including researchers participating and contributing to the Horizon Europe projects under the topics, especially to the ones that are expected to use or extend the results of ICOS in the future.

Finally, this document will also be the basis for the final version of the architecture that will be produced in M22 and documented in deliverable "D2.4 - ICOS architectural design (IT-2)" [3].

1.3 Structure of the document

The deliverable is structured in 7 main sections:

- "1. Introduction": (this section) presents the objectives of the document and introduces its content, structure, relationships with the other project's documents and the glossary.
- "2. Methodology": presents the guiding principles and the methodology used in the definition of the system architecture.
- "3. System Use Cases": provides a formalised list of functionalities that the system will provide to its users.
- "4. Logical View": introduces the structure of the ICOS System. It presents the decomposition of the ICOS System into functional layers and components. It provides a description for each component that includes its role, responsibilities in the architecture and its relationships with other components. In some cases, a preliminary internal design of the components is also provided.
- "5. Operational View": analyses the realisation of some of the ICOS functionalities focusing on what components will be involved and how they will communicate.
- "6. Development process": provides initial considerations, decisions and plans concerning the technical realisation of the ICOS components including technological decisions, integration processes and tools and release plans.
- "7. Conclusions": summarises the content of the document and provides final considerations on how the technical work will continue in the project.

1.4 Glossary adopted in this document

The following table provides definitions of ICOS concepts and artefacts that are discussed in the following sections of this deliverable.

Architecture
An Architecture is an abstraction of the system that identifies its essential aspects like which is its intended use, its main elements and properties, the relationships between elements and between the system and the environment.
Architecture Description (AD)
An Architecture Description (AD) is an artefact that documents the architecture of a system. An AD can take the form of one or more documents, models, diagrams, or simulations. An essential aspect for producing an AD is to know to whom the AD is of interest. The AD of ICOS is mainly constituted by this document (D2.2).
Architectural View
An Architectural View is a part of an AD that expresses the Architecture of the System from the perspective of one or more Stakeholders to address their specific concerns. They are composed of one or more View Components (e.g., diagrams, tables, sections of texts, interviews). This document (D2.2) presents a functionality view, a logical view, an operational view, a physical view and an implementation view.
Stakeholder
Stakeholders are entities that have an interest in the System. They can be individuals, groups, or organizations. Stakeholders use the AD to understand and analyse the system and as guidance for their activities. The AD should address all the interests (concerns) of all the stakeholders. However, doing it with a single model or diagram for a complex system like ICOS is impossible. For this reason, the AD is organized in multiple Architectural Views.
System Use Case (SUC)
A System Use Case (SUC) represents a single functionality that the system can provide to one or more actors that results in an observable result that is of some value for those actors. The term "System Use Case" used in this context is not related to the project's "Use Cases." The former, defined here, represents basic pieces of functionalities that the system provides; the latter identify the four project's use cases described proposal that will be used to validate the ICOS System. When references to SUCs are used in this document, they are expressed with italic and underlined text (e.g., *SUC_CC_1*).
Cloud Continuum
A group of resources (CPU, memory, network, storage, intelligence) managed seamlessly end-to-end that may span across different administrative/technology domains in multi-operator and multi-tenant settings.
ICOS Agent
The basic ICOS software component that needs to be installed on each node (infrastructure node) capable of executing external software and/or providing data and metadata.

ICOS Controller
The ICOS component/software that is responsible for peering with ICOS Agents for providing. control and lifecycle
ICOS Instance
A subset of the CC participating in the execution of a multi-component Application of a certain topology, resource requirements and constraints.
ICOS Node
Any resource, physical or virtual, that is running either the ICOS Software (can be either an ICOS Controller or an ICOS Agent)

2 Methodology

The main goal of the architecture definition activity carried out in Work Package 2 (WP2) is the definition and the description of a high-level architecture for the ICOS System able to capture i) the main components and their properties, ii) their interactions and iii) why and how they can address the needs of the different stakeholders.

The architecture focuses on the system as a whole and analyses its functionalities and interactions with its users and external systems. It does not address neither the internal design of its components nor the specific tools, libraries and languages used for the implementation. These aspects will be analysed and discussed in the project's technical Work Packages WP3, WP4 and WP5.

A well-defined architecture serves multiple purposes in the project:

- have a common terminology and understanding internally in the Consortium (and externally) of what the ICOS System is and what it can do,
- early analyse and address the main risks related to the design of the system,
- organize and prioritize the development activities in the project,
- assign the work to the technical teams,
- have a smooth, efficient, and effective development phase,
- plan the validation of the system (what, how and when functionalities should be validated in the project).

The definition of the ICOS architecture started from the analysis of deliverable D2.1 [1], since it is the main source of information for the ICOS System. It identifies:

- The main actors and stakeholders of the system.
- The main functional requirements.
- How the users expect to interact with the system.
- The main quality properties that the ICOS System should expose.
- Which technologies the system should be compatible with and/or use.

Starting from the analysis of these elements, through a series of activities conducted collaboratively by all partners of the Consortium with virtual and physical meetings, brainstorming sessions, proposal and review of models and diagrams, an initial architecture of the ICOS system has been defined and documented in this deliverable.

During this process, multiple design alternatives have been explored and alternative architectural decisions have been proposed. The different proposals have been evaluated by the Consortium following some guiding principles based on requirements in deliverable D2.1 [1] and on quality characteristics identified by the **ISO/IEC 25010** [4] standard.

The first criteria for evaluating the different alternatives were the **Functional Suitability**: whether the proposed solution satisfies the functional requirements. Then, alternative solutions were evaluated based on how much they addressed the quality (non-functional requirements). This was of particular importance since it is a way to ensure that the final ICOS System will expose specific quality properties of interest. The main ones considered were:

- **Performance Efficiency**: maximize user's workloads performance while optimizing the infrastructural resources usage and energy consumption; minimize system and user data movements.
- **Scalability**: the system should be able to adapt easily and dynamically to changes in the number of devices controlled and the user's workloads.
- **Reliability and Robustness**: the system should be tolerant to failures. The system should be able to predict and detect failures and reconfigure itself for recovering from failures and continue to operate normally.

- **Trust**: following the Zero Trust principle, the trust in the system should be never granted implicitly but must be continually evaluated. Trust should be evaluated end-to-end, from the infrastructures that join the system (verifying their identity and security levels) to the networks, the services, and the users. In addition, trust should be the guiding principle in the design of the AI functionalities.
- **Privacy**: access to data in the system should always be granted based on user and services permissions and regulations (e.g., General Data Protection Regulation (GDPR)); data should always be secured against unauthorized access at rest and in movement.

Finally, although with low priority for the scope of the ICOS system, other quality characteristic has been evaluated like:

- the buildability (ease and efficiency of realization),
- the portability (the ability of the system to run under different execution environments - besides the ones identified in the project),
- the modifiability (the ease of evolve the components accordingly to new/modified requirements),
- the reusability and extensibility (the possibility to reuse and extend the components for usages also beyond the ICOS project).

The resulting ICOS architecture is described in this deliverable adopting standard frameworks, methods and tools for system's architecture descriptions briefly introduced in section 2.1.

2.1 Architecture Description

The process that led to the definition of the architecture as well as the terminology used and the artefacts produced to document it, follows the concepts and the definitions contained in the international standard "**ISO/IEC/IEEE 42010:2022**" [2] (that is a revision of the older "ISO/IEC/IEEE 42010:2011" [5]). This standard specifically addresses the definition of system architectures defining how to approach the activity and how to describe an architecture in a formal way to address all the concerns of the different stakeholders.

The adoption of this standard as main guidance for the definition of the architecture in ICOS allowed to have a common understanding and interpretation of the artefacts produced internally among the consortium and externally to present and disseminate the project architecture to external stakeholders. It also allowed to check that the description of the architecture provided can satisfy all concerns from the different stakeholders.

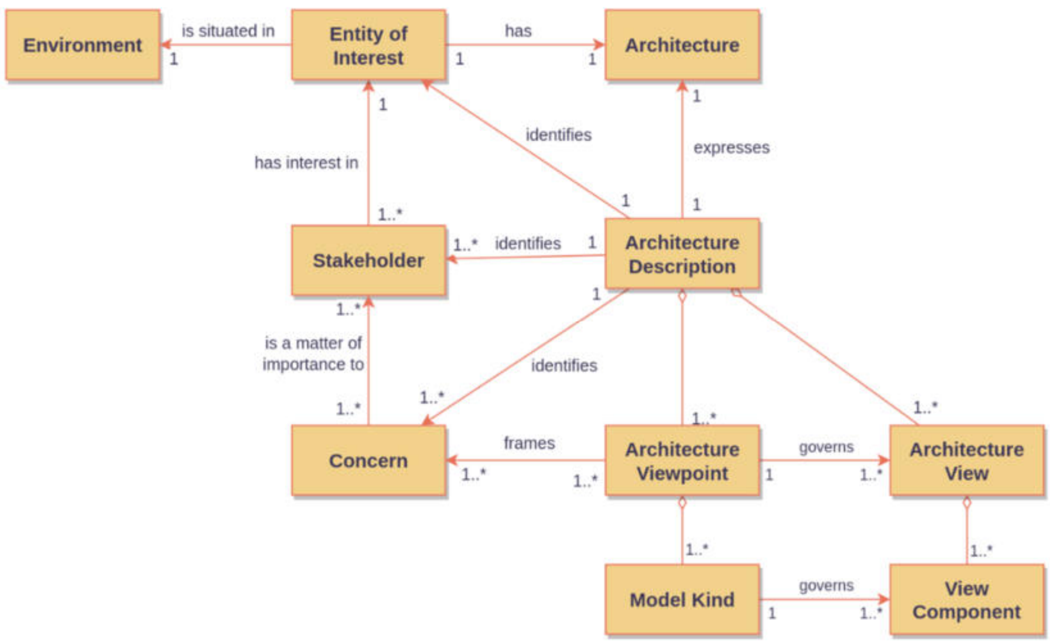


Figure 1: Main concepts defined by the ISO42010:2011

Figure 1 shows the main concepts defined by the standard ISO/IEC/IEEE 42010:2022 and used in the context of ICOS:

- The **Entity of Interest** is the objective: the system for which an architecture is being defined. Each system is situated in an Environment and exhibits an Architecture.
- An **Architecture** is an abstraction of the system that identifies its essential aspects like how its intended use, its main elements and properties, the relationships between elements and between the system and the environment.
- An **Architecture Description** (AD) is an artefact that documents the architecture of a system. An AD can take the form of one or more documents, models, diagrams or simulations and it is the result of the execution of architecting activities. An essential aspect for producing an AD is to know to whom the AD is of interest.
- **Stakeholders** are entities that have an interest (called **Concerns**) in the System. They can be individuals, groups, or organizations. Stakeholders use the AD to understand and analyse the system and as guidance for their activities. The AD should address all the interests (concerns) of all the stakeholders. However, doing it with a single model or diagram for a complex system like ICOS is impossible. For this reason, the AD is organized in multiple Architectural Views. Each **Architectural View** addresses one or more concerns of one or more stakeholders. They are composed of one or more **View Components** (e.g., diagrams, tables, sections of texts, interviews).
- The **Architectural Viewpoints** govern what concerns of which stakeholders are addressed by each Architectural View. Viewpoints act as templates for the Views.

As described by Nick Rozanski and Eoin Woods in “Applying Viewpoints and Views to Software Architecture” [6], using multiple Architecture Viewpoints to describe an architecture brings multiple benefits like the separation of concerns, the communication with multiple stakeholder groups, the management of complexity and the improved focus of developers. However, it also brings some pitfalls like possible inconsistency, fragmentation of the information, selection of the wrong views.

The ISO 42010:2022 [2] standard does not define which Architecture Viewpoints should be used and what they should contain because it highly depends on the system, the stakeholders, and the concerns to address. The standard conceptualizes the existence of **Architecture Description Frameworks** specific for a given system domain and/or community of stakeholders that lists the Viewpoints to use for an AD.

Multiple Architectural Frameworks for software systems have been proposed and considered here to find the best selection of architectural viewpoints to realize: Kruchten's 4+1 Views framework [7], Rozanski & Woods' framework [6], C4 framework by Simon Brown [8], Zachman's framework [9], Siemens Four View model [10], TOGAF [11]. Each of these frameworks proposes its own set of viewpoints, even if there exist several overlaps (e.g., all of them have a Functional - or Conceptual - Viewpoint, almost all of them have the Development Viewpoint (DV).

To document the ICOS Architecture, a set of viewpoints have been selected, mainly inspired by the "**Kruchten's 4+1 Views**" framework. It has been chosen for its simplicity, flexibility, popularity, the previous knowledge, and experience that some partners in the consortium had with it and because it adapts to the type of information and the level of detail needed for the ICOS architecture. The Kruchten's 4+1 model defines five different viewpoints:

- the **Logical** view, which decomposes the system in functional blocks and identifies their properties and relationships,
- the **Process** view, which captures the concurrency and synchronization aspects of the design,
- the **Physical** view, which describes the mapping of the software onto the hardware and reflects its distributed aspect,
- the **Implementation** view, which focuses on the code organization and management and on the development process,
- the **Scenario** view, which describes why and how the end users will use the system. The scenario view is the "+1" in the name of the framework because it guides all the other views.

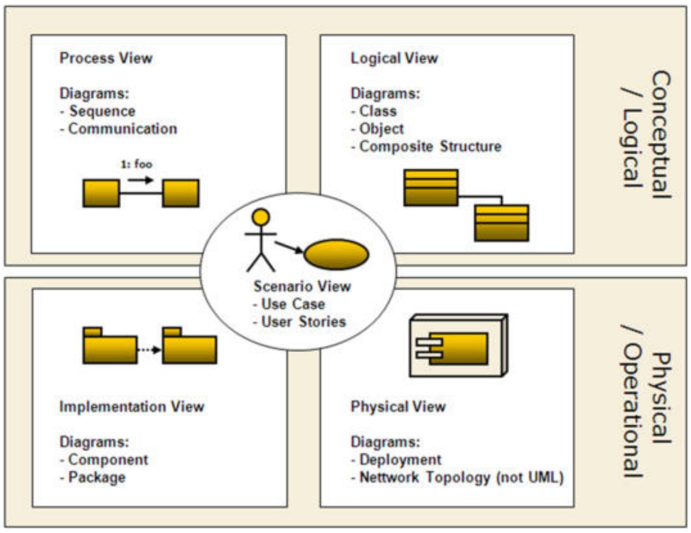


Figure 2: Kruchten's 4+1 framework model[1].

The ICOS Architecture description (this deliverable) follows the Kruchten's 4+1 framework proposing the Scenarios view in section 3, the Logical View in section 4 and the Process View in section 5. The Implementation View and the Physical view are partially presented in section 4 and 6 and will be detailed in future deliverables of WP3, WP4 and WP5 that will define the detailed design of the architectural components.

Besides the artefacts contained in this deliverable, there are other ones of interest to describe the architecture of the system that are contained in other project's deliverables (past or future).

[1] Figure source: https://wiki.cantara.no/display/dev/4+plus+1+View+Model

Table 1 summarizes the main artefacts of architectural interest with a reference to how they map to the 4+1 model, the main concerns and stakeholders addressed and where it can be found in the project's documents.

Table 1: Main artifacts of architectural

Artefact	Concerns	Main stakeholder	4+1 Viewpoint	Reference
User Stories	How do the users interact with the system to perform end-to-end operations? How does the system internally fulfil the user requests?	Users of the system (WP6) System Architects (WP2)	Scenario	D2.1 - Section 3
System Use Cases	What are the functionalities offered by the system to the users?	Users of the system (WP6)	Scenario	D2.2 - Section 3
Overall functional architecture	How is the system decomposed?	System Architects (WP2) Users of the system (WP6)	Logical	D2.2 – Sections 4.1 and 4.2
Functional blocks description	What are the basic modules in which the system is decomposed? How are responsibilities distributed in the system? What are the main logical interconnections between the system blocks?	System Architects (WP2) Development Team (WP3, WP4) Users of the system (WP6)	Logical	D2.2. - Section 4
Sequence diagrams	How do the architectural components interact to achieve the expected functionalities? What are the common communication paradigms in the system? Which integration points are expected between components?	Development Team (WP3, WP4) Integration Team (WP5)	Process	D2.2 - Section 5

Artefact	Concerns	Main stakeholder	4+1 Viewpoint	Reference
Deployment diagrams	How and when are the different components of the system deployed? What is the hardware, software, and network requirements for the deployment of the different components? How are the ICOS components configured?	Integration and Testing teams (WP5) Users of the system (WP6)	Physical	D2.2 – Sections 4.1 and 5 Future WP3 and WP4 Deliverables
Topology diagrams	How are the high-level components (i.e., controllers and agents) of the ICOS System distributed in the infrastructure? How does the connectivity between nodes and infrastructure should be set-up?	System Architects (WP2) Integration and Testing teams (WP5) Users of the system (WP6)	Physical	D2.2 - Section 4.1 Future WP3 and WP4 Deliverables
Detailed design	How is the code of each architectural component organized in packages, modules, services, components?	Development Team (WP3, WP4)	Development	Future WP3 and WP4 deliverables
Integration Process	How is the code integrated, packaged, and released?	Development Team (WP3, WP4) Integration and Testing teams (WP5)	Development	D2.2 - Section 6.2 Future WP5 Deliverables (D5.1)
Assets and Technologies	Which are the baseline technologies used for the implementation of the ICOS System? Which is the technological stack on top of which the ICOS components will be integrated into a unique platform?	Development Team (WP3, WP4)	Development, Physical	D2.2 - Section 6.1 - Table 8 Future WP3 and WP4 deliverables

Artefact	Concerns	Main stakeholder	4+1 Viewpoint	Reference
Responsible Matrix	Who are the partners responsible to deliver each functionality of the system?	Development Team (WP3, WP4)	Development	D2.2 - Section 6.1 - Table 8
Release Plan	When will functionality be delivered in the project's lifetime? What can be tested	Development Team (WP3, WP4) Integration and Testing teams (WP5) Users of the system (WP6)	Development	D2.2 - Section 6.2 - Table 9 Future WP5 Deliverables (D5.1)

3 System Use Cases

This section analyses the ICOS System from the user's point of view. It identifies the main functionalities that the user expects from the system and decomposes them at the level of basic functionalities: the basic interactions with the system that bring a concrete benefit to the user. This activity started from the analysis of the **User Stories and the Requirements** presented in deliverable D2.1 [1]. However, while user stories well describe the functionalities expected from the system, they do it in natural language, with different levels of detail, with duplications, variations, and gaps across different stories. To make this knowledge to be useful as input for the definition of the architecture, they have been analysed and translated in System Use Cases.

A System Use Case (SUC) represents a single functionality that the system can provide to one or more actors that results in an observable result that is of some value for those actors. The objective is to identify in a formal and structured way the set of all basic functionalities that ICOS should provide to its users.

It is worth mentioning that the term "System Use Case" used in this context is not related to the project's "Use Cases." The former, defined here, represents basic pieces of functionalities that the system provides, the latter are defined in the project proposal and represents the demonstrators that will be used to validate the ICOS System.

The actors of an ICOS system have been identified and initially presented in deliverable D2.1 [1]. They are summarized in Table 2.

Table 2: Actors of the ICOS system

Actor	Role
Infrastructure Provider	Owner of computational or networking resources. Resources can be of different types (Cloud, Edge, Internet of Thing (IoT), Network). The main role of the Infrastructure Provider is too on-board the resources in the ICOS Cloud Continuum.
Application Developer	Develops the application components. They can access ICOS resources (e.g., documentation, Software Development Kits (SDKs) to make their application aware of ICOS and use specific functionalities of the platform (e.g., data processing).
Application Integrator	Integrates application components developed by the Application Developers and models the application to be deployable by ICOS (e.g., builds the application descriptor, specifies application requirements). The Application Integrator also interacts with the ICOS System to deploy and manage the application.

Having a set of well defined, agreed, and referable System Use Cases is useful in many ways, not only for the definition of the system's architecture, but also for other project's activities. In fact, SUCs:

- constitute a body of knowledge shared by all partners that ease and make more efficient and effective the communication within the consortium,
- are an input for the definition of the functional architecture of the system,
- provide a link between the requirements and the functional components that is useful for traceability,
- can be used to prioritize the development activities,
- can be used to drive the testing activities and
- can be used to analyse and validate the usability of the system.

To describe the SUCs for the ICOS System in this document, UML Use Cases Diagrams are used. They are a type of UML's behaviour diagrams able to represent a set of actions (the SUCs) and a set of actors and the relationships between them.

The Use Case Diagrams presented in this section use the following elements and conventions (see Figure 3 for the graphical notation of each element):

- The Use Case, represented by an ellipse containing the name of the use case. It is the single functionality associated with the System Use Case.
- The Subject, represented by a rectangle. The subject corresponds to the system (i.e., ICOS) or one of its subsystems (e.g., the Intelligence Layer). A subject rectangle contains all the use cases applicable to the subject, while the actors are drawn outside of the rectangle.
- The Actor represents the role played by an external entity (a human or a software system) that interacts with the subject to generate some value. When the actor is a human, it is represented by a stylized person icon, while when it is a software system (i.e., a layer of the ICOS System) it is represented with a 3D rectangle.
- The association between an Actor and a Use Case indicates that the Actor and the Use Case somehow interact or communicate with each other. This relationship is shown as a straight line connecting the Actor and the Use Case.
- The "include" relationship is a directed relationship between two use cases which is used to show that behaviour of a Use Case is included in another Use Case. The include relationship could be used to simplify large Use Cases by splitting it into several Use Cases or to extract common parts of the behaviours of two or more Use Cases. This relationship is shown as a dashed arrow from the including Use Case to the included Use Case.
- The "extends" relationship is a directed relationship that specifies how and when the behaviour defined in one Use Case can be inserted into the behaviour defined in another Use Case. The extended Use Case is meaningful on its own, it is independent of the extending Use Case. The extending Use Case typically defines optional behaviour that is not necessarily meaningful by itself. This relationship is shown as a dashed arrow labelled with "<<extend>>" from the extending Use Case to the extended one.
- The generalization relationship is a relationship between Use Cases. The child Use Case inherits properties of the parent Use Case and may override its behaviour. The parent Use Case is generally an abstract Use Case: a Use Case which does not have complete declaration (is incomplete) and cannot be instantiated. Generalization is shown as a filled arrow.

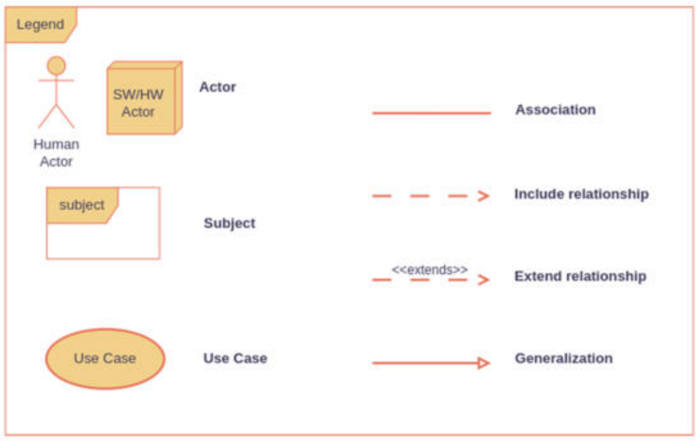


Figure 3: Notation used for the Use Case Diagrams that describe the SUC.

Given the complexity and the extension of the ICOS System, the System Use Cases are presented in different diagrams that group the SUCs based on the layer of the ICOS System they (functionally) belong to:

- the Continuum Management in section 3.1;
- Runtime Management in section 0;
- Intelligence Layer in section 0;
- Data Management in section 3.4;
- Security Layer in section 3.5.

Each section includes a figure that introduces the SUCs identified for the subsystem plus a table that includes:

- An ID for the SUC to uniquely identify a SUC and reference to it in this and another project's documents. When references to SUCs are used in this document, they are expressed with italic and underlined text (e.g., *SUC_CC_1*).
- A brief description of the functionality identified.
- Relationships of the SUC with user stories and/or functional requirements described in deliverable D2.1 [1] to keep trace of the process and the decisions that led to the identification of the system functionalities.

3.1 Continuum Management System Use Cases

Figure 4 presents all the System Use Cases for the management of the continuum. They represent the set of functionalities that the ICOS System should provide to its users to manage an ICOS Cloud Continuum. Six different SUCs have been identified and they are described in Table 3.

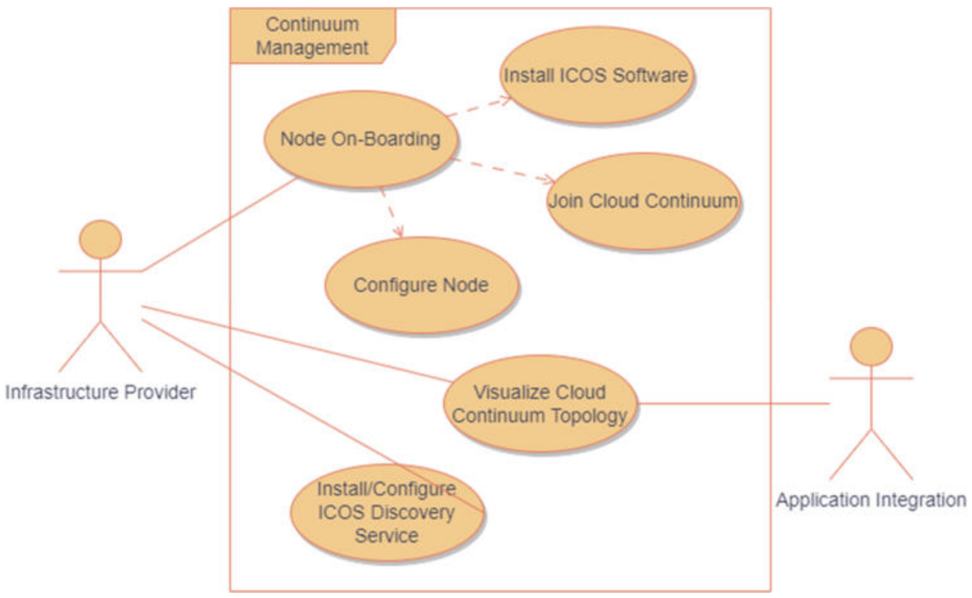


Figure 4: SUCs for the management of the continuum

Table 3: System Use Cases for the management of the continuum

ID	System Use Case	Description	Related User Story	Related Requirement
SUC_CC_1	Node On-Boarding	Allows to add a new node to an ICOS Cloud Continuum	US.1, US.3	CC_FR_04, OP_FR_04
SUC_CC_2	Install ICOS Software	Allows to install the ICOS Controller Software	US.1	
SUC_CC_3	Configure Node	Allows to specify the required configuration to ensure that the new node can join the CC	US.1	SST_FR_01
SUC_CC_4	Join the Cloud Continuum	Allows to make the new node start communicating with other nodes in the Cloud Continuum	US.1, US.3	CC_FR_02, CC_FR_04, CM_FR_21
SUC_CC_5	Visualize Cloud Continuum Topology	Allow the user to graphically visualize the topology of a Cloud Continuum	US.1	CC_FR_03, OP_FR_01, OP_FR_02, CM_FR_21, OP_FR_03, OP_FR_04
SUC_CC_6	Install and Configure ICOS Discovery Service	Allows to set-up a new Discovery Service for an ICOS Cloud Continuum	US.1	CC_FR_02

3.2 Runtime Management System Use Cases

Figure 5 presents all the System Use Cases for the runtime management. They represent the actions that the ICOS System should allow the user to perform to manage the lifecycle of an application (e.g., definition, start/stop, monitoring) in an ICOS Cloud Continuum. In total, eleven SUCs have been identified and they are described in Table 4.

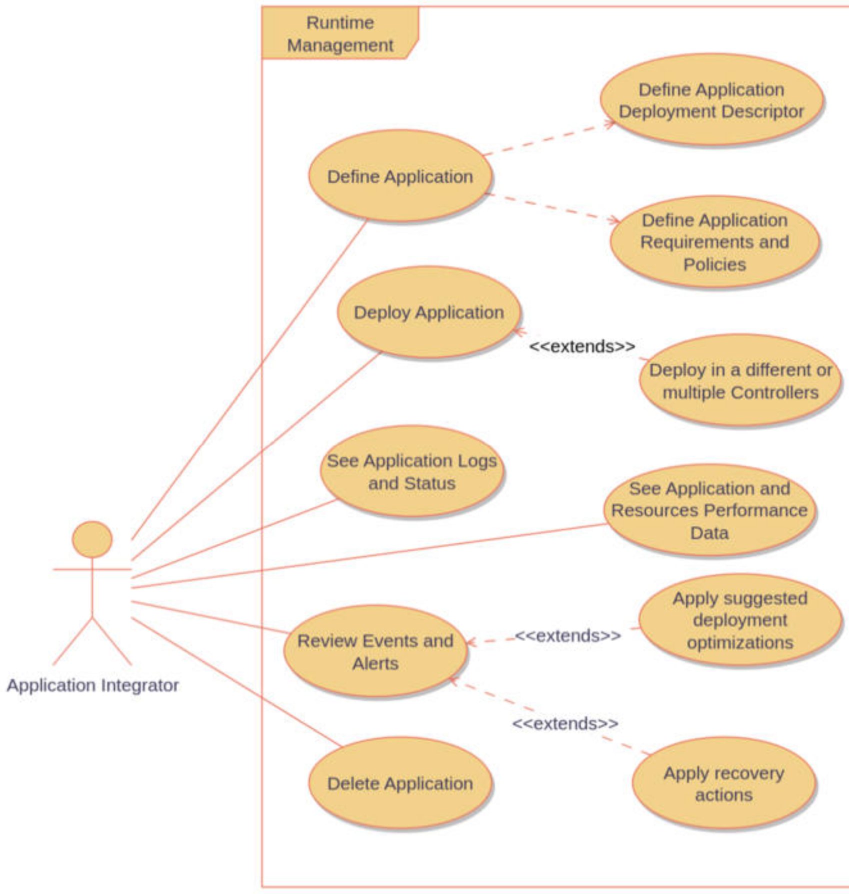

Runtime Management
Define Application Deployment Descriptor
Define Application
Define Application Requirements and Policies
Deploy Application
<<extends>>
Deploy in a different or multiple Controllers
See Application Logs and Status
See Application and Resources Performance Data
Application Integrator
Apply suggested deployment optimizations
Review Events and Alerts
<<extends>>
<<extends>>
Delete Application
Apply recovery actions

Figure 5: SUCs for the runtime management.

Table 4: SUCs for the runtime management

ID	System Use Case	Description	Related Use Story	Related Requirement
SUC_RT_1	Define Application	Allows the definition a new application and all the details for its deployment	US.4, US.7	
SUC_RT_2	Define Application Deployment Descriptor	Allows to describe the deployment of an application (e.g., services and interconnections between them)	US.4, US.7	
SUC_RT_3	Define Application Requirements and Policies	Allows to define a set of non-functional requirements and policies that ICOS should optimize while running the application	US.7	CM_FR_05
SUC_RT_4	Deploy Application	Allows to deploy an application in an ICOS Instance	US.4	CC_FR_01, CC_FR_03, CM_FR_01, CM_FR_03, CM_FR_09, CM_FR_13
SUC_RT_5	Deploy in a different or multiple Controllers	Allow to deploy the application in an ICOS instance composed of multiple Controllers based on user requirements and matchmaking results	US.4	CC_FR_02, CC_FR_04, CM_FR_02
SUC_RT_6	See Application Logs and Status	Review the status and the logs of all the application components	US.4	CM_FR_06, OP_FR_05
SUC_RT_7	See Application and Resources Performance Data	See and query the performance metrics related to the resources that are hosting the application and the metrics generated by the application itself	US.5	CM_FR_06, CM_FR_07
SUC_RT_8	Review Events and Alerts	Review all the events and alerts generated from the runtime, security, and intelligence layer about the application optimization	US.5	CM_FR_07, CM_FR_08
SUC_RT_9	Apply suggested deployment optimizations	Apply suggestions generated by the system to optimize the deployment of the application	US.5	CM_FR_07, CM_FR_08
SUC_RT_10	Apply recovery actions	Apply recovery actions generated by the system	US.5	CM_FR_08, SST_FR_10
SUC_RT_11	Delete Application	Un-deploy and remove an application	US.7	

3.3 Intelligence Layer System Use Cases

Figure 6 presents all the System Use Cases for the intelligence layer. They represent the functionalities that the ICOS System should provide to its users to manage and exploit AI models. Usage of AI is at the basis of the management of runtime resources and the security in ICOS. For this reason, functionalities offered by this layer to the other layers of the ICOS System are identified and listed. A total of ten SUCs have been identified and described in Table 5.

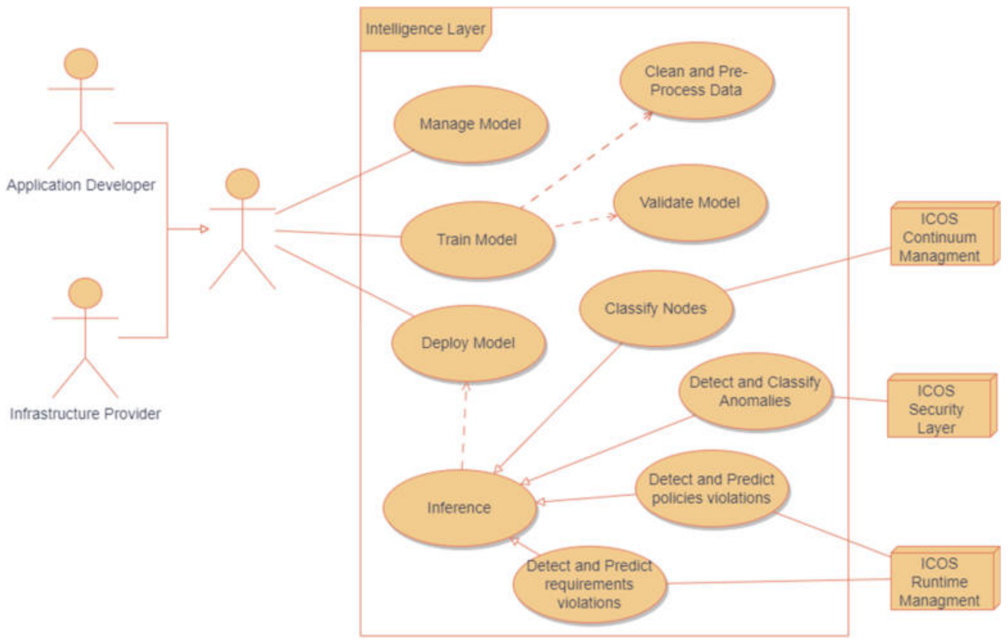


Figure 6: SUCs for the Intelligence layer.

Table 5: SUCs for the Intelligence layer.

ID	System Use Case	Description	Related User Story	Related Requirement
SUC_AI_1	Manage Model	Upload/Modify/Retrieve models	US.5	CM_FR_18, CM_FR_19, OP_FR_06
SUC_AI_2	Train Model	Train a model using resources in the Cloud Continuum.	US.5	CM_FR_14, CM_FR_15, CM_FR_16, SST_FR_04, SST_FR_05, CM_FR_20
SUC_AI_3	Clean and Pre-Process Data	Allows to do the necessary pre-processing of data before training a model	US.5	
SUC_AI_4	Validate Model	Validate a model	US.5	
SUC_AI_5	Deploy Model	Deploys the model where it is needed to make inference	US.5	CM_FR_18, CM_FR_20

ID	System Use Case	Description	Related User Story	Related Requirement
SUC_AI_6	Inference	Execute inference using trained models on data provided. This is an abstract SUC that is implemented by SUCs from SUC_AI_7 to SU_AI_10.	US.5	
SUC_AI_7	Classify Nodes	Classify ICOS Nodes based on their metadata	US.1	CM_FR_17, OP_FR_03
SUC_AI_8	Detect and Classify Anomalies	Detect anomalies analysing the logs of resources/applications	US.8	SST_FR_09
SUC_AI_9	Detect and Predict policies violations	Detect and predict violations of the policies set for an application	US.4, US.5	
SUC_AI_10	Detect and Predict requirements violations	Detect and predict violations of the requirements set for an application	US.4, US.5	CM_FR_17

3.4 Security Layer System Use Cases

Figure 7 presents all the System Use Cases for the Security layer. They represent the functionalities that the ICOS System should provide to its users and to the other ICOS layers to manage the security of users, resources, applications, and data within the system. In total ten security-related SUCs have been identified and described in Table 6.

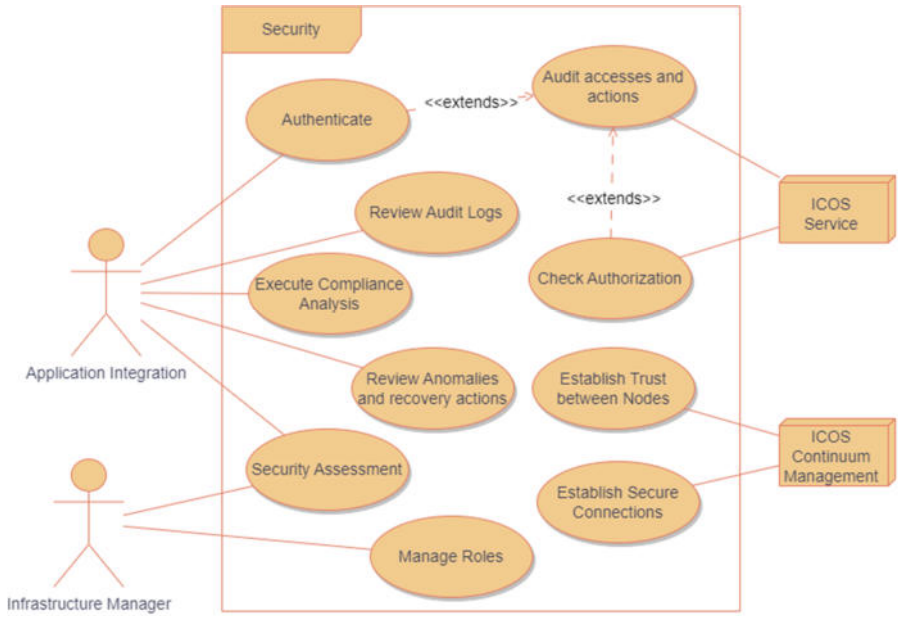

Security
Authenticate
<<extends>>
Audit accesses and actions
Review Audit Logs
<<extends>>
ICOS Service
Execute Compliance Analysis
Check Authorization
Application Integration
Review Anomalies and recovery actions
Establish Trust between Nodes
ICOS Continuum Management
Security Assessment
Establish Secure Connections
Manage Roles
Infrastructure Manager

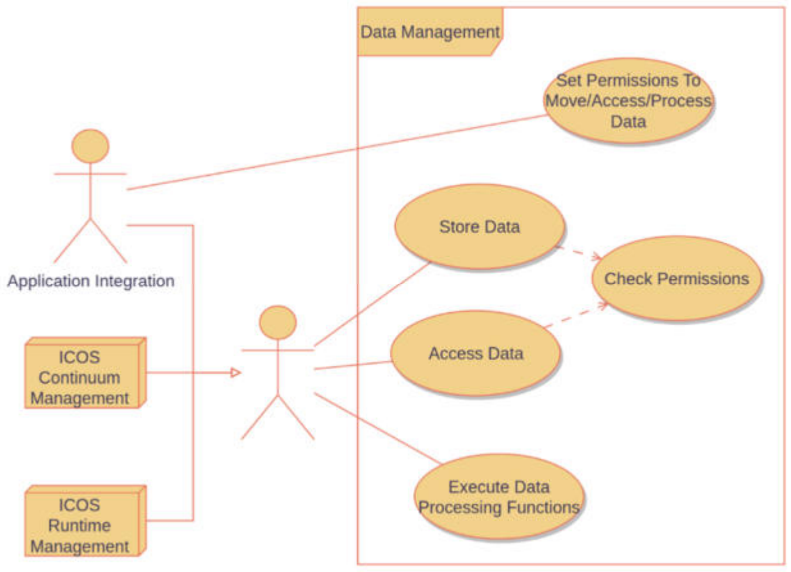
Figure 7: SUCs for the Security layer.

Table 6: SUCs for the Security layer.

ID	System Use Case	Description	Related User Story	Related Requirement
SUC_SC_1	Authenticate	Authenticate and get access tokens	US.8	SST_FR_07, SST_FR_08
SUC_SC_2	Check Authorization	Checks authorization to execute a given operation	US.8	SST_FR_03, SST_FR_07, SST_FR_08
SUC_SC_3	Audit Accesses and Actions	Logs all security related events	US.8	SST_FR_07
SUC_SC_4	Review Audit Logs	Review all the security logs	US.8	
SUC_SC_5	Execute Compliance Analysis	Execute a compliance analysis for an application	US.8	SST_FR_11, SST_FR_12
SUC_SC_6	Review Anomalies and recovery actions	Review the anomalies identified and the suggested actions	US.8	SST_FR_09, SST_FR_10
SUC_SC_7	Manage Roles	Create, modify, and delete users and roles	US.8	OP_FR_09
SUC_SC_8	Security Assessment	Assess the security of the infrastructure and of the code running	US.8	SST_FR_06
SUC_SC_9	Establish Trust between Nodes	Establish a trust relationship between the ICOS nodes	US.8	SST_FR_01
SUC_SC_10	Establish Secure Connections	Establish a trust secured communication channels between nodes that run an application	US.8	CM_FR_22, SST_FR_02

3.5 Data Management Layer System Use Cases

Figure 8 presents all the System Use Cases for the Data Management layer. They represent the functionalities that the ICOS System should provide to its users and to the other ICOS layers to manage the data within the system in a secure, efficient, and distributed way. Five SUCs have been identified for data management and they have been described in Table 7.



Data Management
Set Permissions To Move/Access/Process Data
Application Integration
Store Data
Check Permissions
Access Data
ICOS Continuum Management
ICOS Runtime Management
Execute Data Processing Functions

Figure 8: SUCs for the Data management layer.

Table 7: SUCs Data Management layer.

ID	System Use Case	Description	Related User Story	Related Requirement
SUC_DM_1	Access Data	Access data stored in an ICOS Node	US.6	CM_FR_10, CM_FR_11
SUC_DM_2	Store Data	Store data in an ICOS Node	US.6	CM_FR_10
SUC_DM_3	Execute Data Processing Functions	Execute functions on the data remotely (where the data is stored)	US.6	DRT_FR_03, DRT_FR_04
SUC_DM_4	Set Permissions to Move/Access/Process Data	Set permissions for data usage	US.6	DRT_FR_06
SUC_DM_5	Check Permissions	Checks the permissions before a data operation	US.6	DRT_FR_02, DRT_FR_06

4 Logical View

This section analyses the ICOS System from a logical decomposition point of view. It identifies and describes the main components of the ICOS System and analyses the responsibilities, functionalities, and dependencies of each component. It also shows the organisation of the components into packages, subsystems, and layers to help understand the decomposition of the functionalities.

The logical view of the architecture is extremely useful to the technical team to make the analysis of the system easier by decomposing the system into well-defined and delimited objects (the architectural components) that can be treated separately in the design, implementation and testing phase, making the entire development more efficient and effective. It also allows to identify common functionalities and patterns that can be grouped and homogenised across the whole system. The logical view is also used as the basis for the split of the work for the implementation of the system, assigning different components to different partners and tasks that will be responsible for the implementation.

The presentation of the architecture in the next sections follows a top-down approach presenting the system at three different level of abstraction (similar to the ones proposed by the C4 model [8]):

- A high-level visualisation of the system as a whole in section 4.1;
- A visualisation of the software layers (section 4.2) and, finally,
- The visualisation and description of the components in each layer, from section 4.3 to section 4.8.

4.1 System Overview

ICOS has been conceived as a dynamic metaOS platform distributed along the continuum. One major design principle in ICOS is that it leverages both the cloud (almost unlimited computing and storage capacity, ubiquity, …) and the edge (locality exploitation, latency and communication reduction, privacy preservation, …). In addition, ICOS is elastic, dynamic and mobile, allowing nodes to dynamically join or leave the system according to their own interests, and being able to adapt to eventual system topology changes.

ICOS is made up of two main types of nodes: the **ICOS Controller** and the **ICOS Agent**.

- The ICOS Controller is responsible for managing the continuum (keeping track of the current system topology and availability) and the run-time (launching and monitoring the services execution on demand). This is achieved through a three-layer architectural design, as described section 4.2.
- The ICOS Agent is responsible for executing offloaded users' services, taking care of code execution, data access, telemetry, and, eventually, runtime communication between other Agents. This is further described in section 4.2.

ICOS Agents are distributed along the continuum and include from computation restricted devices at the far edge to high performance computing resources at the cloud level. The ICOS system has been designed to be able to provide a very large number of heterogeneous Agents and distributed over a wide geographical area. To efficiently manage such a huge number of nodes, while exploiting the advantages of locality, a set of ICOS Controllers should be deployed along the continuum to cover the whole geographical area. ICOS Controllers are distributed with a flat, unstructured organisation.

Each Controller manages a set of Agents based on proximity criteria, shaping a highly distributed, flat, unstructured control organisation. Controllers will keep track of the set of Agents assigned within its scope (continuum management) and will satisfy the services requested (runtime management), giving efficient reaction and leveraging locality features of the executed applications and services.

The ICOS multi-controller architecture provides several advantages: it leverages locality issues (latency and communication reduction, privacy guarantee), facilitates system scalability and elasticity (by allowing dynamic and mobile nodes) and becomes fault tolerant (no single point of failure). A typical ICOS scenario can be viewed in Figure 9.

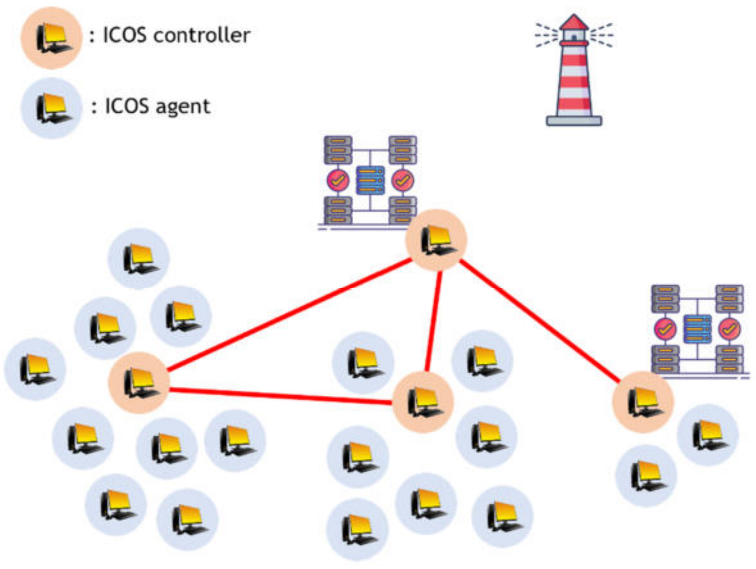


Figure 9: typical ICOS scenario

A number of heterogeneous ICOS Agents are distributed along a wide geographical area. Agents can range from powerful (cluster) to constrained computing devices and provide computing and/or data accessing capabilities. And ICOS Controllers are distributed across the continuum and cover the whole area to create an unstructured organisation. Agents are assigned to Controllers based on proximity principles, as shown in the Figure 9. To simplify and automate the dynamic on-boarding, disconnection, re-connection and migration of ICOS Agents to ICOS Controllers, a new actor is envisioned, called Lighthouse (shown in the Figure 9), that keeps data and references to endpoints that allows ICOS Agents and Controllers to establish a relationship. The **Lighthouse** will be fully described in section 4.8.

4.2 Functional Architecture Overview

ICOS Nodes, as initially defined in deliverable D2.1 [1], are infrastructure elements that will run the ICOS System software distribution and capable of communicating with other ICOS Nodes to realise a unique, distributed, metaOS that will provide all the functionalities of ICOS.

The ICOS System is organised in three main layers: the Meta-Kernel, the Security, and the Intelligence layers, plus two additional modules the ICOS Shell and Data Management as depicted in Figure 10.

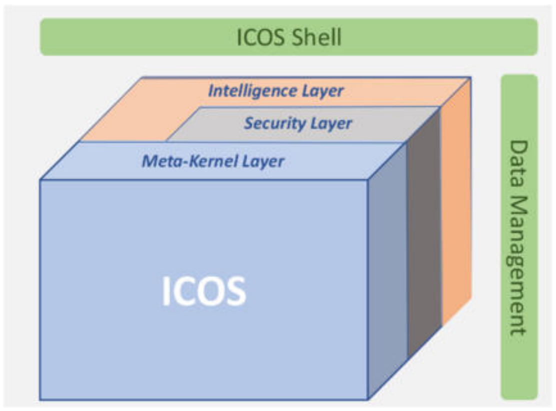


Figure 10: how to be organized ICOS system.

The **Meta-Kernel Layer** is responsible for providing the principal OS functionalities to the continuum (Continuum Management, Runtime Management and Logging & Telemetry). It tightly integrates with the **Security Layer** that has the responsibility of guaranteeing security and trust, as well as with the **Intelligence Layer** that will enrich any action to be taken with innovative AI approaches. **The Security and Intelligence layers** will also closely interact to improve the security-related analysis and actions with AI models. The **Data Management** layer will allow the different components in the system to communicate through secure and distributed data services, while the **ICOS Shell** will provide access to the user to all system functionalities.

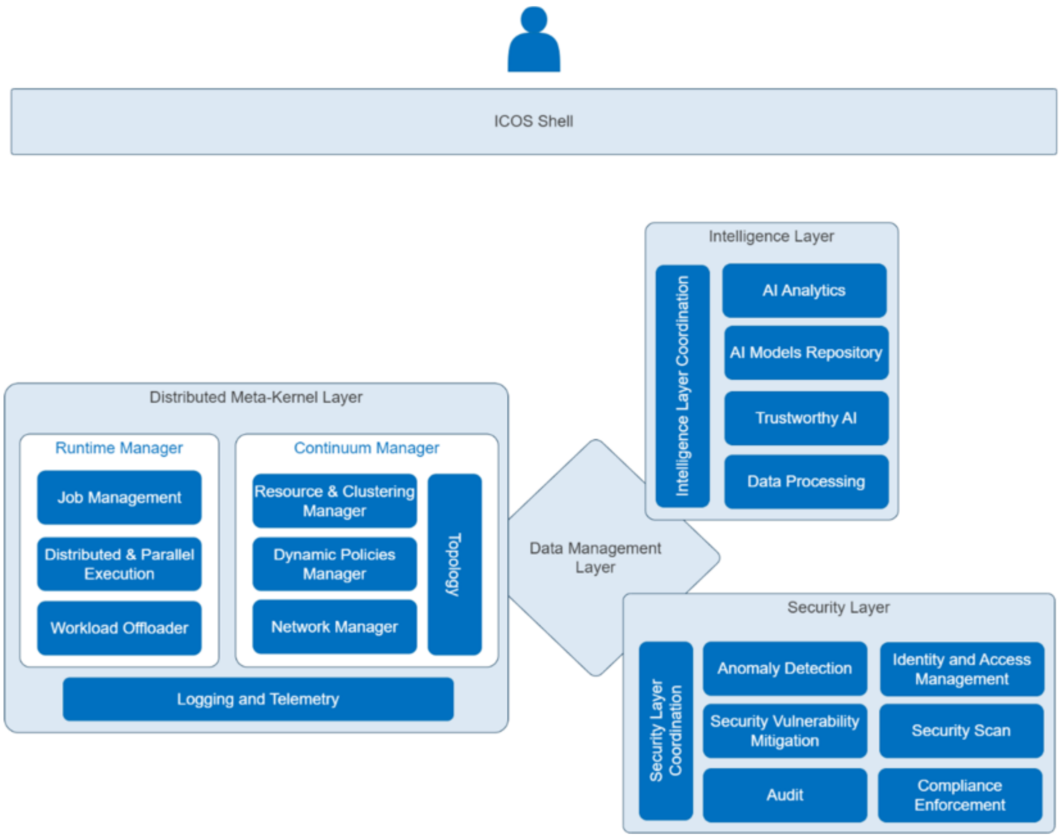


Figure 11: Overview of the ICOS Layers.

As shown in Figure 11, the three layers have been decomposed in multiple functional components. Each component has a well-defined set of responsibilities and interfaces and will realise one or more functionality to the ICOS System. This decomposition allows to manage the complexity of the ICOS System by dividing it into smaller blocks that can be analysed, designed, and implemented separately.

The architecture definition activities in WP2 had the objective of defining these architectural components and identifying their functionalities and their relationships. The following sections will detail the internal composition of the Meta-Kernel Layer (in section 4.3), the Security Layer (in section 4.4) and the Intelligence Layer (in section 4.4).

4.3 Distributed Meta-Kernel Layer

The Distributed Meta-Kernel Layer provides the two base functionalities necessary to make cloud and edge resources manageable and ICOS-ready: the Continuum Manager, responsible for the set-up and maintenance of the Cloud Continuum, and the Runtime Manager which is responsible for the execution of user's applications. In addition, it includes the Logging & Telemetry module responsible for the collection of metrics and logs from the different ICOS Nodes and services.

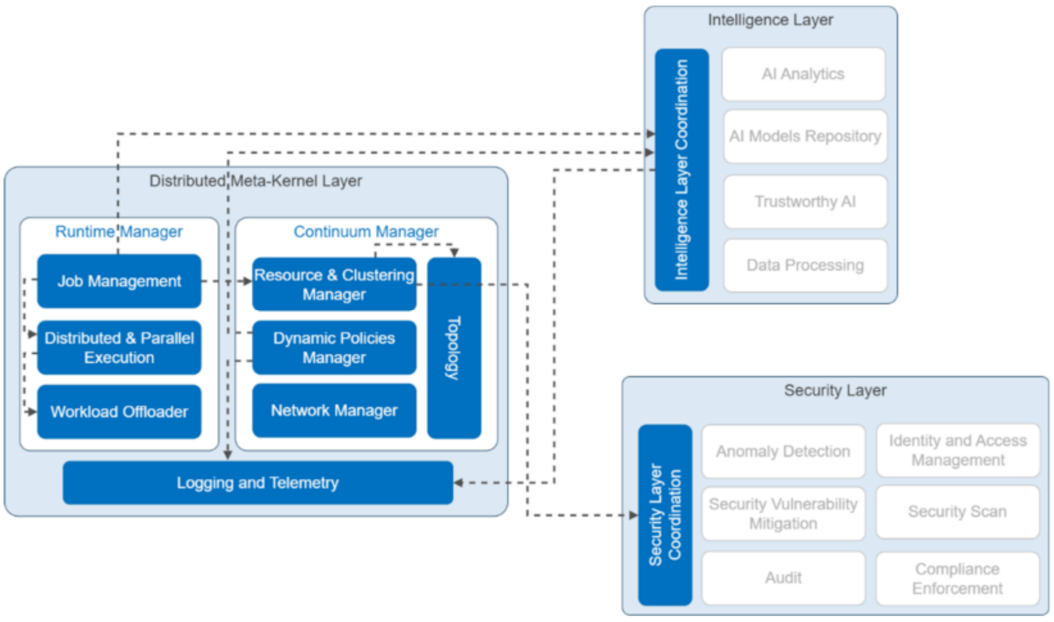


Figure 12: ICOS Meta-Kernel Architecture

Figure 12 shows the internal architecture of this layer and identifies the main intra- and inter-layer relationships. The components and their relationships will be extensively presented in the next subsections.

4.3.1 Continuum Manager

The **Continuum Manager** is responsible for i) registering and configuring resources (cloud resources and edge devices) to the Cloud Continuum, ii) discovering, labelling, and classifying the edge devices and the IoT sensors iii) enabling the infrastructure and operational aspects to fulfil the remote execution of the jobs composing the workload of tasks on the ICOS infrastructure, and finally iv) ensuring that the user workload is performing according to the expected SLA (Service Level Agreement) criteria.

The Continuum Manager is composed of four components distributed across the Cloud Continuum that provides local (to the single ICOS Node) and global control of the resources: the Resource & Clustering Manager, the Dynamic Policies Manager (DPM), the Network Manager, and the Topology.

In the following subsections, the main functionalities of these four components are detailed, while section 5.1 presents the way these components interact to realise the on-boarding of ICOS Nodes.

4.3.1.1 Resource & Clustering Manager

The **Resource & Clustering Manager** is responsible for bootstrapping and on-board new resources to the Cloud Continuum. It provides the mechanisms for i) registering and activating the devices and ii) enabling the infrastructure and operational aspects to fulfil the remote execution of management jobs (issued by users or automatically from the intelligence layer). The remote execution of management jobs also covers the re-configuration of the ICOS Agent, as well as low-level operations such as real time configuration, kernel tuning or updates/upgrades of the ICOS Node.

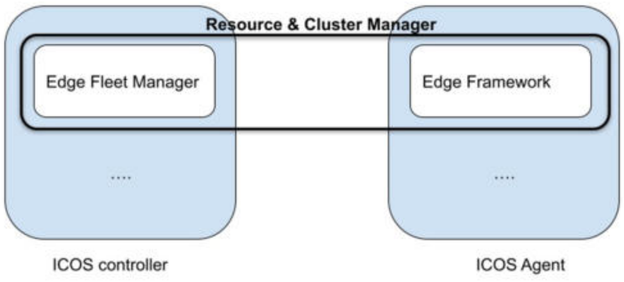


Figure 13: Resource & Cluster Manager architecture.

The Resource & Clustering Manager is composed of two main components (Figure 13), Edge fleet manager and Edge Framework:

- Edge fleet manager: The Edge fleet manager is one of the components of the ICOS Controllers. It creates, configures, deploys, and monitors Edge devices (running ICOS Agents). The Edge fleet manager is an intelligent orchestration service. Edge devices and their IoT devices are mostly autonomous. The ICOS Agents are packaged into some artefacts and provisioned to edge devices via a Cloud repository. Thus, a bootstrapping phase is needed to prepare new edge devices with the needed edge-framework modules. As a best practice, a dedicated control path would be used for the Edge fleet management operations (especially monitoring and telemetry).
- Edge Framework: The Edge Framework is one of the components of the ICOS Agent. It enables edge modules programming and execution. This usually takes the form of a Software Development Kit (SDK), with an application programming interface (API) to drive the application logic, and, possibly, auxiliary tools like a command line interface (CLI) to remotely operate the application. The ICOS ecosystem might provide services to build Edge Framework artefacts and package them as containers.

From an end-user perspective, ICOS will provide the users with an abstract layer that allows them to overcome the complexity of the Container Orchestration Engines (COE). The ICOS Agent will be *"Container Orchestration Engine (COE) agnostic"*: the user has no idea about the COE running behind the ICOS agent. The ICOS Controller is provided with a list of drivers related to the different COEs on which an ICOS Agent can be deployed. During the registration step, the ICOS Controller uses the appropriate driver to configure the ICOS Agent.

4.3.1.2 Topology

The **Topology** component will be responsible to collect, store and maintain updated data related to the ICOS Nodes that join the Cloud Continuum. The data stored for each ICOS Nodes are both static and dynamic data related to i) hardware/software characteristics (e.g., number of CPUs, memory, power status, energy efficiency, installed software), ii) computation, network and/or data capabilities, iii) security assessments, iv) compliance levels.

Data managed by the Topology component has a low update frequency, usually hours or even days. In some cases (e.g., number of CPUs) it does not even need to be updated after the first retrieval. In other cases, like ICOS Nodes that are moving (e.g., in a car), some topology data (e.g., the current location) will need to be updated with higher frequency. This data has not to be confused with data maintained by the Telemetry and Logging component which, on the contrary, reflect the actual status and usage of the resources and has a higher update frequency (usually seconds).

The data managed by the Topology component will be mainly used by the Intelligence Layer to classify the nodes of the Cloud Continuum. This classification will be the main input of the matchmaking phase when a new deployment request arrives. It will also be used to find better deployment schemas to optimise the usage of resources and the services performance. The Topology component could also store historical data (e.g., ICOS Nodes that joined the past, but then disconnected). The Topology component will run both on ICOS Agents (where topology data is produced) and in ICOS Controllers (where data is collected, stored and made accessible to other components).

4.3.1.3 Dynamic Policies Manager

The **Dynamic Policies Manager** component is responsible for the management of the policies related to technical and business performance associated with each application and the detection and prediction of violations of such policies in the running application. The policies will be set by the Application Integrators for each application and will be composed by one (or a combination of) metrics, one (or more) thresholds (that can be also dynamic, and change based on multiple conditions) and corrective actions. Metrics usable by the policies will be the ones provided by the Logging and Telemetry component (e.g., number of anomalies detected, application status, deployment topology). A common model for expressing the policies will be adopted and used to store the policies internally and to transfer the policies in the system.

The component will constantly analyse the monitoring data available to compute the status of all the policies and detect values that violate the thresholds. Violations will trigger actions defined by the policy itself that can range from simple notifications to the user to automated changes in the application deployment to solve the violations. The component will also implement a module to analyse data through machine learning models (managed in the intelligence layer) to predict potential violations on the policies and react accordingly to avoid the violations. The component will interact with the Runtime Manager layer to apply automated corrective actions defined by policies solve/avoid violations. Section 5.5 provides some diagrams that presents these interactions in more detail.

Evaluation of policies and corrective actions will be at different levels: **local** (for evaluation and actions that can be taken ad ICOS Node level – e.g., adjusting CPU frequency based on application load) or **global** when evaluation needs data from multiple nodes and actions involve decisions that affect multiple nodes (e.g., migrate application from one node to another based on the overall performance of the application). For this reason, it is expected that the Dynamic Policies Manager component will run both on ICOS Agents and ICOS Controllers (although monitoring different types of policies).

4.3.1.4 Network Manager

The **Network Manager** component is responsible for defining and implementing the secure overlay network integration that enables all the functional components to communicate between them and data from external sensors and devices to be collected into the system. This component also addresses the aspects related to how the Meta-kernel layer is integrated across multiple and distributed clusters of computing resources. To realise this goal, ICOS will support different communication types, such as:

- **Peer-to-peer communication** is a network exchange model where each ICOS Agent can be both client and server. This flexibility of configuration can be used, among other things, for file sharing, distributed computing, or communication.
- **Brokered communication:** An ICOS Controller can route the data exchange between different ICOS Agents acting as clients and peers in any given topology.
- **Routed communication:** An ICOS's System could support multiple ICOS Controllers cooperating with each other to support the communication between ICOS Agents and clients. A routing protocol will be implemented to automatically forward data to the right place in the ICOS's infrastructure, as illustrated in Figure 14.

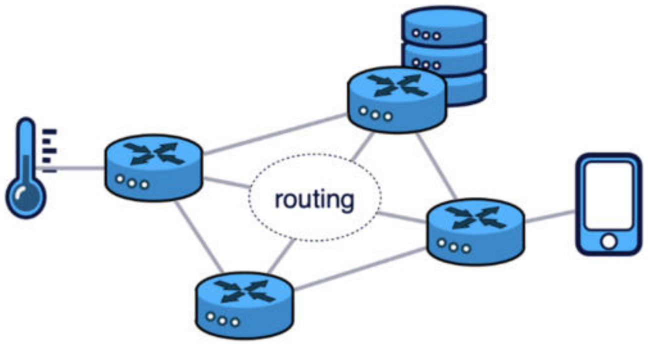


Figure 14: routed communication.

Providing reliable communications over a wide area network with multiple and heterogeneous devices is challenging. Trade-offs need to be made regarding reliability for the system to work, scale smoothly and have sufficient strong guarantees. System's heterogeneity, in terms of network capabilities and node's resources, increases the complexity and has important consequences in the networking mechanisms to be used. Thus, these multiple challenges require proper strategies to avoid that an unresponsive node slows down and impacts the entire system.

ICOS Controller will provide mechanisms to deal with these kinds of challenges by allowing the users to define reliability semantics without extra complexity. The different reliability mechanisms may include a) **one-to-one reliability** where the Network Manager will establish, monitor, and optimise a bidirectional session between two nodes and b) an **end-to-end reliability** (also known as many-to-many) where the Network Manager will manage sessions with many producers and many consumers.

To avoid loss of data due to unreliable or slow nodes, ICOS network manager will support different flow control strategies including:

- Data Receivers control reliability by selecting a resending strategy. They declare if they need missing messages to be resent or not.
- Data Senders and intermediate infrastructure components (ICOS Controllers) individually decide how much memory they are willing to dedicate to reliability. This allows constrained devices to dedicate few resources to reliability while resourceful intermediate routers can dedicate more memory lowering the probabilities of congestion situations.
- Data Senders control congestion by selecting a message dropping strategy. For each sample they decide what should be done in case of congestion (the reliability queue is full) – drop the sample or block the publication. The congestion control strategy is propagated from the sender to all involved infrastructure components and applied along the entire routing path.

4.3.2 Runtime Manager

The **Runtime Manager** is responsible for ensuring the underlying devices to fulfil service requirements deployed from the upper layers of the ICOS ecosystem (i.e., the ICOS Shell) to provide a high-performance execution. The Runtime Manager components will:

- convert the service execution request into a workflow of tasks and generate an execution strategy,
- distribute the execution of tasks into multiple devices and orchestrate their execution,
- enforce optimal allocation policies and react to anomalies adapting the execution.

These functionalities are realised thanks to a collaboration of three main components: the Job Manager that is in the ICOS Controllers and receives execution requests, and the Distributed & Parallel Execution and the Workload Offloader that are in ICOS Agents and control the actual execution. These components will be presented in detail in the next subsections.

4.3.2.1 Job Manager

The **Job Manager** is part of the ICOS Controller, and it is involved in the deployment and the execution of user's services (described in detail in sections 5.2 and 5.3 respectively). The main responsibilities of this component are the initial resources matching, ensure that the execution finishes successfully and returns any eventual result to the user who launched the execution.

The Job Manager component receives a given service from the Application Integrator through the ICOS Shell with its corresponding metadata defining the task requirements (e.g.: specific hardware needs like CPUs/GPUs, location constraints, predefined SLAs, security, data needs/requirements). It then consults the taxonomy/topology component to gather a set of available ICOS Agents and its corresponding compliance policies and performs the matchmaking process (also exploiting the Intelligence Layer capabilities), which finds the best matching that satisfies all constraints.

4.3.2.2 Distributed & Parallel Execution

The **Distributed & Parallel Execution** (D&PE) is a component of the ICOS Agent that aims at exploiting at their best the available computing devices composing an ICOS Instance. To that purpose, the component analyses the logic of the application to divide its workload in different parts that could potentially run in parallel or have dependencies among them. This analysis happens in the first sub-component of the Distributed & Parallel Execution Component, the Access Processor, which with this information dynamically creates a Directed Acyclic Graph (DAG) that represents the workflow of the execution. This DAG is submitted to the second subcomponent: the Task Scheduler.

The Task Scheduler subcomponent aims at optimising the execution of the DAG by orchestrating the execution of as many tasks in parallel on the available ICOS Agents while guaranteeing the fulfilment of the dependencies among them. To fully exploit the computing capacity of the infrastructure, the Task Scheduler interacts with the instances of the Distributed & Parallel Execution component in other Agents to be aware of the workload on the remote devices, and thus, achieve a better balance of the workload. The Task Scheduler can optimise the execution according to different configurable policies such as pursuing the shortest execution time, achieving the lowest energy consumption, or reducing the amount of data transferring.

If the Task Scheduler decides that the Agent is to host locally the execution of one task, it finds the most suitable computing device in the node available to run its workload; otherwise, if it decides to run the execution in another Agent, the Distributed & Parallel Execution component will interact with the Workload Offloader to submit the execution onto the remote Agent.

4.3.2.3 Workload Offloader

The **Workload Offloader** is also a component of the ICOS Agents whose purpose is to enable the sharing of application-level workload among different Agents belonging to the same ICOS Instance. On the offloader side, the component will receive the request to offload some workload from the Distributed & Parallel Execution component of the local Agent and will find the most suitable mechanism to delegate the execution on the remote Agent. Likewise, on the receiver side, the Workload Offloader will contact the local Distributed & Parallel Execution component upon the reception of the request to exploit its inner parallelism. Both Workload Offloader components will cooperate to enable the remote monitoring of the execution progress.

4.3.3 Logging and Telemetry

Monitoring a large and diverse set of distributed computing resources and software generates a large amount of data of various types and may require data to be analysed, pre-processed, and aggregated if necessary, and transmitted from *edge nodes / IoT devices* where monitored services are deployed to locations where end-to-end services and resources management decisions are made.

To prevent resource and application monitoring data from further contributing to the data overload, intelligent mechanisms are needed to automatically and dynamically decide what data to collect (i.e., what type of monitoring data to measure), the granularity of the data (i.e., what level of information to collect for a given item), and the frequency of the data (i.e., the interval between two collections for a given item).

The **Logging and Telemetry** component defines and implements the ICOS cloud-native monitoring and observability strategies. It facilitates the collection of telemetry reports from all services, providing vertical application operators with a single view of performance, errors, logs, and component availability, integrating all functional components across multiple and distributed clusters of computing resources.

Its goal is to effectively run and operate an end-to-end, multi-tenant, easy-to-operate, scalable observability system on the ICOS ecosystem. It is independent of the type of observability data. The component enables the ingestion, long-term storage, and use of common observability signals such as *Quality of Service* (QoS) metrics (performance and availability), logging and tracing the entire continuum under a single consistent system with well-defined tenancy APIs and signal correlation capabilities.

As shown in the Figure 15, the objective of this component is to develop a scalable, configurable, global, metric stack that can be run locally (in each ICOS Node) as well as in a central location aggregating data from multiple nodes for monitoring and telemetry purposes. As depicted in Figure 15, while all nodes will run the Logging and Telemetry module and will be capable of collecting, store and query metrics, a centralised module of the Logging and Telemetry component will aggregate data from different nodes and will make possible global querying and analysis.

This component will offer an API to perform parametric queries and retrieve metrics, logs and traces stored in the system. This API will be extensively used by multiple other components in the system that needs to analyse data (e.g., AI Analytics, Anomaly Detection) or get the status of the system and/or applications (e.g., Dynamic Policies Manager) to get local and global decisions.

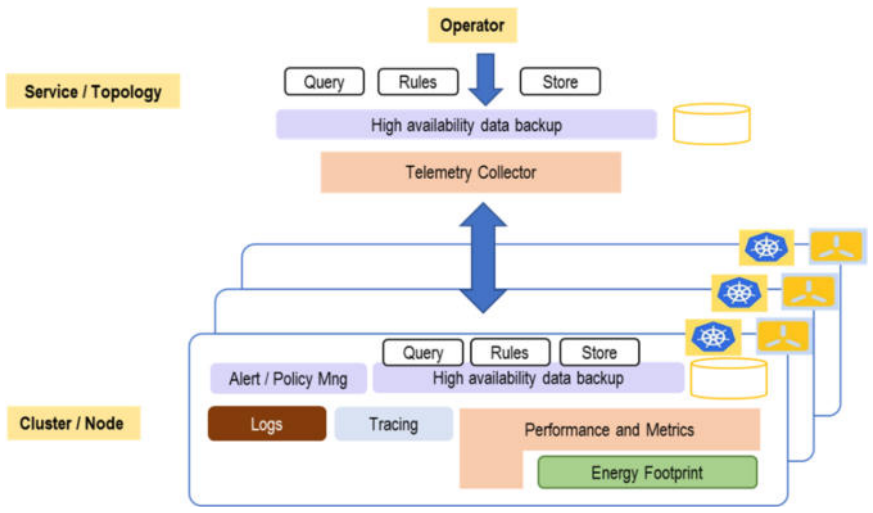


Figure 15: Telemetry and Logging architecture

4.4 Intelligence Layer

This section describes the Intelligence Layer. The Intelligence Layer is responsible for delivering capabilities to facilitate the training, testing, deployment, maintenance, and updating of analytical and machine learning models across the edge/cloud spectrum. Its objective is to bolster and enhance the functions and efficacy of the security and meta-kernel layers while adhering to particular data and model usage policies, focusing on ensuring trustworthiness.

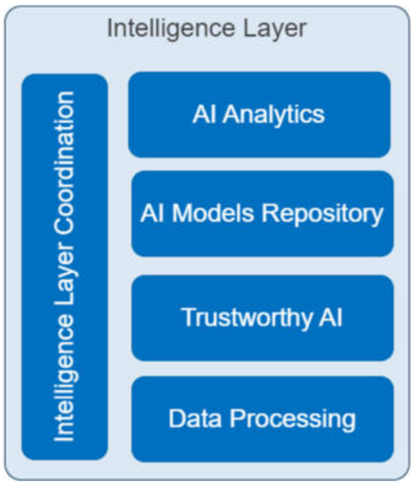


Figure 16: Intelligence Layer

This layer, as seen in Figure 16, contains the following modules:

- The **Data Processing** module aggregates libraries and frameworks to allow data processing and transformation at scale in various devices ranging from cloud to low-end devices using the ICOS ecosystem. It allows data Access & Storage and pre-processing using the Data Management module (batch, stream, local, remote). This module will run distributed in the ICOS Agents.
- **AI Analytics** aggregates libraries and algorithms to train and forecast using state-of-the-art methods to achieve a smart meta-kernel and support project use cases.
- **Intelligence Layer Coordination** is an interface to communicate to the modules of the intelligence layer and provides coordination between the meta-kernel and user layers providing and requesting services. Then this API, located in an ICOS Controller, will forward AI or data processing workloads to ICOS Aagents.
- **AI Models Repository** refers to ICOS repositories to store trained models to make them available to the public and successful use cases deployed in the OS. This repository is accessed from the Controller.
- The **Trustworthy AI** component considers data privacy concerns and uses data anonymisation and federated learning for this. Additionally, it provides functions to make models explainable and transparent, explaining what leads to a decision or forecast. This module will involve AI-related and Explainable AI workloads in agents.

The diagram shown in Figure 17, illustrates the relationships between modules of the AI layers and from the AI layer to Security, Data Management layers and APIs for Meta-kernel modules such as the Continuum Manager. The access point to the Intelligence Layer, as for Data Management and the Meta-Kernel layers, will be using APIs.

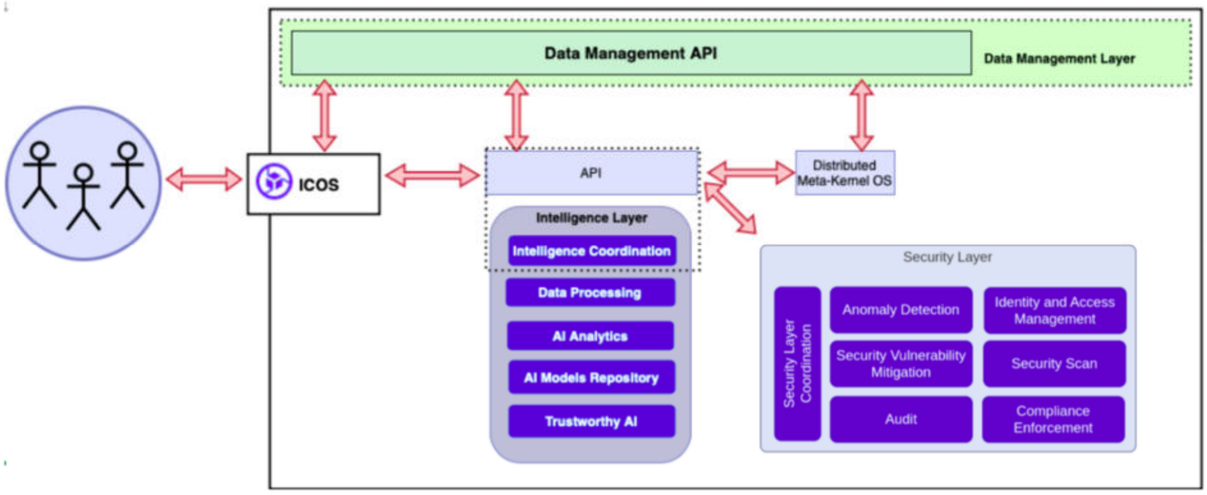


Figure 17: Interactions between the Intelligence Layer and other layers.

AI Analytics, Intelligence Layer Coordination, AI Models Repository and Trustworthy AI components are covered in Subsections 4.4.1 to 4.4.4.

4.4.1 Intelligence Layer Coordination

The AI Coordination module facilitates optimisation, predictive analytics, and applying machine learning models across the edge-cloud continuum. It entails implementing policies for utilising, sharing, and updating models, including adopting federated learning strategies. This acts as an interface and provides coordination between the meta-kernel and user layers providing and requesting services. This component helps to coordinate with other intelligence layers of the same domain in the continuum, providing ICOS with the ability to learn collaboratively.

It provides a programming interface for the ICOS Models Repository (Marketplace in the proposal), allowing to push or pull trained models on demand. It maintains and provides metadata as model descriptions in the repository, algorithms in the AI APIs and statistics about the usage of the interfaces, Intelligence Layer modules and AI APIs.

This component also maintains and provides policies for sharing, updating, and using models in the edge-cloud continuum, graphical interfaces for model monitoring and experiment management and APIs for MLOps and storage of the models are also coordinated by this layer.

Figure 18 illustrates the interactions of this module with other modules of the layer and other layers.

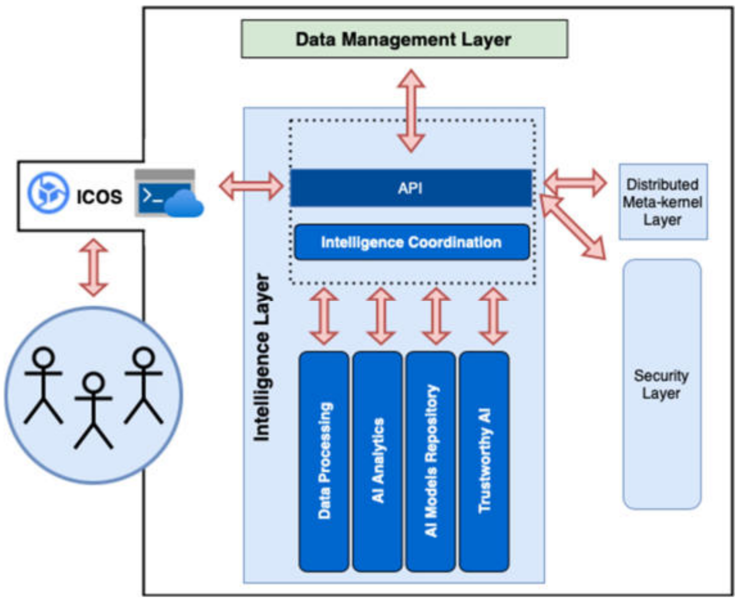


Figure 18: Interactions of the Intelligence Layer Coordination Module

The AI Coordination module is the central module of the intelligence layer, and any request incoming to the other four modules of the layers is received by the API developed in this and redirected to the relevant module. For instance, if a model is requested for inference from the meta-kernel, the relevant module in the meta-kernel will raise a request to the API of the AI coordination module, which will be redirected to the relevant model in the AI Analytics module.

The Intelligence Coordination API runs in the controller an receives incoming requests for rest modules of the Intelligence Layer. Such requests are then forwarded to agents to run distributed workloads on demand. The API distributes ML and processing workloads by using the programming concept of 'Runners'[2]. A 'Runner' represents a unit of computation that can be executed on a remote worker and scales independently, using agent's Graphic Processing Unit (GPU) if available and needed for the computation.

In scenarios where a model needs to be first retrieved from a repository, the Coordination module will ping, from the Controller. the repository-related libraries developed in the AI Models module.

The Coordination module will be responsible for requesting the Data Management Layer to store or retrieve a model saved as an object. The AI analytics module will interact with the Coordination module to request or save a model, which acts as a middle agent with the Data Management Layer.

The process will behave similarly internally for the Data Processing and Trustworthy AI modules and externally to the shell and security layers. For example, the security layer will perform operations with or to models using the AI Coordination module API.

4.4.2 AI Analytics

This component comprises a selected stack of streamlined machine learning algorithms and pipelines tailored to train and test predictive and optimisation models. These are executed as workloads on ICOS Agents. It encompasses deep learning, adaptive and static batch machine learning, traditional machine learning, transfer learning and reinforcement learning libraries optimised to function efficiently on constrained devices. It considers models for both structured (e.g., time-series) and semi (nested objects) or unstructured data (Text, Images). Advanced techniques, such as transfer learning, are to be considered

[2] https://docs.bentoml.org/en/latest/concepts/runner.html

in the second version of the module. The AI Analytics module will enable the proliferation of Artificial Intelligence/Machine Learning (AI/ML) algorithms aiming to empower autonomous network management, heavily facilitating, and automating the orchestration of the network resources. Automatic management of AI/ML workflows and life cycles can act as a significant subcomponent towards ICOS automation.

The development of AI/ML algorithms requires a set of tools to facilitate building, training, evaluating, and serving procedures. In this module, ICOS may make use of cutting-edge and open-source platforms for supporting end-to-end AI/ML capabilities, facilitating open, programmable, transparent, and interoperable AI/ML model training, validation, and monitoring. The *AI Analytics* component will act as a platform for building and managing AI/ML workflows in production-ready environments. It should provide a set of components for building an AI/ML pipeline, a set of libraries that provides functionalities to the different components and a toolkit for building the pipeline that can be orchestrated with several platforms (e.g., Kubeflow[3] or Apache Airflow[4]). An AI analytics block diagram defines the series of operations performed in the AI/ML workflow.

Figure 19 depicts the set of several AI Analytics components.

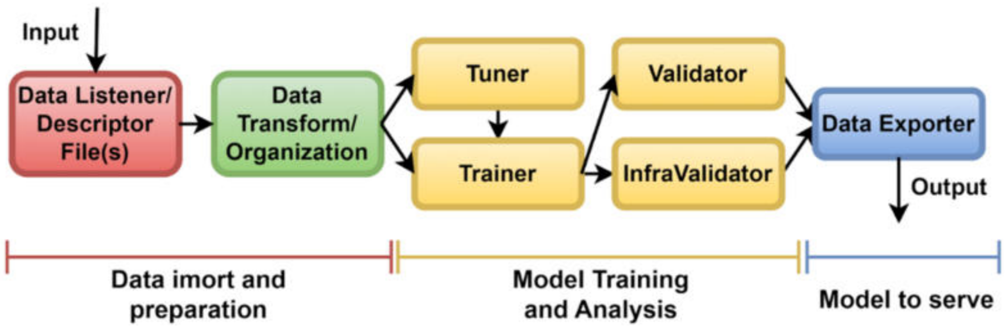

Input
Data Listener/ Descriptor File(s)
Data Transform/ Organization
Tuner
Trainer
Validator
InfraValidator
Data Exporter
Output
Data imort and preparation
Model Training and Analysis
Model to serve

Figure 19: Block diagram of AI analytics for end-to-end AI/ML delivery[5].

As shown in Figure 19, the pipeline is divided in three general phases, namely the ingestion of data, training of the model and model deployment. The starting component of the AI/ML analytics block diagram is related to the input and data import. This block will have proper interfacing with the sources of training data, such as the Data Processing component and the Intelligence Coordination Layer. The internal functionalities of Data Listener are in charge of ingesting the data and, potentially, splitting into training and testing datasets. Other analytics and statistics related to the considered AI/ML dataset can be also provided by the Data Processing component. Data transformations, normalizations and any other feature engineering procedures on the datasets take place in Data Transform to allow organization of the data for the upcoming training. Training of the model can be then executed in the Trainer, in collaboration with the Tuner being responsible for hyperparameter stabilisation of the AI/ML model. Then, the Validator evaluates the training results, makes inferences on unseen data samples to validate the AI/ML models and ensure the required performance for production. Prior deployment, InfraValidator checks whether the model is servable from the existing infrastructure. Finally, the Data exporter exports the model in a servable format.

Once the model is developed and exported in a servable format and the model is deployed in a serving infrastructure, the AI Analytics module should offer model serving capabilities as a flexible, high-

[3] https://www.kubeflow.org/
[4] https://airflow.apache.org/
[5] https://www.tensorflow.org/tfx

performance serving system for production environments (e.g., TensorFlow[5] serving). This model serving module can be installed natively with pip, apt or it can be built from source or executed in a Docker[6] container. Once the model serving instance is running, it can serve one or multiple models. As a result, it does not require to build the original model in every run, simplifying the sharing and deploying procedures. Optionally, the model serving module can support inference requests to supplement AI Coordination functionalities, allowing the access to the serving models in a language-agnostic manner.

The AI Analytics module should offer the Model Analysis component, which provides a mechanism for model evaluation. This will also allow model evaluations in the AI analytics pipeline and viewing resultant metrics and plots. More specifically it can provide metrics computed on training and testing dataset, monitoring metrics over time, modelling quality performance on different feature combinations, and last model validation for ensuring that the model keeps a consistent performance.

This module will interact with the AI Coordination module as all incoming requests to apply AI algorithms will be passed by and to the AI coordination API. In addition, the first part of the AI analytics module, covering the ingestion and data validation, may be covered partially or holistically by the Data Management module. In this case, pre-processed data provisioning will be achieved through direct or indirect (through AI coordination module) interactions between data management and AI analytics modules. Moreover, AI Analytics shall support interaction with the Security Layer (especially in the case of anomaly detection alarming to collect and prepare for training appropriate ML algorithms). Finally, the main sources of input of the AI Analytics module are obtained upon interactions with the Logging and Telemetry component, located in the distributed meta-kernel layer (where the data are collected and stored).

4.4.3 AI Models Repository

The ICOS Model repository offers a collection of pre-trained analytics and ML models that may be reused, updated, modified (e.g., transfer learning), and integrated to promote the deployment of novel AI approaches across the ICOS meta-OS levels. This repository is accessed from the ICOS controller.

It includes capabilities for training and compressing these models for use in constrained devices (e.g., pruning unused branches in trees or simplifying Neural Network architectures). This repository will make it easier to store pre-trained models of different versions that can be pulled and integrated with the existing infrastructure. It will also make it easier to improve models and push them back into the repository. This repository will allow managing all algorithms and libraries used in the various versions of the meta-OS and will be maintained in such a way that, based on the device constraints, only the essential functions will be fetched at installation or update time.

ICOS should provide a version control system, such as Git[7], for maintaining machine learning models, which is optimised for handling large files and complex file dependencies, such as datasets, models, and code. Moreover, it should also be able to handle data pipelines which can be easily versioned to reproduce workflows better. This will enable data scientists and machine learning engineers to ensure that their models are reproducible. This is crucial for assuring the dependability and trustworthiness of machine learning models since it allows model findings to be replicated and confirmed by other users. ICOS model repository should be able to make it simple to maintain and manage the versions of datasets, models, and code, ensuring that the same inputs and parameters are utilised for training and evaluating a model.

The goal of this module is to provide a platform for managing, storing, and maintaining pre-trained ML models, the codebase, and data, as well as an easy-to-use interface for uploading and retrieving machine learning models, allowing users and data scientists to create a repository for their machine learning

[6]https://www.docker.com/resources/what-container
[7] https://git-scm.com/

models. This platform will allow data scientists to maintain and manage file dependencies such as those between a dataset and a model or between various versions of a model. This can help with project management and collaboration with other data scientists and machine learning developers.

To encourage the development of optimised models suitable for Edge-cloud computing, the ICOS model repository can simplify the development and storage of numerous models and their variants with respect to each User scenario. These improved models can be made available to users who download the operating system and use it to develop their own applications.

ICOS Model Repository will be able to communicate with the AI layer coordination modules and Data Management layer. The Data Management layer can offer the datasets needed to train and improve the models, while the AI coordination module can provide the availability of the most recent models. The following figure illustrates the interaction between the ICOS repository with the AI layer coordination module and the Data Management layer.

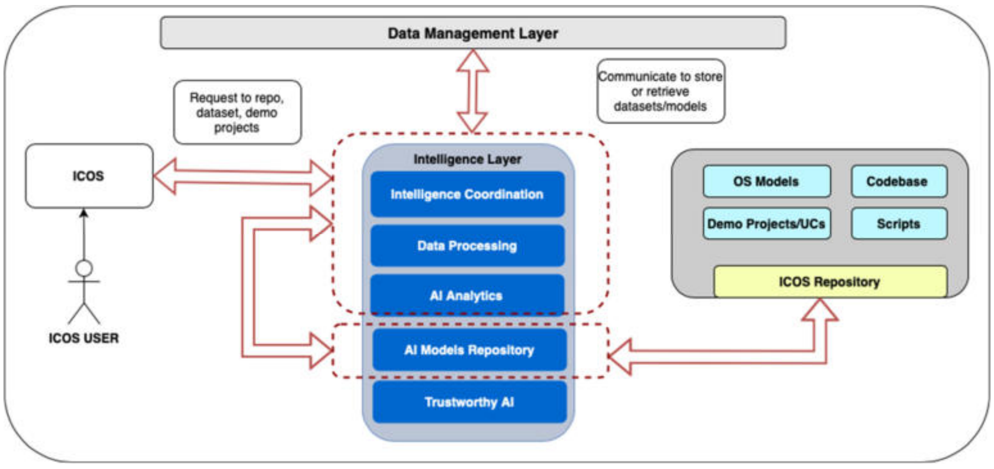


Figure 20: ICOS Repository interactions with other modules.

An ICOS user can request pre-trained OS models from the ICOS model repository; this request can be made from the AI coordination layer, which acts as an interface between the ICOS user and the model repository. This also gives access to the demo projects, use cases and scripts to the user, which it can download using a git pull request and try on their local setup.

Next, an ICOS user can request to improve these models with the help of datasets available in the Data management layer; in this case, as well the request goes through the AI coordination layer, which further collects the dataset from the data management layer and passes it to the corresponding Data processing and AI analytics pipeline, upon completion of the model it gets saved in the ICOS model repository from where the user can access its custom model.

4.4.4 Trustworthy AI

Trustworthy AI is a crucial factor in ensuring the responsible use of AI in the ICOS ecosystem. For this purpose, its goal is to provide specific algorithms to analyse the datasets and develop models while maintaining the utmost respect for privacy and trustworthiness policies. By adhering to these requirements, AI models in ICOS can be developed to provide reliable results while safeguarding individual users' data protection strategies and promoting ethical standards. This can be done by allowing models to be trained in a federated learning fashion to ensure data protection in datasets containing user-specific data and by providing explainable AI algorithms that give meaningful insights into the output of models to the different layers in ICOS.

Therefore, this component can be organized two-fold:

- to ensure trustable, secure & robust model training via federated learning techniques, and
- to provide model explain ability through a series of AI interpretability algorithms, to aid the decision-making process of the models while understanding inputs and outputs.

Both explainable AI methods and the ability for models to be trained via federated learning will be made available to provide data security in datasets, including user-specific data. This will give the various ICOS layers confidence in the models' output.

Trustable, secure & robust model training. This joint of algorithms includes anonymisation/privacy/federated learning to ensure that the data used to train models is protected and that the resulting models conform to policies for privacy and trustworthiness in two different aspects to ensure privacy and model robustness. Models in ICOS can be trained in a federated learning fashion to ensure privacy and trust. Privacy can be guaranteed by training data on the edge. Robustness is assured by assessing the data against anomaly detection.

- Providing trustable models. ICOS trains ethically unbiased models via data stratification techniques to ensure the data is correctly distributed to represent each class realistically. Models may be trained to consider these factors and avoid unfair or discriminatory outcomes by segmenting the data into different and representative groups based on relevant traits, such as race, gender, age, socioeconomic or geography. By using these models, the objective is to preserve trust in the outputs of the system and provide a fair and equitable experience for all.
- Federated Learning models. For the purpose of developing machine learning models using distributed data sources, federated learning is a useful framework to ensure the protection of the data. Compared to conventional centralised training techniques, it has several benefits, including increased privacy protection, scalability, effectiveness, and robustness. For this section, multiple frameworks can be used in the ICOS ecosystem. Figure 21 depicts a visual example of how data can be treated via federated learning models such as FedAvg [15], FedProx [16] and FedPer [17]. The use of Federated Learning in ICOS implies that ICOS Agents will have a local model registry, while controllers will have a global model registry. Inference in the AI Analytics layer would be performed by running distributed workloads of global models in the agents. Simultaneously, model training would be first performed with the local model registries in the agents; finally local model weights would be aggregated in global models to update these.

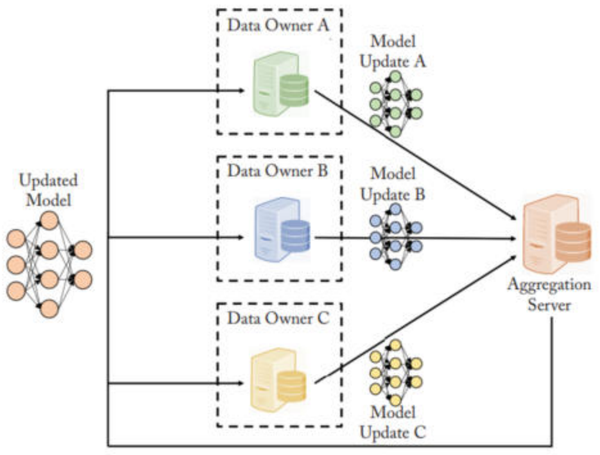


Figure 21: Federated Learning process[8]

[8] Figure source: "A Client-Server federated learning architecture", Federated Learning: https://link.springer.com/book/10.1007/978-3-031-01585-4

- Libraries. Flower[9] is a highly customizable federated learning library that supports multiple machine learning frameworks and allows for customized strategy and model evaluation on the server side. Flower can be used with any machine learning framework, for example, PyTorch[10], TensorFlow[5], Hugging Face Transformers[11], PyTorch Lightning[12], MXNet[13], sci-kit-learn[14], JAX[15], TFLite[16], or even raw NumPy[17] for users who enjoy computing gradients by hand. It allows customizing the strategy, initializing parameters on the server side, choosing a different strategy, and evaluating models on the server side.

Model interpretability. The employment of explainable AI algorithms for model interpretability is essential for guaranteeing that AI systems are transparent and reliable. These algorithms, that will run on the Agents as distributed workloads on demand and triggered by the Controller, can assist in identifying and mitigating biases, enhancing the accuracy and fairness of AI systems by giving insight into how models make judgments. These methods offer insight into the decision-making process of models and may be used to spot biases or flaws in the model architecture or training set. Some possible techniques include:

- Visualizations: Via visualization software of the latent space of the data, it is possible to inspect AI models via AI interactive dashboards that show the inner workings of so-called "blackbox" machine learning models.
- Automated Cause Analysis is another powerful technique for model explain ability that can be used to identify anomalies in the feature space of an AI model. It is possible to increase the model's precision, identify potential biases, and ensure the model produces accurate predictions by comprehending the reasons behind anomalies.

Making sure AI components are trustworthy, dependable, and ethical is a crucial component of trustworthy AI. In this section, the integration of this module with the other layers and components in ICOS is described. The trustworthy AI component will be working collaboratively with the Data Management Layer and the Intelligence Layer to ensure that the data is correctly stratified for tackling biased data, to provide algorithms and finally to analyse data using explainable AI. Figure 22 depicts the action-role of this component.

[9] https://flower.dev/
[10] https://pytorch.org/
[11] https://huggingface.co/docs/transformers/index
[12] https://lightning.ai/docs/pytorch/stable/
[13] https://mxnet.apache.org/versions/1.9.1/
[14] https://scikit-learn.org/stable/index.html
[15] https://jax.readthedocs.io/en/latest/index.html
[16] https://www.tensorflow.org/lite
[17] https://numpy.org/

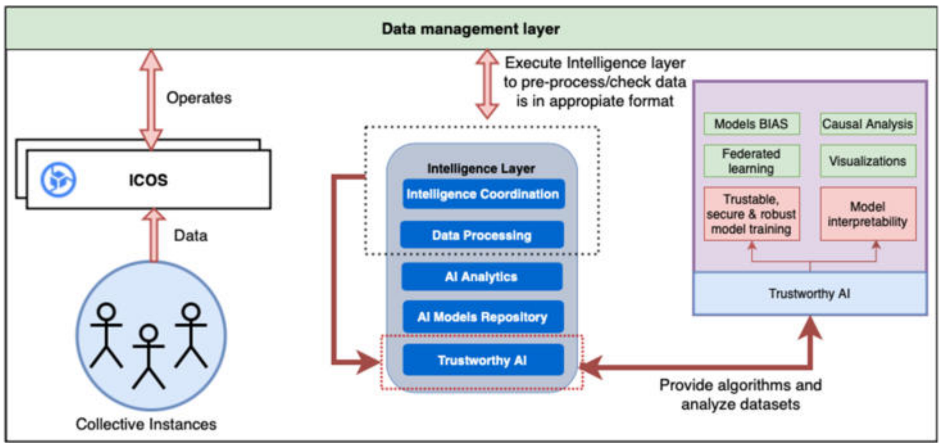


Figure 22: Action roles for trustworthy AI components.

To guarantee that AI components adhere to the standards of trustworthy AI in this situation, action roles are essential in defining their tasks and responsibilities. The suggested method for constructing linked instances Network Provider (NP), Cloud Provider (CP), Edge Computing Infrastructure Provider (ECIP), IoT Provider (IoTP) and other instances that are overseen by a data management layer is used to implement action roles for trustworthy AI components. Before transferring the instances to the intelligence layer (IL), the data management layer must handle them internally. Unit tests are used to confirm that the input data follows the correct formatting once the instances are within the IL. If not, the pre-processing component of the IL does all necessary pre-processing. It will ensure that during the data preparation, validations are performed to ensure that there is no data leakage while model training (through unit tests, for example). The data is sent to the Trustworthy AI component for safe and resilient model training once it has passed the required pre-processing. The security, openness, and interpretability of the AI model are to be ensured by the Trustworthy AI component. To do this, sophisticated methods like explainable AI must be used to reveal the model's decision-making process. The model's resilience to assaults and ability to continue operating amid hostile inputs are guaranteed by the component of trustworthy AI.

4.5 Security Layer

The Security Layer is responsible for guaranteeing the security of ICOS users, resources, and applications at all times. It includes modules for authentication and authorization operations in the system, assess security of resources and applications and suggest mitigation actions, proactive discovery of anomalous behaviours and verification of the compliance of resources and applications. Trust (identity validation) and Privacy (anonymisation and encryption) are architecture-wide functionalities and not specific modules. Figure 23 depicts the internal components of the Security Layer and the main expected integrations with components from the other layers.

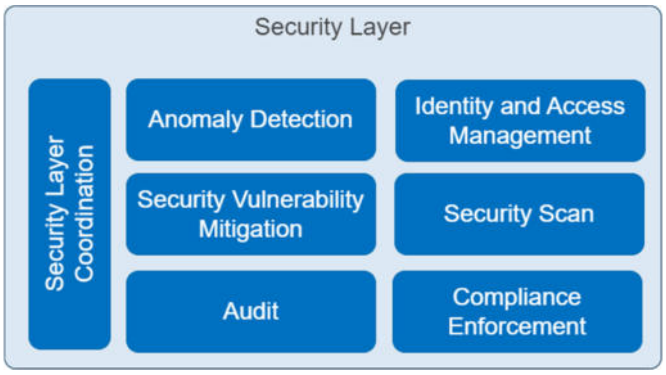

Security Layer
Security Layer Coordination
Anomaly Detection
Identity and Access Management
Security Vulnerability Mitigation
Security Scan
Audit
Compliance Enforcement

Figure 23: Security Layer composition

4.5.1 Security Layer Coordination

The **Security Layer Coordination** module provides a unified interface for interacting with Security Layer Modules - with some exceptions such as Identity and Access Management module's API. The Coordination module also manages data flows inside the layer and provides additional logic needed for seamless interactions between Security Layer's modules. If the latter do not offer functionalities such as task scheduling and periodic tasks, the Coordination module also provides this and offers easy job scheduling.

In general, Security Layer modules do not interact directly with each other but rather interact through the Security Layer Coordination module. One exception that does not follow this concept is Identity and Access Management module as it is used by multiple modules in different layers, so it is directly accessible. Other modules such as Security Scan forward their outputs to the Coordination module which then orchestrates recovery, reporting, etc.

Some Security Layer modules might only be installed on ICOS Controller nodes, while others can be also installed on Agent nodes. It is the Security Layer Coordination module's task to enable seamless interactions between modules installed on both ends and provide a system for distributed processing - similar to the Intelligence Layer Coordination module.

4.5.2 Anomaly Detection

The **Anomaly Detection** module is responsible for detecting anomalies in system and application logs. Log messages are crucial pieces of information generated by software systems to capture and report events that occur during the system's operation. These messages provide a historical record of the system's behaviour and help system administrators diagnose and troubleshoot issues. The sheer volume and complexity of log messages generated by modern software systems make it challenging for system administrators to manually monitor them for anomalous events, highlighting the need for automated tools to assist with log analysis and anomaly detection.

Exploratory Data Analysis (EDA) is thus an important step in anomaly detection as it helps in understanding the data and identifying patterns that may indicate the presence of anomalies. The first step to follow is to visualise using graphs, charts, and histograms. This can help in identifying any outliers or unusual patterns in the data that may be indicative of an anomaly. Next, we can look at the distribution of the data to see if it is normal or skewed. Skewed data may be indicative of an anomaly, as it may be caused by a single large value. Furthermore, checking if there are any missing values in the data is also valuable, as these can cause anomalies in the results. The next step involves identifying any correlations between variables in the data, as this can help us understand how different variables are related and how they might be contributing to any anomalies. The following step could be looking for patterns in the data that might be indicative of an anomaly. This could include spikes or dips in the data, sudden changes in the trend, or unusual patterns that do not fit with the expected behaviour. In addition, comparing the current data with historical data shows if there are any changes that may be indicative of an anomaly. This can be done by plotting the current data against historical data and looking for any differences. The use of statistical methods like mean, standard deviation, and range may help to identify any values that are significantly different from the expected range. By following these steps, we can gain insights into the data and identify any anomalies that may be present. This can help in taking appropriate action to prevent or mitigate any negative impact that the anomalies may have on the system or process.

The Anomaly Detection module will follow two possible workflows. The **first workflow** is a typical log-based anomaly detection workflow, consisting of log template extraction, learning normality, and anomaly detection. First, the initial amount of training data must be collected. The required amount depends on the complexity of patterns present in log sequences. As a rule of thumb, several hundred thousand logs are enough for most use cases. Sometimes even less is sufficient. The important thing is, that the data contains the events and patterns which are expected during the normal operation of the monitored system. The logs are stored in the database (e.g., Elasticsearch[18]).

After the initial amount of training data is collected, the training phase must be initiated. The training phase covers two of the main steps. In the first step, log templates (also known as events) are automatically extracted from the historical data. The selected period of data must be pulled from the database into the Anomaly Detection module where it is processed. Log templates are the static part of a log message, which consists also of the variable part, the parameters. The extracted log templates are then written back to the database and the trained log template extractor is stored in the model registry (e.g., MLflow[19]). Parsing of the log templates usually does not require extensive hyperparameter tuning. However, adding a few simple regular expressions that mask complex parameters can be beneficial for log template extraction quality.

In the second step, the extracted log templates are pulled from the database and the individual log templates get represented in a dense vector form (embedding) and are grouped into sequences. The grouping method can depend on log properties and is roughly divided into two categories: sliding window and session window sequences. The sliding window can be time-based or can aggregate based on the number of consecutive log messages. Session window sequences are generated based on the session id present in logs. The model then learns patterns in sequences present in the training data, which are assumed to be normal. Training the model is compute intensive task and requires GPUs. The trained model is stored in the model registry.

The **second workflow** can use time series models by using numerical features extracted earlier from log-templates. Selecting and tuning the optimal machine learning models is a crucial step in anomaly detection. First, we need to split the data into training, validation, and test sets using the data processing module. The training set will be used to train the model, the validation set will be used to tune the model's hyperparameters, and the test set will be used to evaluate the final performance of the model. The next step includes choosing a set of candidate models to try, for example traditional machine learning models like decision trees or state-of-the-art deep learning-based approaches. The nature of the data also plays an important role in deciding which models to choose. For instance, Long Short Term Memory (LSTM) networks work well with time series data. This can be based also on prior experience and literature review. Both the complexity of the model and the size of the data need to be considered. Next comes the training of each candidate model on the training set and evaluating on the validation set using a suitable performance metric such as accuracy or mean squared error. The best model is selected based on the validation set performance. Moreover, the hyperparameters of the best model can be tuned using techniques such as grid search or random search. This involves searching over a range of hyperparameters to find the combination that gives the best performance on the validation set.

Next, the module enters inference mode. In this mode, a previously trained log template extractor and anomaly model are pulled from the model registry. The period selected for anomaly detection is

[18] https://www.elastic.co/

[19] https://mlflow.org/

provided to the Anomaly Detection module, which pulls the required data from the database. The module uses the trained log template extractor and model to process the data and computes anomaly scores. An anomaly score is a number in the range between zero and one, where zero represents normality and one represents a high anomaly. The scores are written back to the database. The inference can be executed also as a periodical job. In this case, the Anomaly Detection module pulls the new data since the last execution from the database, together with margin logs. Margin logs are needed, because the anomaly score is computed for the last log message in the sequence.

This module will communicate with the API provided by the AI Coordination module in the Intelligence layer to train new models and request inference in already trained models. Such models are in charge of detecting anomalies or issues in the device. It will also interact with the AI Coordination module as this is a middle agent for the data pre-processing, ICOS Model repository and Analytics modules. For this, this module will use Python libraries to make requests to the API.

4.5.3 Security Vulnerability Mitigation

The Security Vulnerability Mitigation module operates based on a Wazuh[20]/LOMOS-like [18] strategy, wherein rule triggering initiates the execution of certain recovery or mitigation processes. The latter can either be fully independent and automated or require the user's input, such as the confirmation of suggested action.

The basic rules and responses are pre-defined in static maps that can be further modified by the end-user. Additionally, the Intelligence layer's AI Analytics component will be used to offer further mitigation strategies and rules. The Security Vulnerability Mitigation module collaborates also with the Security Scan and Compliance Enforcement modules, which provide relevant input data that is evaluated against the Security Vulnerability Mitigation module's rules.

The Security Vulnerability Mitigation module relies on issues found in the Anomaly Detection and Security Scan modules. After the security coordination module receives an alert due to an anomaly or a threat being found, the Security Layer Coordination module will communicate with the Security Vulnerability Mitigation module to find a recovery strategy. As part of these strategies, this module will also communicate with the Intelligence layer to provide smart next steps for self-healing.

Regarding self-healing, depending on the failures or anomalies detected by other modules of the security layer, the Security Vulnerability Mitigation module may be able to use different models to produce recommendations for the next steps to take to solve such issues. Common steps taken in related research to self-healing are the next: i) Identifying the failure or anomaly type, ii) identifying where this issue has occurred, iii) determining the cause of this issue, and iv) recommending strategies to address it. The Security Vulnerability Mitigation module can interact with the Anomaly Detection module and Intelligence layer to produce intelligent recognition and amendments of potential system issues. For this purpose, data collection is based on information from all involved ICOS resources of all ICOS instances, later gathered and pre-processed by both the pre-processing module in the Intelligence layer and data management layer.

Regarding this data-gathering process, self-monitoring solutions might be explored. These can provide useful real-time feedback to keep the runtime manager in the loop. Examples here are alarm systems and concept or data drift monitoring to inform users that only temporal data correlations have changed and reduce false positives in anomalies.

Regarding self-healing, a self-healing framework that trains a neural network-based remediation selection agent to recover from malfunctions and cyberattacks automatically can be explored if enough data can be extracted from systems logs and activity.

[20] https://wazuh.com/

Finally, predictive maintenance using time series approaches could be explored to reduce the need for system recovery. Techniques like LSTMs, other recurrent neural networks and ensembles for regression can be explored. Recommendation systems could be in place to clarify the best next action after a certain type of anomaly or security threat is found. Other than this, the Security Vulnerability Mitigation module will need to leverage static action points developed as part of this security layer. After these action points, a script coded as part of the Security Vulnerability Mitigation module, for an individual action, will be executed for the recovery of the system to be completed.

From the implementation point of view, all the functionalities provided by this module will be final subject to system specifications, and a contingency plan will need to be in place with approaches to cover low-end devices unable to run them. For instance, learning tasks may not occur on the device and data-gathering processes may only be performed only essential information and specific time intervals.

This module will communicate with the API provided by the AI Coordination module in the Intelligence layer in the Controller to request inferencing by models regarding self-healing. Models will be trained in the Agents using the Analytics module through the AI Coordinator API first to classify a system issue with the available information and then to provide recommendations on potential actions depending on the forecasted probability of a given issue. Requests to this API, as in the previous module, will be written in Python.

4.5.4 Audit

The component within Security Layer that is responsible for executing the light-audit checks and providing the resources to the Security Assurance component is the Audit component. The goal of the component is to identify potential vulnerabilities and risks and provide recommendations for improving the security level of the deployed application.

The security audit process typically involves several activities such as risk assessment, vulnerability assessment, penetration testing, security policy review, security awareness training. In ICOS perspective the above processes, as defined by cybersecurity specifications and best practices, are assumed to be conducted across the continuum by the various Infrastructure Providers.

The process of security audit is very meticulous, requires human intervention and depends on the adopted internal policies and procedures of each infrastructure provider. By assuming audit processes already in-place for each Infrastructure comprising the Cloud Continuum, ICOS will implement a lightweight auditing scheme to specify a security audit workflow, defining specific checkpoints that need to be passed to consider the audit successful. Those checkpoints explore different aspects of the Cloud Continuum realisation and take place upon the on-boarding of any infrastructure, ICOS element and resource.

The **Audit** component accepts as inputs static information on security auditing status of each infrastructure and ICOS node, dynamic information via telemetry and anomaly detection results available by the Anomaly Detection component. Finally, the component also considers the ICOS application composition, evaluating the existence of security functions within the application. The outcome is an evaluation of the overall end-to-end security grade within the examined Cloud Continuum instance.

4.5.5 Identity and Access Management

The **Identity and Access Management** (IAM) component has the overall objective of ensuring that the right people will have access to the right resources in the system. The three main responsibilities of these components are the management of the users and their roles/permissions, the authentication of the users and the authorisation of the requests within the system. All ICOS components will rely on the IAM component to make sure that the received requests are properly authenticated (the requesting entity is known, and it is confirmed that it is who it says to be) and authorised (the requesting entity has the requested privileges to do that request). The users managed by the IAM will be the ICOS Users: the Application Integrators that interact with ICOS for managing their application. This does not include the users of the user's applications running in ICOS. At the moment of writing, further analysis is needed to understand if this component, beside the ICOS users, will be also responsible for authenticating and authorising, the devices that will join the Cloud Continuum.

The component will offer a common API to check user's identity and to validate access requests that other ICOS services will call using appropriate client libraries. Authentication and authorisation should be checked at every request (in compliance with a zero-trust principle [14]).

However, scalability and performance aspects will need to be taken into consideration when designing the component to avoid it becoming a bottleneck for the entire system. The IAM component will also implement a Single-Sign-On (SSO) and delegated authorization mechanisms to allow users to keep their identity across interactions with multiple ICOS services with (potentially) service-to-service calls to satisfy a single user request.

To support the distributed and heterogeneous nature of the ICOS system, the IAM component will be based on standard protocols such as OAuth2.0[21] (for authorisation) and OpenID Connect[22] (for authentication) that ensure the maximum level of security and that are very popular and widely supported in several technologies and programming languages. In addition, it will integrate with existing user databases (e.g., LDAP[23]) to allow a smoother adoption of ICOS in already existing environments. Section 5.7 presents the usage of the IAM component in three different scenarios: user, service-to-service and cross-Controller authentication and authorisation flows.

4.5.6 Security Scan

The Security Assessment module provides the following main capabilities: security analytics, intrusion detection, log data analysis, file integrity monitoring, vulnerability detection, and configuration assessment.

In conjunction with the Security Vulnerability Mitigation and Compliance Enforcement modules, regulatory compliance and incident response functionalities are also provided by it. Various security analytics data is provided to the module by multiple security tools with a proven track record, such as Wazuh, Vulnerability Assessment Tool (VAT), and Persistent Memory (PMEM) [19].

Different kinds of data will be analysed, specifically PMEM can be trained to work with network data. Using Machine Learning techniques PMEM detects network anomalies and classifies them into known and unknown attacks. The system is continuously trained to learn new patterns of attacks and then updating the proven track record which feeds the Security Assessment module. The Security Assessment module processes this data further to produce simple and uniform security assessment reports with suggested incident/vulnerability responses for users.

The complexity and extent of monitoring are limited by the processing power and I/O capabilities of the ICOS Node. Still, assessments of critical operations are provided while minimising unnecessary use of resources. This module collaborates with other modules from the security layer through the Security Coordination component. For instance: checking for anomalies, providing recovery and mitigation strategies, and having audit mechanisms, are part of a security assessment.

This module will communicate with the API provided by the AI Coordination module in the Intelligence layer in the Controller to request inferencing by various models, such as the ones in charge of detecting anomalies or issues in the device, detecting anomalies, or providing recommendations for self-healing. Requests will be written in Python.

[21] https://oauth.net/2/
[22] https://openid.net/connect/
[23] https://ldap.com/

4.5.7 Compliance Enforcement

To build a fully trustable system, ICOS needs to verify that all the parts of the system (e.g., resources, applications, networks) complies to certain security policies and rules at any time. In general, there are two types of compliance to security policies: the first one refers to the resources that want to join the cloud continuum. It is necessary for these resources to meet certain compliance criteria, such as minimum computational power for security enforcement, deployment of specific protocols, etc. The second type of enforcement refers to the policies imposed by the applications to the participating ICOS nodes, such as issuing certain rights per node and enforcing privacy regulations.

The goal of the Compliance Enforcement module is to i) define (and, in general, manage) security compliance policies for resources and applications in the ICOS System, to ii) automate the verification of the active policies, iii) to trigger remediation activities for compliance violations.

This module will use data collected within the system mostly from the Logging & Telemetry and the Topology modules to verify compliance to the active policies. Compliance Enforcement will work in conjunction with the Security Vulnerability Mitigation module for the definition of remediation activities. It will interact with the AI Coordination API to request inferences to recommendation systems providing actions to take depending on the probability of an issue. This is based on available data provided by the Meta-kernel, as the self-healing functionality mentioned in the subsection dedicated to Security Vulnerability Mitigation.

Beside policies defined by the ICOS project and by the users, also the support for checking compliance to existing security frameworks/standards will be evaluated during the project. Some examples are framework issued by specialised bodies like:

- The "CIS Benchmarks"[24] issued by the Center for Internet Security (CIS).
- The "Kubernetes Hardening Guide"[25] issued by the United States National Security Agency and Cybersecurity and Infrastructure Security Agency.
- The "Threat Matrix for Kubernetes"[26] defined by MITRE and Microsoft.

ICOS will leverage already existing tools that automate the verification of the compliance for one, or more, of these public frameworks by integrating them into its CI/CD pipelines and running them periodically to ensure continuous compliance monitoring. Another important aspect is the customisation of these tools based on specific requirements and security frameworks that ICOS will target.

4.5.8 Additional functionalities provided by the Security Layer

The Security Layer provides additional functionalities outside of its main modules. Some of the concepts are implemented at the system-wide scale, while the others are part of the modules outside the Security Layer.

Trust. ICOS envisions to support trust and security by implementing both transport protocols, Transport Layer Security (TLS) and mutual TLS (mTLS), and making them configurable in two ways:

- Support for server-side authentication: clients validate the server TLS certificate but not the other way around, that is, the same way of operating as on the web where the web browsers validate the identity of the server via means of the TLS certificate.
- Support for mutual authentication (mTLS): where both server-side and client-side authentication is required.

The configuration of TLS certificates is done via a configuration file.

[24] https://www.cisecurity.org/cis-benchmarks
[25]https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF
[26] https://microsoft.github.io/Threat-Matrix-for-Kubernetes/

ICOS will provide a communication protocol that enables automated and assisted distributed systems to publish secure data which can be subscribed to by the safety monitoring ICOS system. The safety monitoring system could run on the same virtual or physical ICOS controller (i.e., computer/device, a separate computer/device on the IoT network or a remote computer at the edge or in the cloud).

The communication protocol supports server-based authentication over the TLS communication protocol. This is the kind of authentication similar to a web browser using a HTTPS connection. Where the browser verifies if the server it is connecting to, is legit, but the server does not authenticate the browser. That is why a username and password are required if someone wants to authenticate to the service itself. It is similar as depicted in the following figure. As a Client or Peer, one could verify the authenticity of the router by using TLS, but then the router would use the username and password authentication to validate the clients, peers as well as other routers. Moving one step forward, mutual TLS (mTLS) authentication allows the router to authenticate incoming TLS connections from clients, peers, and other routers as shown in Figure 24. In this case, the user will no longer need to use user and password mechanisms for authentication.

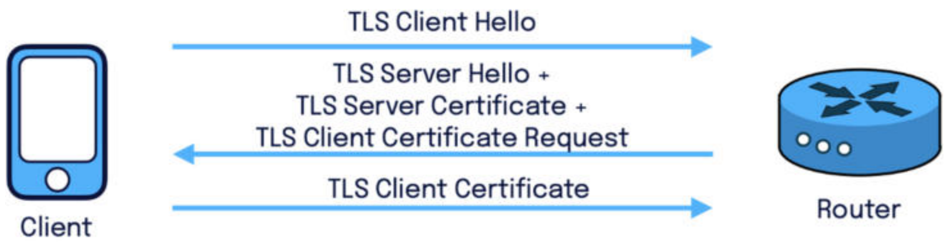


Figure 24: mTLS authentication

Privacy. The privacy functionality comprehends a set of fundamental elements which can be utilised for transforming data, including encryption and anonymisation. It is part of the Data Management and Intelligence layers. In the case of the former, data stored might need to be anonymised to ensure compliance with the GDPR's specific component requirements. The second layer contains the data processing module, which aggregates different functions to anonymise data.

This functionality also ensures compliance with GDPR allowing training algorithms with the Analytics module in a Federated Learning fashion when the raw data cannot be transferred across devices. Access to the two modules of the Intelligence layers is through the AI Coordination API.

Privacy protection is also guaranteed by Federated Learning approaches taken by ICOS, since training is locally performed in various participating nodes without the need to transmit sensitive information to unauthorised third parties. The outcome of each training round is an ML model based solely on the data of the specific node. The model parameters (e.g., weights in hidden layers in case of neural networks) can be then sent to the intelligence layer that performs aggregation and the master model is updated. This updated model is then sent back to the involved nodes that periodically update the employed ML models with more accurate representations. Figure 25 illustrates all the modules involved in privacy related tasks.

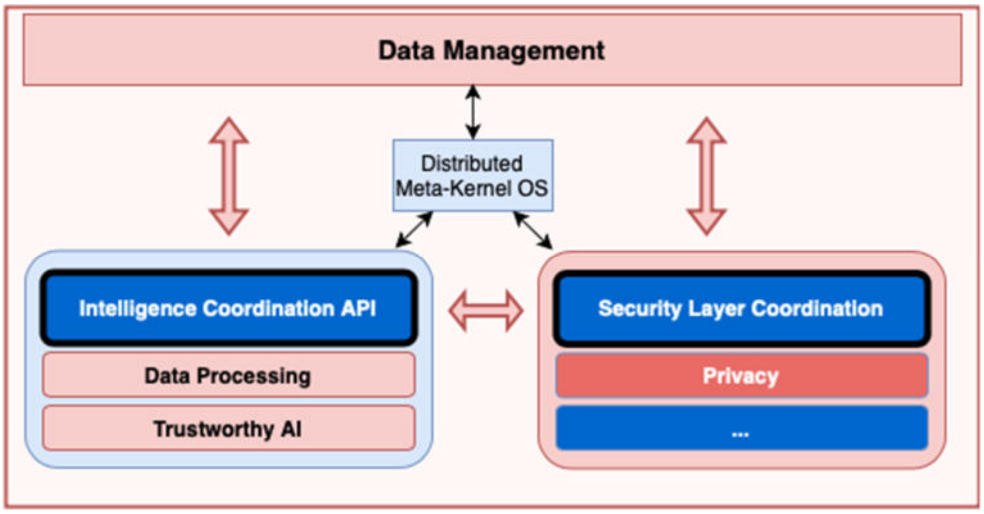

Data Management
Distributed Meta-Kernel OS
Intelligence Coordination API
Data Processing
Trustworthy AI
Security Layer Coordination
Privacy
...

Figure 25: Modules and connections that contribute to privacy functionality are coloured in red

Another part of this functionality is to ensure information encryption in communication and storage. For connectivity, the technology used will be Zenoh. Eclipse Zenoh[27] is compatible with Let's Encrypt[28] which is a free, automated, and open certificate authority (also part of the WebKPI) that provides digital certificates for enabling HTTPS (SSL/TLS) encryption mechanism on websites. Let's Encrypt is unique in that it offers digital certificates that are automatically issued, renewed, and managed, with no cost to the website owner. This removes the cost and technical complexity of obtaining and managing SSL/TLS certificates and makes it easier for website owners to encrypt their traffic and protect their users' privacy. The connectivity component is represented as red arrows in Figure 25.

Certificate authorities (CAs) play two-fold role in the operation of TLS (i) verifying the identity of the sender/receiver of a message; and (ii) providing the means to cypher/decypher messages between sender and receiver (i.e., associate an entity to their public key).

When a client requests a secure connection to a server, the server replies by presenting its digital certificate to the client. This certificate contains information about the server's identity, including its public key, which is used to establish an encrypted connection with the client.

To ensure the authenticity and integrity of the server's digital certificate, it must be signed by a trusted third-party CA (for instance MiniCA[29] if we handle the certificates by our own means). This helps to establish trust and security in the TLS protocol.

4.6 Data Management Layer

The **Data Management Layer** (DML) provides a data management solution distributed across the edge to cloud continuum, supporting the data needs of the three layers in the ICOS architecture.

Its main functionality is to ensure that the required data is available in those devices where it is needed and at the time that it is needed, to efficiently support ICOS operations. This layer will abstract infrastructure and communication details, so that the rest of ICOS components can focus on their specific functionalities and remain agnostic of the dynamicity and heterogeneity of the infrastructure.

[27] https://projects.eclipse.org/projects/iot.zenoh
[28] https://letsencrypt.org/
[29] https://github.com/jsha/minica

In addition, unnecessary data transfers will be avoided to reduce network congestion and increase overall performance of the platform.

In summary, the DML shall be adapted to:

- **Heterogeneity** of the infrastructure: nodes of the DML can have different characteristics, from a small device at the far edge with very limited capacity and computing power, to a cluster consisting of nodes interconnected through a fast and reliable network in a datacentre.
- **Dynamicity** of the infrastructure: the distributed infrastructure is not known a priori, as each node can join or leave at any time during operation and can later re-join.
- **Hyper-distribution** of the nodes: although nodes in ICOS are usually grouped according to locality aspects, nodes that are not necessarily close to each other (e.g., different controllers within an ICOS instance) need to share data to coordinate their operations.
- **Different kinds of data**: the DML must support time series data (e.g., telemetry, etc.), as well as arbitrary data structures to manage the platform (e.g., devices taxonomy, etc.).
- **Performance**: the DML should avoid unnecessary data transfers across devices, e.g., sending only the relevant part of the data, possibly pre-processed and/or aggregated, to its consumers, leveraging the capacity of devices at any level in the continuum.
- **Security**: the DML must ensure that all the data transfers between devices, as well as data at rest, are confidential.

The DML should work across different communication technologies (such as Ethernet, Time-Sensitive-Networking (TSN), Wi-Fi, and 4G/5G), operate at different geographical scales (such as Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN)), and in various topology configurations (such as peer-to-peer, mesh- brokered and routed) to guarantee the coverage of all possible deployment scenarios of ICOS. To minimize the adoption effort, it should also provide a plugin mechanism to integrate with other middleware like Message Queue Telemetry Transport (MQTT), DDS, and HTTP, as well as to integrate with other storage technologies like InfluxDB[30], RocksDB[31], or MariaDB[32].

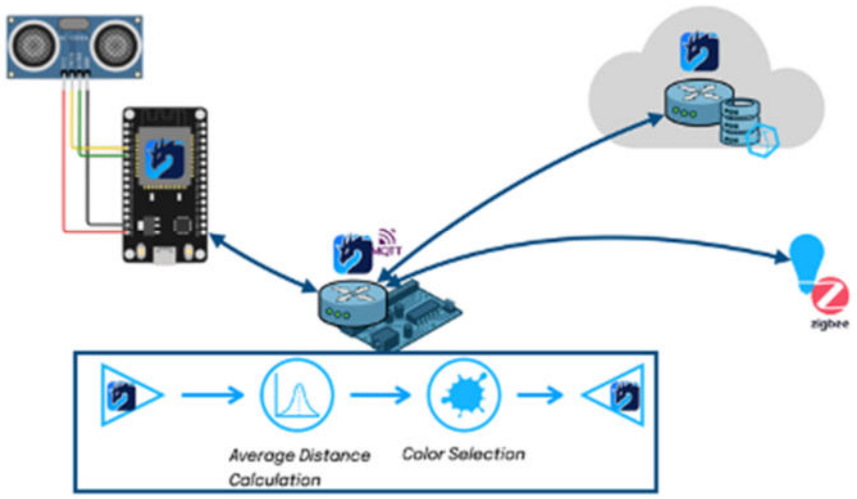


Figure 26:Typical workflow supported by the DML to collect, analyse and store data.

[30] https://www.influxdata.com/products/influxdb-overview/
[31] https://rocksdb.org/
[32] https://mariadb.org/

The DML will also provide an efficient and scalable publish/subscribe mechanism as well as a programming framework to perform computations on the edge. An example of the workflow that will be supported is depicted in Figure 26 where data is collected from a microcontroller-powered sensor, then performs some processing at the edge to perform local actuation (turn on a lightbulb) and stores some historical data in the cloud.

4.7 ICOS Shell

The ICOS Shell is responsible for the communication with the ICOS controller. This includes the managing of authorization tokens, finding the appropriate service endpoints for requests if they have not been configured manually, compilation of queries into a compatible format, and sending them to the appropriate ICOS service endpoints. Furthermore, the Shell layer presents the results of queries to the user, either through the graphical or the command line interface.

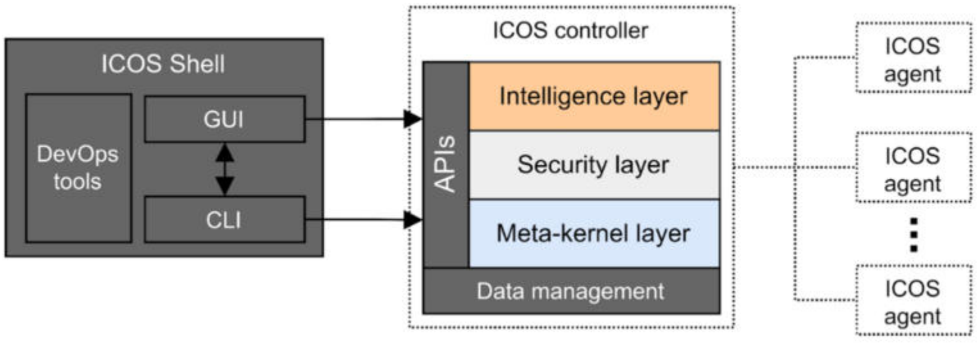


Figure 27: ICOS Shell interfacing the ICOS controller.

The ICOS Shell, shown in Figure 27, consists of three main components: the command line interface (CLI), the graphical user interface (GUI) and the development and operations (DevOps) tools. The Shell will be able to integrate with other tools and pipelines through support for machine-readable input and output.

The Shell layer will interact with:

- the Lighthouse to retrieve available service endpoints,
- the Identity and Access Management to retrieve an authorization token,
- the Continuum Manager to list available resources and
- the Runtime Manager Controller to define and deploy new applications.

This list will be extended in the future as needed whenever new functionalities of the overall system are implemented. In section 5.8, a typical workflow of how the ICOS Shell will interact with the other ICOS architectural components to fulfil the user's requests (a service deployment in this specific example) is provided.

The ICOS Shell layer will be a standalone appliance that can be installed on machines that are not part of an ICOS instance. All features and functionalities will be implemented into the CLI first and then mirrored to the GUI if applicable, so the capabilities of the CLI might exceed the capabilities of the GUI. The component will depend on the security layer; specifically, ICOS Identity Management service, the discovery service, the continuum manager, the runtime manager controller, as well as the controller address. Extended functionality might necessitate communication with other ICOS components and thus will add them to the list of dependencies. The DevOps tools will partially integrate with the CLI and provide tools for application management and to enhance the shell's automation capabilities. While the CLI might integrate libraries that are still to be identified, the GUI will be presented in the form of a locally hosted website.

4.8 ICOS Lighthouse

As previously described, ICOS is envisioned as a dynamic metaOS distributed across the continuum, taking advantage of the capabilities of both the cloud (unlimited resources) and the edge (locality and privacy). One of the most interesting features of ICOS is its elasticity, being able to manage dynamic and mobile nodes efficiently and seamlessly, whether they are Agents or Controllers. Thus, nodes can join or leave the system dynamically, or move through the continuum, establishing new proximity-based relationships between other nodes in different geographic locations.

To manage the system effectively, ICOS has been designed as a distributed multi-controller framework with a flat, unstructured organisation between Controllers. Controllers are distributed among the continuum to provide locality-based management, and Agents are connected to Controllers based on proximity (and eventually also considering the Controller's load capacity or any other relevant characteristics).

Managing such a complex scenario, where nodes dynamically join and leave the system, poses several research challenges. For instance, assume Agent A1 is in ICOS through Controller C1, and that A1 leaves the system. Later, A1 wants to re-join the system. Considering that nodes (Agents and Controllers) may eventually join or leave the system, assume that C1 has left ICOS for any reason. In this scenario, how can A1 re-join ICOS if his main point of contact (Controller C1) is disconnected? To solve this problem, a new actor in ICOS has been proposed: the Lighthouse. The **Lighthouse** is a node with a fixed address, always running, and known by all ICOS members. Whenever a node wants to join ICOS, and for any reason cannot find its corresponding Controller (remember that ICOS is a dynamic system), then the node should always be able to contact the lighthouse to request the most appropriate Controller to join.

For this reason, the main functionalities of the lighthouse are the following:

- Keeps track of all active ICOS Controllers in the system.
- When a Controller joins the system, it will inform the lighthouse.
- When an Agent joins the system for the first time, and it does not know the address of any Controller, it will request a Controller to the lighthouse.
- When an Agent re-joins the system, and it does not find the known Controller, it will request a Controller to the lighthouse.
- When an ICOS Shell user opens a session in ICOS, he or she will request a Controller to the lighthouse.

In summary, the lighthouse is a new actor in the ICOS system whose main role is to be the known and fixed contact point, always available, to be contacted during the node onboarding stage or in case of loss of connectivity with the system.

5 Operational View

This section presents an overview of the behaviour of the ICOS System at runtime. It aims at describing the main tasks to be realized by each system component, as well as identifying and analysing the interactions between the main architectural components when performing a task. From the list of system functionalities identified in section 3, the most representative have been selected and analysed identifying all the interactions between components needed to realize such functionality.

This point of view on the system is useful for system architects and the development team at design time to capture the needed interactions and identify common communication patterns (e.g., publish/subscribe) across the system. It also highlights concurrency and synchronization issues among the different system components that can be discovered and solved early in the design process. This view also supports the integration team as a source of information to select the best integration approach and as a checklist to verify that all interactions expected are realized in the integrated ICOS System. Finally, it is also extremely useful at testing and debugging time to decompose complex tests and workflows into basic interactions to be tested and debugged.

The following subsections analyse some of the ICOS functionalities and propose a description of how these functionalities are implemented by ICOS analysing the interactions of its internal components. UML Sequence Diagrams[33] are used to visualize the described interactions. The functionalities analysed are the ones that have been considered the most critical and risky for the ICOS System and, for this reason, they have been deeply investigated in this first iteration of the architecture definition. The analysis of the other functionalities already started or will start as activity in the context of the technical work packages WP3 and WP4. Therefore, the analysis of the functionalities presented in this section will be refined and extended in the second iteration of the ICOS Architecture.

5.1 Agent On-Boarding

The On-Boarding of new nodes in the Cloud Continuum (SUC_CC_1) is an essential functionality to constitute and maintain the Cloud Continuum. An ICOS Agent node needs to join an active ICOS Controller node before it is exploitable by ICOS and be part of the Cloud Continuum. The same workflow is used when an Agent that was already part of an ICOS System lost the connection with the Controller (or the Agent has been manually put offline).

This section examines the Node Joining step (SUC_CC_4) that is the one, in the on-boarding process, that needs more interactions between the system components. A prerequisite for this step is that the ICOS software is installed and configured in the node (SUC_CC_2 and SUC_CC_3).

To join, the Agent needs to contact a Controller node. However, it is assumed that in some cases the Agent does not know any Controller to contact (e.g., the first time the node is joining ICOS or an already known Controller is not currently running in ICOS). In this case, as shown in Figure 28 the joining workflow starts with the Agent that checks if it has contact with any Controller. If not, then it asks the Lighthouse component for a Controller endpoint, and the Lighthouse returns a Controller endpoint based on multiple criteria.

Once the Agent acquires a Controller endpoint, it first sends a request to join the Continuum also including its authentication credentials. The system verifies the credentials and trust, and secure communications are established (SUC_SC_9 and SUC_SC_10) with further interactions not examined in this section. The Agent then sends its properties and status data. The Controller uses the data sent by the Agent to classify it and update the topology of the Continuum. As shown in the last step in Figure

[33] https://www.uml.org/

28, the Intelligence Layer uses the updated topology data to forecast improvements in the current and new deployments.

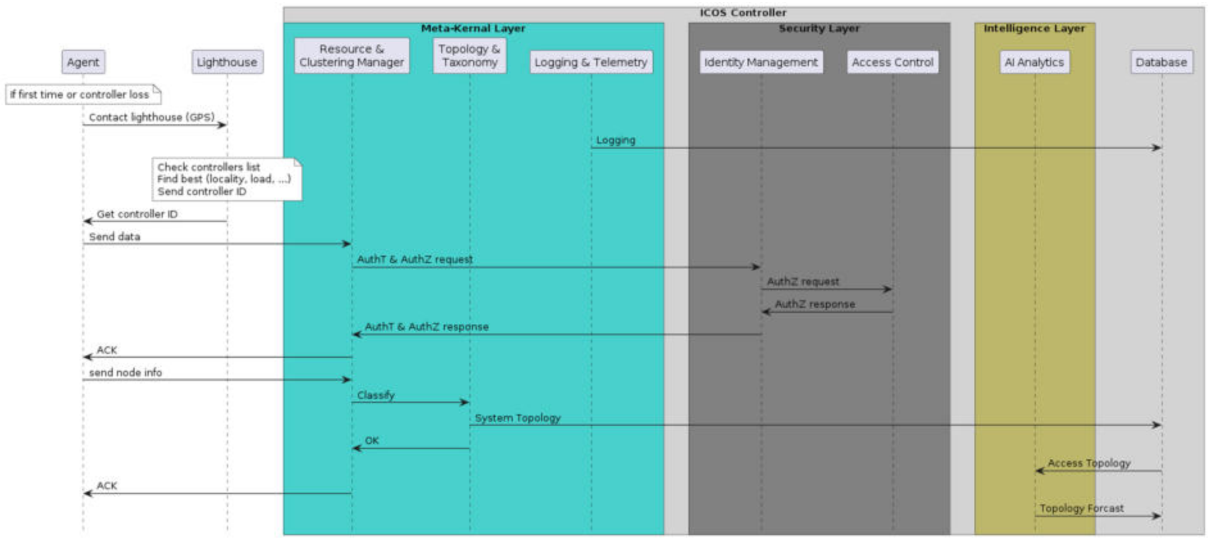


Figure 28: Agent on-boarding process.

Once the Agent has successfully been connected to the ICOS System through a Controller, then it becomes an available resource (according to the properties and status data provided to the assigned Controller) in the continuum.

5.2 Service Deployment

This section examines the interactions between the ICOS components to implement the deployment of a new user's Application or Service (*SUC_RT_4*). A prerequisite, not analysed in the section, is that the application should be already registered in the system. It means that ICOS already knows the application's deployment descriptor and its requirements and policies defined by the user, and a "Service ID" is assigned to the application.

Depending on the structure of the service to deploy, its requirements and the system topology, different cases can occur:

- One service (single task) which can be offloaded to a single Agent, within the Controller scope.
- One service (multiple tasks) which should be offloaded to a set of Agents (becoming an ICOS instance), all of them within the Controller scope.
- One service (multiple tasks, with multiple data source requirements) which cannot be offloaded within the Controller scope, so coordination with other Controllers is required.

In the following subsections the sequence diagram for cases 1 and 2 will be presented. Case 3 will be described in the last subsection.

5.2.1 Single-task service deployment

In the first case (Figure 29), the Controller that receives the request, verifies its validity, and then executes a **matchmaking** process. During this stage, the Controller has a description of the service requirements (header, hints), the system available topology (local database) and the user preferences (compliance policies) and finds the best Agent (in this case, a single Agent) candidate to offload the execution of the service. During this stage, the matchmaking process also generates the **execution strategy** that provides the expected performance. At this point, the Controller offloads the execution to the Agent that will be in charge of the execution.

Once the Agent receives the Service ID of the application to be executed, it will execute the service alone (note this case is a single task to be executed by a single Agent) according to the hints given in the execution strategy information provided by the Controller. Multi-Agent execution is analysed in more detail in section 5.3.

During the service execution, in parallel to the execution by the Agent, the Controller monitors such execution to keep track of the execution health and detect eventual anomalies and/or potential improvements and be able to react accordingly. For instance:

- If the Agent breaks, the Controller will detect the failure and will be able to reassign (or resume) the execution in another Agent.
- If the execution in the Agent does not run according to the expected performance (service compliance policies and/or estimated performance), then the Controller can re-evaluate the matchmaking decisions and decide if the execution should be reallocated.
- If a new Agent joins the Controller scope, and the Controller realizes this Agent can provide much better performance (considering the overhead to stop and resume execution), then the Controller can decide if the execution should be reallocated.

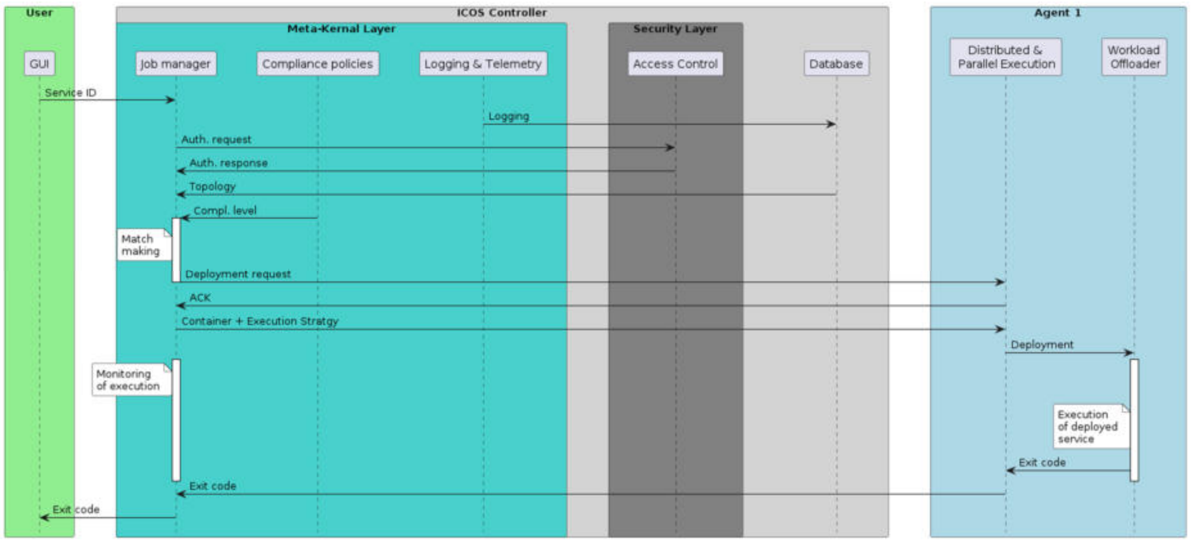


Figure 29: Controller monitor

5.2.2 Multiple-tasks service deployment

In the case that the execution strategy includes the offloading to multiple Agents in the same Controller scope, the workflow is depicted in Figure 30. This case is quite like the previous one, but now the Controller must agree with a set of Agents. Then the actual service execution is initiated by one of the Agents (together with the execution strategy), which in turn will "activate" (task offload) the other Agents (driven by the execution strategy) to create an ICOS Instance.

Now the execution will be performed by a set of Agents, which will create and offload tasks according to the execution strategy (more details on how the service is being executed by a set of Agents is included in the section 5.3). Similarly, meanwhile the Controller will be monitoring the execution to detect and execution failure or opportunity for improvement.

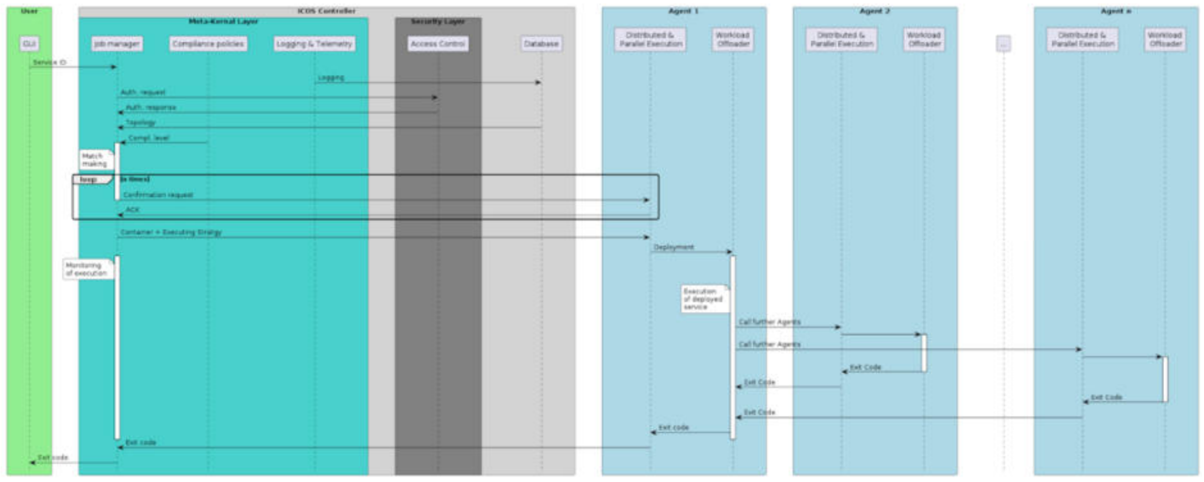


Figure 30: Workflow of multi-task service deployment.

5.2.3 Multiple-tasks, multiple-scopes service deployment

In the third case, the service to be executed consists of several tasks (and/or data sources), and the Controller, during the matchmaking process, realizes that the service cannot be executed successfully with the set of available Agents inside its scope. For this reason, the matchmaking process must be done in coordination with other Controllers. In this case, the Controller contacts with other known Controllers to find the optimal strategy for service execution. Initially neighbour Controllers will be contacted but, if the service request cannot be satisfied yet, then more distant Controllers will be contacted to solve the request.

Once the set of Controllers find the best execution strategy, the process continues as in the previous flow sequences: Agents are contacted to make sure they can participate in the service execution and, once all agree, an ICOS Instance is created to execute the service. Note that although Agents belong to different Controllers scopes, the execution instance will be created exactly in the same way: they are just a set of Agents that collaborate in the execution of an application.

This last scenario is more complex than the previous, and some research challenges have yet to be considered and analysed during the Controllers coordination, and this will be addressed in the second project iteration.

5.3 Service Execution

This section analyses how an ICOS Instance (a set of Agents that execute a given service) executes the service, and how the Agent components participate and coordinate in the execution. It is assumed that the Controller has decided the best strategy for executing the service and has decided the service will be executed by a number of Agents (see section 5.2). In this case, the service consists of a number of tasks, and the Controller has decided, during the matchmaking process, an execution strategy.

The execution of the service starts when the Controller receives the confirmation from all the expected agents of their involvement in the ICOS instance. At this point, the Controller orders the deployment of an application-specific container to every agent passing in also the description of the expected execution strategy. Besides the application software, this container also includes part of the engine of the Distributed and Parallel Execution (DPE) component. During the container start-up process, an instance of the D&P engine parses the execution strategy description and configures itself to consider the offloading of part of its workload onto other remote agents of the same ICOS instance. Once the D&P engine is ready and properly configured, it starts the execution of one of the tasks composing the application on the local computing devices of the ICOS Agent.

In turn, tasks composing an ICOS application can encapsulate the execution of a nested workflow. The D&P Execution engine analyses (Figure 31) the execution of a task looking for the sub-tasks and identifies the potential data dependencies among them to dynamically build these nested workflows and represent them as a directed acyclic graph. With it, the D&P Execution engine is able to discover the implicit parallelism within the task and orchestrates the execution of the sub-tasks to exploit the available resources according to the execution strategy description. When the right time to start the execution of a time has come, if the D&P Execution engine decides that it should run on the computing resources embedded on the local Agent, it starts the execution of the sub-task in parallel to the execution of other tasks in a different thread or process. Otherwise, if the D&P Execution engine detects that it is worth to offload the execution of the subtask onto another Agent rather than executing it on the local devices, it contacts the local instance of Workload Offloader to submit the task onto the desired remote Agent and receive notifications on their progress.

In order to take these subtasks scheduling decisions and allow Application Integrators to oversee the progress of the execution, the D&P Execution engine constantly pushes monitoring information about the progress of the executions running on each Agent, the pending workload detected so far, and the availability of the resources.

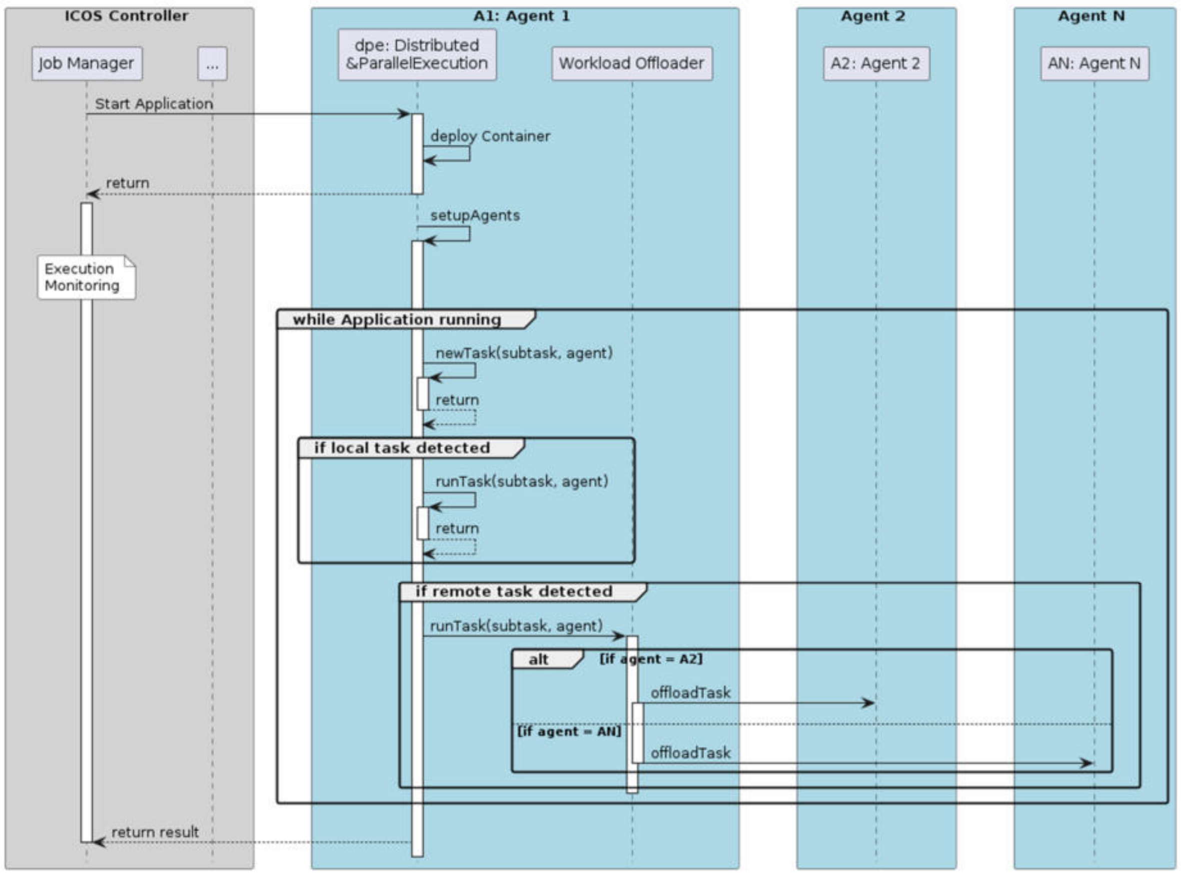


Figure 31: DPE execution

5.4 Optimization of Application Resources

Dynamic reconfiguration of application resources (*SUC_RT_9*) should be based on appropriate AI/ML models that are trained with network telemetry data in one or more ICOS Instances (*SUC_AI_7*, *SUC_AI_9* and *SUC_AI_10*). With respect to the following UML diagram (Figure 32), the individual steps for proper model training are defined as follows:

- An application initiates a resource optimization request for proper execution, or reconfiguration of its resources (it is assumed that such an optimization process has been matched with the appropriate ML model offline).
- The controller associated with this application communicates with the corresponding ICOS Agent to identify if the appropriate ML model has already been deployed in Agent y.
- If this is the case, then the agent requests inference data from the corresponding ICOS Node and a model reconfiguration take place.
- If this is not the case, then data is requested from ICOS Agent y via the telemetry module.
- These data are firstly pre-processed and then passed to the model trainer.
- An interactive procedure is now initiated, where the model is properly trained until a target goal is achieved (model evaluator).
- The final ML model is sent to the ICOS Agent along with an acknowledgement message to the corresponding controller.
- The inference stage along with the reconfiguration stage are now executed, as described in the previous steps.

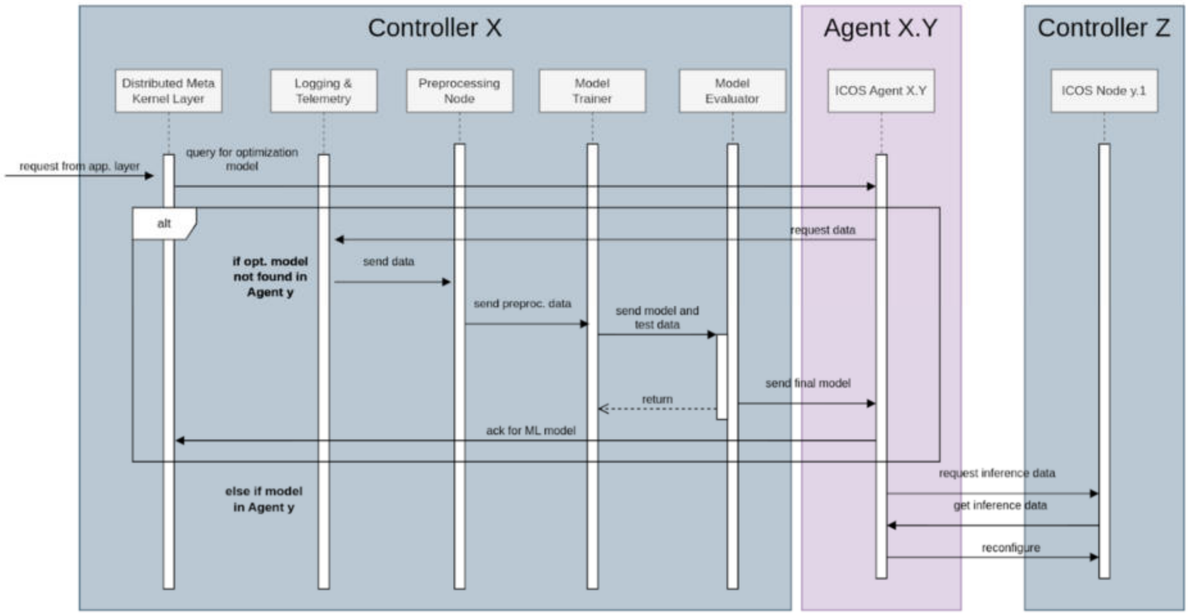


Figure 32: UML diagram

5.5 Policies Management and Evaluation

Users can express policies to control the execution of their applications. ICOS will evaluate these policies before deploying the application as well as during its execution and will generate alerts and optimization suggestions for the user (*SUC_RT_8*). When a new application is created, policies can be defined as well (*SUC_RT_3*) as shown in the box (1) in Figure 33. Policies can be added, removed, and modified also after the application has been created and deployed. There is also the possibility, to be further investigated, that some policies could be defined by other ICOS components (e.g., the Runtime Manager) to enforce specific behaviours.

Policies evaluation (boxes 2 and 3 in Figure 33) starts when the application is deployed and works by retrieving the data from the Monitor and Telemetry component (and potentially from other components) that is needed for the evaluation of the specific policies. In case that violations are detected, alerts, suggestions and/or remediation actions (predefined by the policy definition) are sent to the user or other components of the system that can execute actions to resolve the violations.

Finally, ICOS will also provide a violation prediction feature: looking at the status of the system, it will predict potential future violations of the policies and notify the user and/or apply countermeasures before the violations occur. In this case (box 4 in Figure 33) the prediction part will be offloaded to the intelligence layer through the integration with the Intelligence Layer Coordination component.

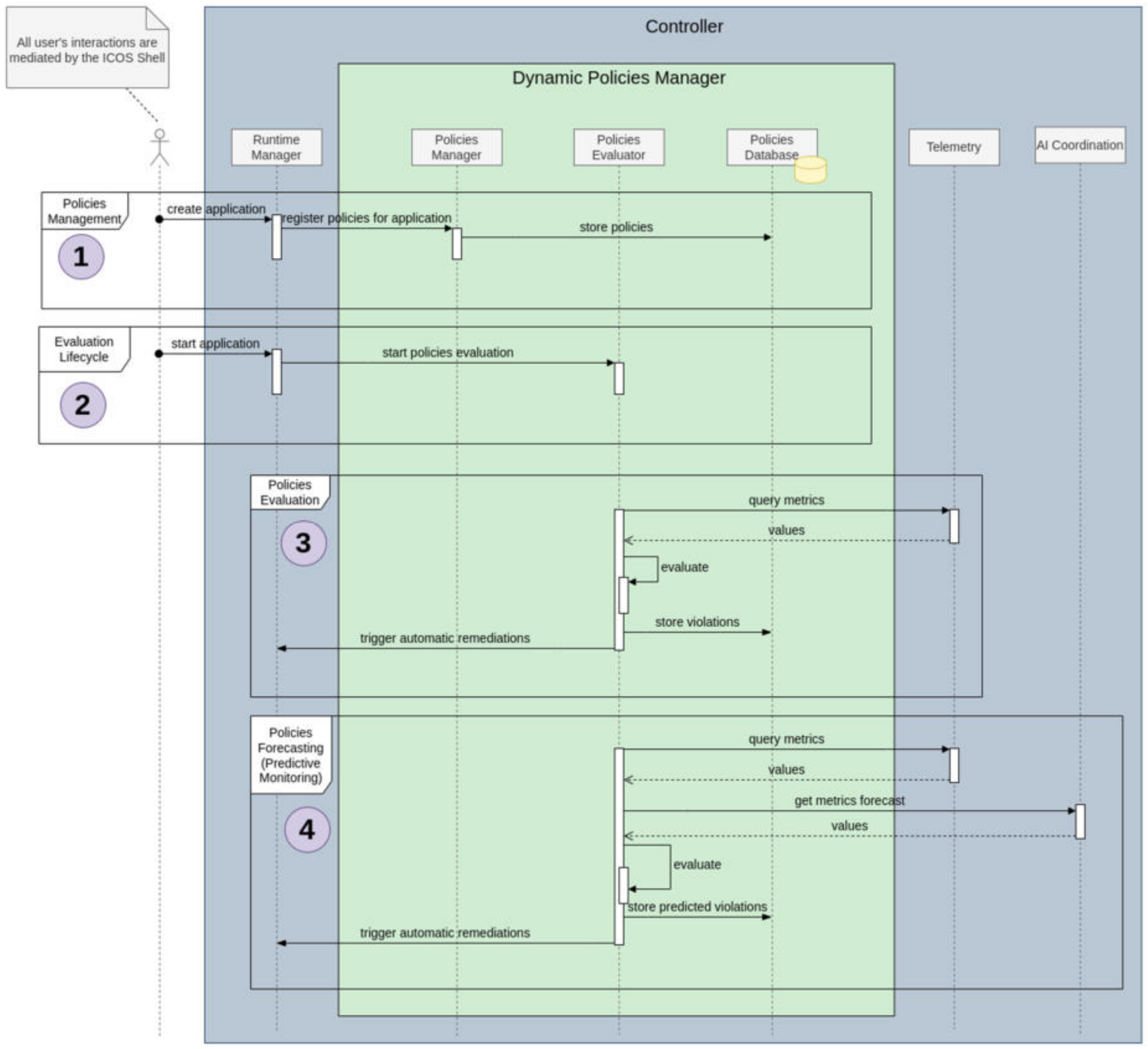


Figure 33: Policies management and evaluation UML diagram.

5.6 Anomaly Detection based on Log analysis

This section examines how anomalies are detected in ICOS (*SUC_AI_8* and SUC_SC_6). The workflow is presented in Figure 34. Logs (e.g., from an ICOS application) are collected in long-term storage (database). The Anomaly Detection module can be started after the initial amount of logs is collected. The required amount of data depends on the nature of the monitoring asset.

The Anomaly Detection module works in three main phases:

1. Training log template extractor
2. Training anomaly detection model
3. Log anomaly detection (inference)

The log template extractor is trained in the first phase. It requires historical logs to detect variable and static parts of the log messages. The extracted log templates are pushed to the database and the trained log template extractor is preserved in the internal model registry. Extracted log templates can be checked by the system administrator. Training log template extractor usually does not require extensive hyperparameter tuning. However, slight modifications can greatly increase the quality of extracted log templates.

In the second phase, the anomaly detection model is trained on the extracted log templates from the previous step. Log templates are pulled from the database. Training of the model requires a GPU. The trained model is stored in the internal model registry.

In the final phase, trained models are used for anomaly detection (inference mode). First, the trained log template extractor and anomaly detection model are loaded from the internal model registry. Next, they are used to process new data in the database, as a one-time or a periodical job. New logs are assigned with anomaly scores and stored in the database.

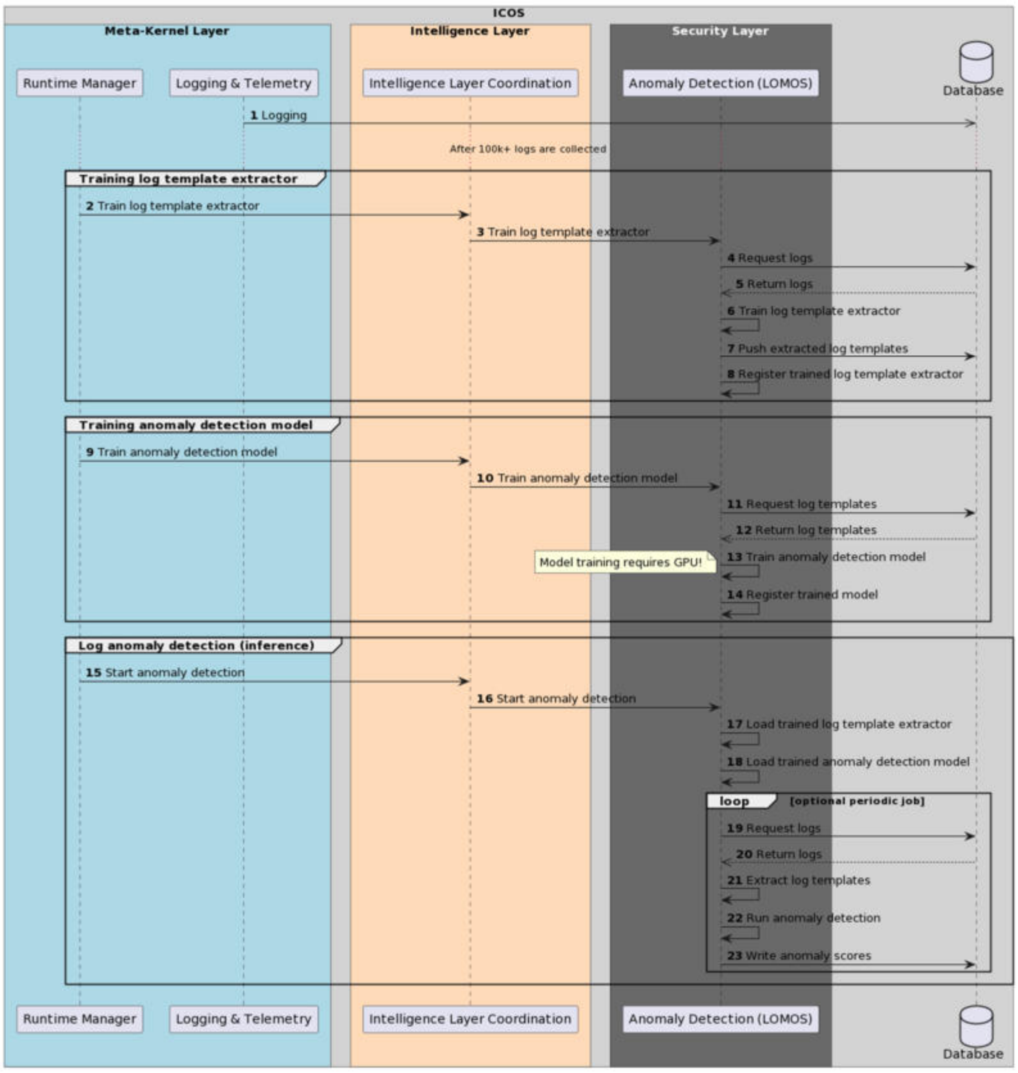

ICOS
Meta-Kernel Layer
Intelligence Layer
Security Layer
Runtime Manager
Logging & Telemetry
Intelligence Layer Coordination
Anomaly Detection (LOMOS)
Database
1 Logging
After 100k+ logs are collected
Training log template extractor
2 Train log template extractor
3 Train log template extractor
4 Request logs
5 Return logs
6 Train log template extractor
7 Push extracted log templates
8 Register trained log template extractor
Training anomaly detection model
9 Train anomaly detection model
10 Train anomaly detection model
11 Request log templates
12 Return log templates
Model training requires GPU!
13 Train anomaly detection model
14 Register trained model
Log anomaly detection (inference)
15 Start anomaly detection
16 Start anomaly detection
17 Load trained log template extractor
18 Load trained anomaly detection model
loop [optional periodic job]
19 Request logs
20 Return logs
21 Extract log templates
22 Run anomaly detection
23 Write anomaly scores
Runtime Manager
Logging & Telemetry
Intelligence Layer Coordination
Anomaly Detection (LOMOS)
Database

Figure 34: Anomaly detection

5.7 Identity Management and Access Control

All operations in the ICOS System are expected to be authenticated and properly authorized before being executed. The authentication (*SUC_SC_1*) and the authorization (*SUC_SC_2*) functionalities are provided by the Identity and Access Management component. For this reason, this component is invoked in almost all interactions between components and involved in the realization of all the functionalities. In this section, three cases are analysed to describe how:

1. Users' authentication and authorization are achieved (Section 5.7.1).
2. Service-to-Service authentication and authorization is achieved (Section 5.7.2).
3. Cross-Controller identities and authorization are achieved (Section 5.7.3).

As specified in section 4.5.5, at the current status of analysis, users that will be managed by the IAM component are the Application Integrators that interact with ICOS for deploying and managing their applications. This does not include neither the final users of applications nor the devices that are part of the Cloud Continuum.

All these workflows rely on the OAuth2.0 protocol for the communication between components. This ensures the maximum security and trust levels at the state of the art and an easy implementation and integration in existing or new components.

5.7.1 User Authentication and Authorisation

In the user's authentication and authorization workflow (Figure 35), the first step is the authentication of the user. In particular, the OAuth 2.0 "Authorization code flow" can be adopted to require an access token that will be stored locally in the ICOS Shell. The ICOS Shell will then send the token in the calls it executes to the other ICOS services on behalf of the user. When the service receives the request, it first checks the presence of the token and asks the IAM component to validate it. If the token is valid, the service asks the IAM component to check the permissions. This type of interaction generates high traffic to the IAM because every service call generates two calls to the IAM. If this will cause a bottleneck in the architecture, both the token validation and the permissions check can be done offline using self-contained tokens (e.g., JSON Web Token (JWT)) without any loss in the security of the system.

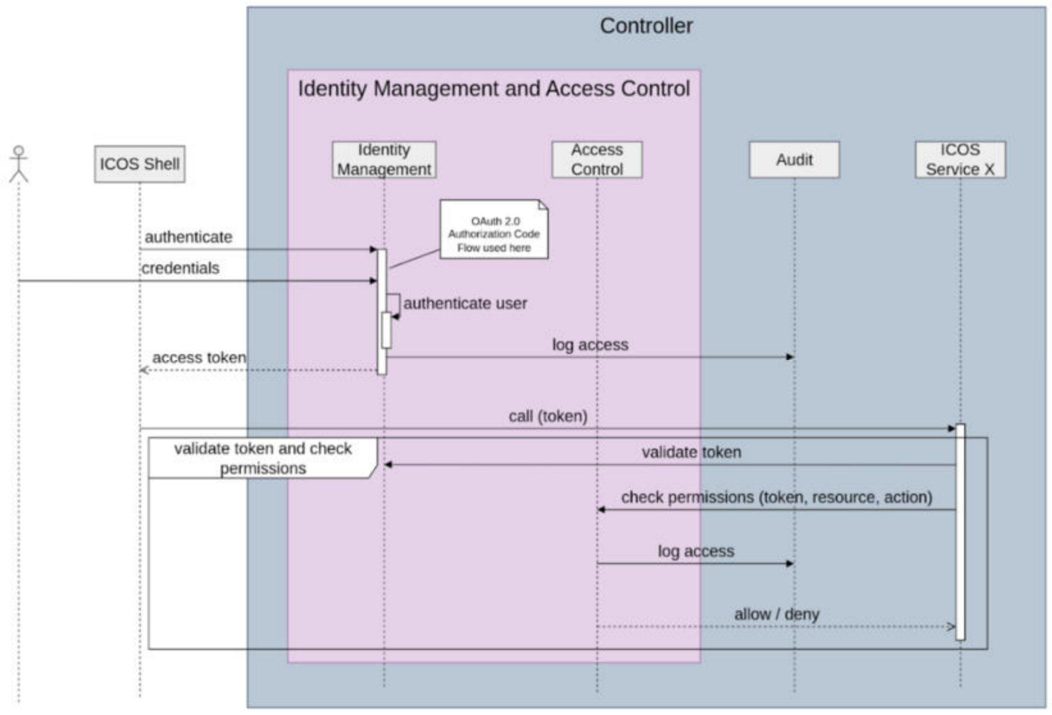


Figure 35:user's authentication and authorization workflow.

5.7.2 Service-to-Service Authentication and Authorisation

Like the user's requests, also the service-to-service requests (Figure 36) should be authenticated and authorized, so they need to contain a token that the receiving service can validate. Service X, before calling Service Y, needs to acquire a token from the IAM component (in case it does not already have a valid token cached). In this case, the Oauth2.0 "Client Credentials flow" will be used to obtain a valid access token. However, if Service X is calling Service Y in the context of a user request, this flow would cause the original user identity to be lost. To avoid this situation, the Oauth2.0 "Token Exchange flow" can be used. In this flow, Service X contacts the IAM component and exchanges a token (the user token in the request) with another token that will contain both the identity of Service X and of the user. In this way, Service Y will know who did the request (Service X) and on behalf of who (the user identity).

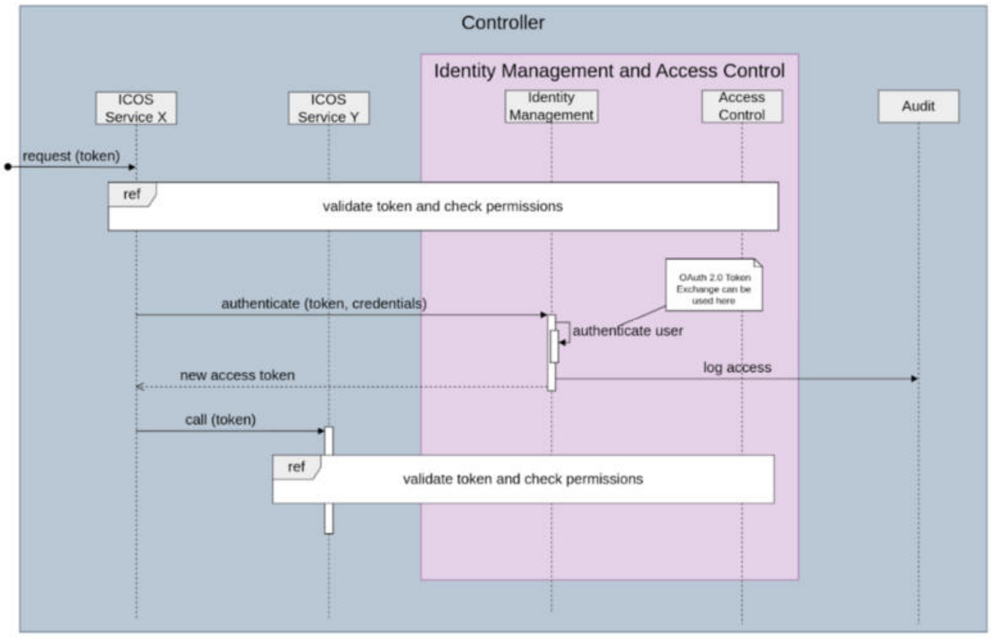

Controller
Identity Management and Access Control
ICOS Service X
ICOS Service Y
Identity Management
Access Control
Audit
request (token)
ref
validate token and check permissions
OAuth 2.0 Token Exchange can be used here
authenticate (token, credentials)
authenticate user
new access token
log access
call (token)
ref
validate token and check permissions

Figure 36: service-to-service requests.

5.7.3 Cross-Controller Authentication and Authorisation

The last diagram (Figure 37) analyses the case where two Controllers have to collaborate to fulfil a user request. This interaction enables one ICOS Controller to make requests to another ICOS Controller on behalf of an ICOS user (e.g., schedule the execution of a job, get monitoring data) in an authenticated and secure way.

The main assumption is that each ICOS Controller will run its own IAM instance with its own user database that will serve all the services in the Controller. There will be no coordination/synchronisation of the user database across ICOS Controllers. This makes it possible for each Controller to apply its own policies for user registration and management. However, to make Controller-to-Controller communication possible, the IAM will be responsible for establishing trust between different Controllers and for mapping identities across Controllers.

Before a user will be able to add Controller B to a continuum, a trust relationship must be established between the two controllers. From the OAuth 2.0 perspective, this means registering an Oauth2.0 client in Controller B that represents Controller A. This action can be partially automated using the OAuth2.0 “Dynamic Client Registration protocol”, however it might involve manual actions by the Infrastructure Providers of the two controllers. This action should be done only one time for each pair of controllers that needs to have a communication (not for every user). As result of this action, Controller A will receive a set of credentials that will authenticate itself to Controller B. These credentials will be stored in the Secrets Storage component. The Secrets Storage components could be a subcomponent of the IAM or an independent component that could be added to the ICOS Architecture. This aspect needs further investigation, and a final decision will be taken in the context of WP4 activities.

After the two Controllers trust one another, when a user wants to add Controller B to the continuum, it will send a request to the Continuum Manager. The Continuum Manager will send the request to the IAM instance on Controller A that will ask (authenticating itself with the stored credentials) the IAM instance on Controller B to authenticate the user. At this point, the OAuth2.0 “Authorization Code flow” will be triggered that will have as result an access token for the user to be issued by Controller B’s IAM. The token is stored on the Controller A’ Secrets Storage and will be retrieved any time Controller A needs to contact Controller B on behalf of the user.

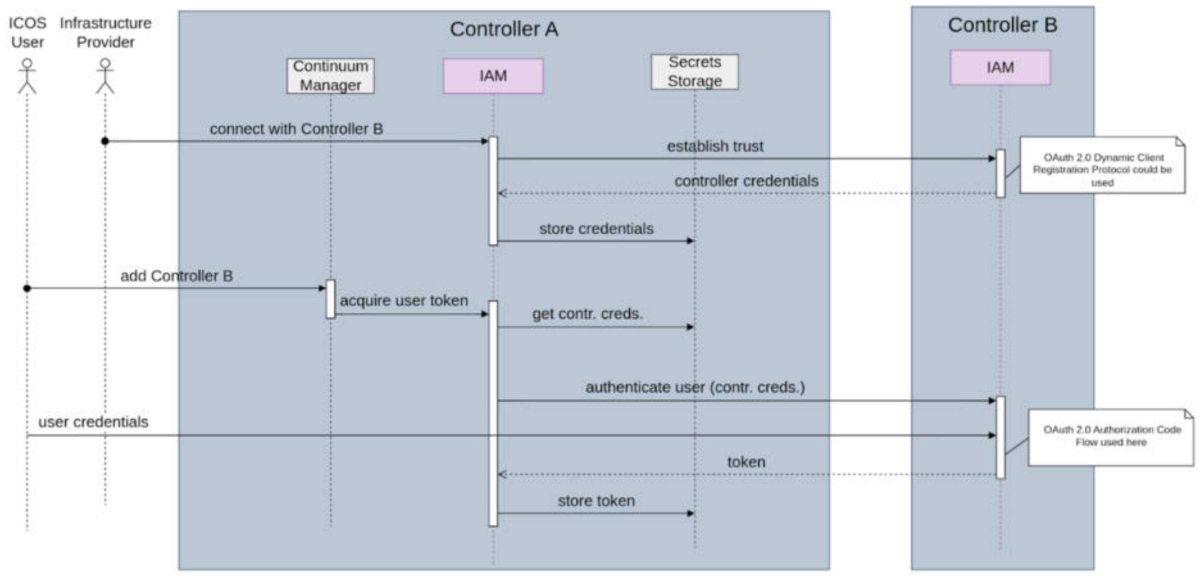


Figure 37: interaction from two Controller to fulfil a user request.

All requests that are managed by the IAM component, even when not highlighted by these diagrams, will also generate audit logs that will be transmitted to the *Audit* component (SUC_SC_3). Similarly, the diagrams presented do not cover the management (creation, removal, modification) of users and permissions (SUC_SC_7) registered in the Identity and Access Management component, which are assumed to be already configured.

5.8 Deployment through the ICOS Shell

The ICOS Shell will provide a user interface to access all the ICOS functionalities. The ICOS Shell will receive the requests from the user and will translate them into requests for the backend services (Figure 38). Responses will be received by the ICOS Shell and presented to the user.

Two pre-conditions for the ICOS Shell to be able to do any request are: i) to know the endpoints of ICOS services to contact (presumably all in the same ICOS Controller), ii) to have an authorization token to contact the ICOS services.

To know the endpoints of the services, the ICOS Shell can be manually configured by the user, or it can make a request to the discovery service (that must be pre-configured manually) that will return the endpoints.

All requests to the ICOS services should be authenticated and authorized. For this reason, the ICOS Shell should request an authorization token to the ICOS Identity Management service before initiating the communication with any ICOS service. Tokens can be reused in multiple calls if not expired and with a compatible scope.

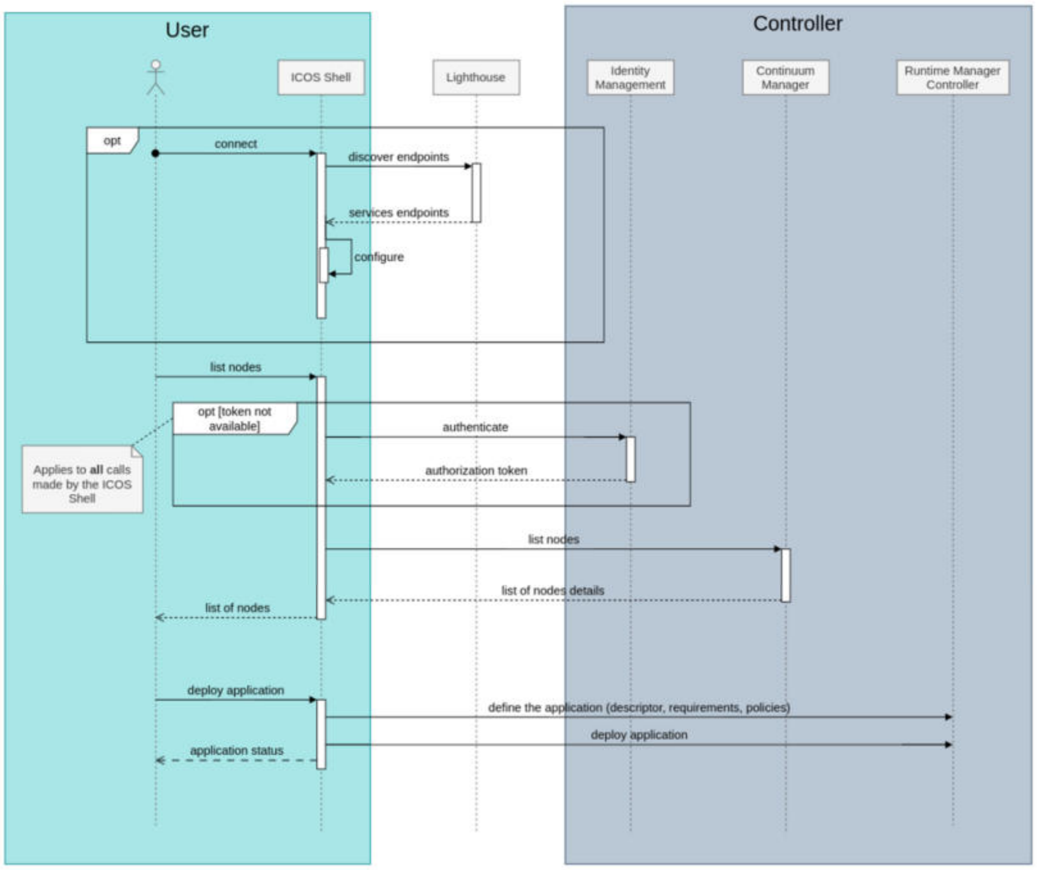


Figure 38: Deployment from ICOS Shell.

6 Development process

The architecture for the ICOS System, defined in this deliverable, represents the first and essential step of the analysis and the design of the system, towards the implementation of the system. Starting from this architecture, the work will continue within the technical Work Packages WP3, WP4 and WP5 with the detailed design of each architectural component, its development, integration, validation, and release.

This section has the objective of preparing the work for the technical Work Packages providing initial considerations on some required aspects to kick-off the technical work. In particular:

- which technological stack will be used to implement the different architectural components,
- which partners and in which project's task each architectural component will be developed,
- a plan for the release of the different features of the system,
- the integration process to use to test and release the components.

6.1 Integration Technologies

The integration of all the components developed in the project into a unique and uniform software system that realizes the architecture defined in this document is a complex task due to the size and the nature of the system. It will be managed on two different planes: the **technological** plane and the **process** plane.

From a **technological point of view**, it is essential that all technologies used for the implementation and deployment of the components are easily and efficiently integrable to reduce the effort for the integration at its minimum and the risks of incompatibilities later in the development phases.

To this end, a set of collaborative decisions and checkpoints have been identified during the development process as depicted in Figure 39. The expected steps are described in the remaining of this section.

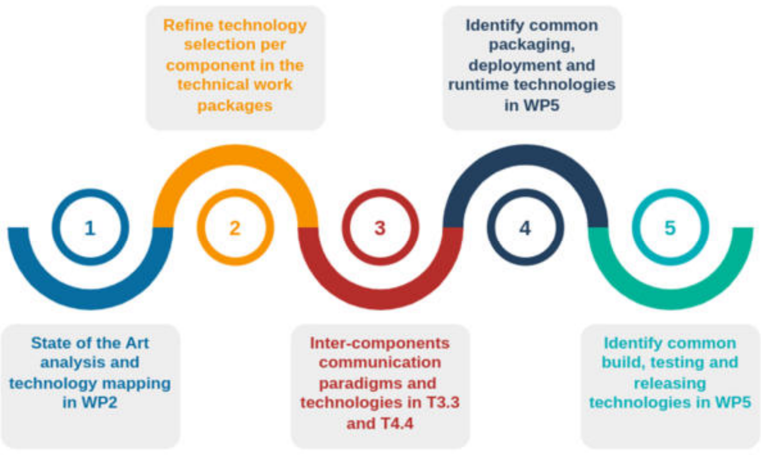

Refine technology selection per component in the technical work packages
Identify common packaging, deployment and runtime technologies in WP5
1
2
3
4
5
State of the Art analysis and technology mapping in WP2
Inter-components communication paradigms and technologies in T3.3 and T4.4
Identify common build, testing and releasing technologies in WP5

Figure 39: Development strategy.

As **first step,** a State-of-the-Art analysis of the baseline technologies that will be used to implement the different ICOS functionalities has been conducted in deliverable D2.1 [1]. These technologies have been filtered and mapped to the ICOS functionalities and the architectural components identified in this deliverable **(step 2)** to produce a first set of candidate baseline technologies to be used in ICOS.

Table 8 shows an initial mapping of all the architectural components to:

- **3rd party technologies** and/or partner's assets that will be used to realize the component. For a single component, multiple technologies can be used each one covering a functionality or as alternatives,
- **the project task** in the technical Work Packages (WP3, WP4 and WP5) where the component will be designed and developed,
- **the partners responsible for the component**. Several partners could contribute to the implementation, but one (or few) responsible should be appointed based on the ownership of the project task where the component is developed or the amount of effort in the implementation.

Technologies and responsibilities not yet identified will be assigned during the activities of the technical Work Packages or in the second iteration for the definition of the final ICOS Architecture.

Table 8: Initial mapping of all architectural components.

Architectural Component	Technology / Assets mapping	Task	Responsible Partner
Distributed Meta-Kernel			
Resource & Clustering manager	Nuvla.io[34], NuvlaEdge, Kubernetes/OKD[35], Open Cluster Management[36] (OCM), InterPlanetary File System[37] (IPFS)	T3.2	SIXSQ, RHT
Topology	n/d	T3.2	SIXSQ*
Logging and Telemetry	Prometheus[38], Loki[39], Observatorium[40]	T3.4	RHT
Network Management	Submariner[41], Eclipse Zenoh[27]	T3.4	RHT, ZSCALE
Job Manager	n/d	T3.1	RHT*
Workload Offloader	COMPSs [20]	T3.1	BSC
Distributed and Parallel Execution	COMPSs	T3.1	BSC
Dynamic Policies Manager	OCM Policy Controller[36]	T3.5	ATOS, ENG

[34] https://sixsq.com/platform
[35] https://www.okd.io/
[36] https://open-cluster-management.io/
[37] https://ipfs.tech/
[38] https://prometheus.io/
[39] https://grafana.com/oss/loki/
[40] https://observatorium.io/
[41] https://submariner.io/

Architectural Component	Technology / Assets mapping	Task	Responsible Partner
Intelligence Layer			
Intelligence Layer Coordination	FastAPI[42], Flask[43]	T4.2	CeADAR
AI Analytics	Tensorflow[5], Pytorch[10], Scikit-learn[14], MXNet[13]	T4.2	NKUA
AI Models Repository	MLFlow[19], HuggingFace Hub[11]	T4.2	CeADAR*
Thrustworthy AI	Flower[9], ExplainerDashboard[44], SHAP[45]	T4.2	CeADAR*
Security Layer			
Identity and Access Management	Keycloak[46]	T4.3	ENG
Anomaly detection	PMEM [19], Wazuh[20]	T4.3	XLAB
Security Scan	PMEM, Wazuh, Vulnerability Assessment Tool (VAT)	T4.3	XLAB, UPC
Compliant Enforcement	Kube Bench[47], OSCAL[48], CIS-CAT[49], kube-hunter[50]	T4.3	NKUA, ENG
Security Layer Coordination	n/d	T4.3	XLAB
Audit	n/d	T4.3	XLAB*
Security Vulnerability Mitigation	n/d	T4.3	XLAB*
Data Management Layer			
Data Management	dataClay[51], Eclipse Zenoh[2727]	T4.1	BSC

[42] https://fastapi.tiangolo.com/lo/
[43] https://flask.palletsprojects.com/en/2.3.x/
[44] https://explainerdashboard.readthedocs.io/en/latest/
[45] https://shap.readthedocs.io/en/latest/index.html
[46] https://www.keycloak.org/
[47] https://github.com/aquasecurity/kube-bench
[48] https://pages.nist.gov/OSCAL/
[49] https://www.cisecurity.org/cybersecurity-tools/cis-cat-pro
[50] https://github.com/aquasecurity/kube-hunter
[51] https://dataclay.bsc.es/

Architectural Component	Technology / Assets mapping	Task	Responsible Partner
ICOS Shell			
ICOS Shell	n/d	T5.1	TUBS
DevOps tool	GitLab[52], SonarQube[53], GOHarbor[54]	T5.3	ENG

(*): responsible of the task where the functionality will be implemented, not the real responsible for the implementation. Responsibilities will be managed within the task.

During the design and development phases (**step 3**) the list of baseline technologies will be further analysed, if necessary, with discussions and decisions taken at work package level (in the technical work packages WP3, WP4 and WP5). In this phase, more detailed aspects can be analysed for each technology, like the requirements for deploying and running it (to see if they are compatible with the deployment scenarios of ICOS) and the security levels required/offered. Analysis of inter-components interactions and communications needs have been started in this deliverable (in Sections 4 and 5). It will be continued in tasks T3.3 and T4.4 that will be responsible for identifying common approaches and technologies for the communication between ICOS components like APIs, software buses, publish/subscribe mechanisms.

In **step 5**, to further support the integration of the different components into a unique and homogeneous system a set of common packaging and runtime technologies will be discussed and selected in the context of Work Package 5. Of particular interest would be the adoption of containerizations (e.g., **Docker Images**) and clustering (e.g., **Kubernetes**) technologies as runtime platforms to run the ICOS system. Their usage can support the integration and the realization of the ICOS System in multiple ways:

- support the achievement of some of the driving principles of the architecture like modularity, robustness, 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 (given the wide adoption of Docker and Kubernetes worldwide).

At the current stage of analysis and definition of components the adoption of these technologies seems possible for large portions of the system already. In addition, from the analysis of the State of the Art, the technology mapping (Table 8) and the analysis of the project's use cases, Docker and Kubernetes are also baseline and/or supported technologies of multiple components in the ICOS System (e.g., the Continuum Manager and the Runtime Manager). However, their adoption for all components will be further discussed also considering aspects like resource requirements (e.g., in very constrained devices).

Finally, in **step 5**, a common technological stack for the integration, testing and release of the ICOS System will be identified in Work Package 5. Section 6.2 proposes a preliminary list of some of these technologies and tools.

[52] https://about.gitlab.com/
[53] https://www.sonarsource.com/products/sonarqube/
[54] https://goharbor.io/

6.2 Integration Process

The integration and testing of the components developed in the project will be coordinated, monitored, and validated in project's tasks T5.3 and T5.4. A specific process will be defined that all the components will have to follow to ensure a high level of quality in the software produced and released in the project.

The overall project's development strategy (Figure 39) is based on a continuous development of the software during the overall project lifetime with fixed checkpoints and milestones (software releases). To support this strategy, the integration process will be based on the **Continuous Integration and Continuous Delivery (CI/CD) principles**.

As depicted in Figure 40, as soon as new code is produced in the technical work packages, it will go through a pipeline of activities aimed at verifying its technical integration in the system and validate its functionality accordingly with the system specifications. These activities will include building, testing, validation, packaging, releasing in an artefact repository, deployment in a staging environment and assessment of the runtime integration and functionality. The fact that this pipeline will run not only when the system releases are expected, but every time new code is available allows to have more frequent feedback on the new code quality, more time for developers to remediate to problems and a reduced risk of releasing software with defects.

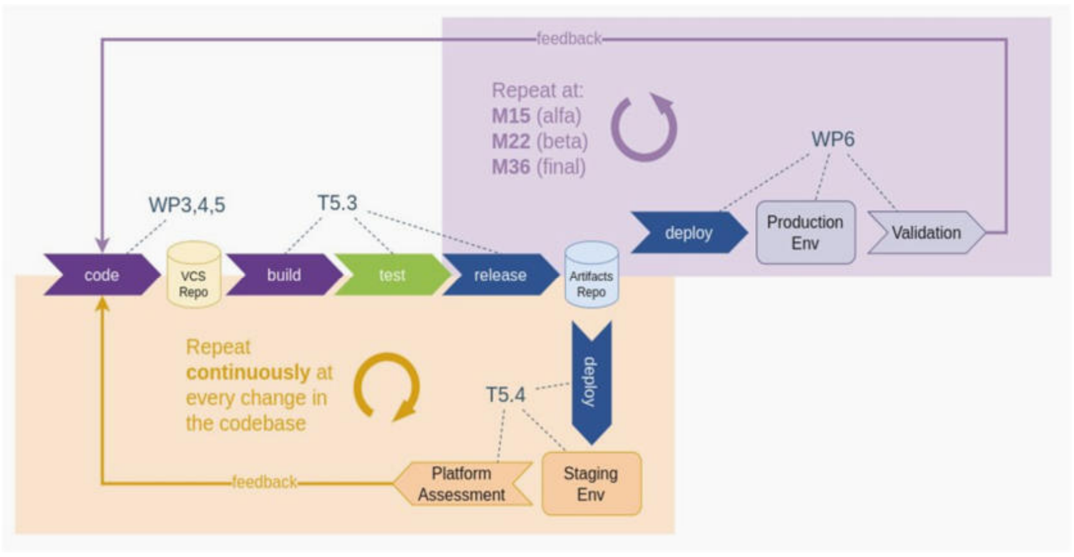


Figure 40: Proposed CI/CD process

A key factor for the integration process to be effective is the automation of these activities to be able to provide immediate and reliable feedback to the developers. To this end, a set of automation technologies will be selected and adopted in Work Package 5. Some candidate technologies considered are:

- **GitLab**[52] collaborative and integration platform to host source code, issue tracker, run automated build and test.
- **SonarQube**[53] as the main tool to check quality metrics for the ICOS code. Metrics collected by multiple testing agents will be collected by SonarQube and the overall quality of the code will be assessed.
- **GoHarbor**[54] and **Artifactory**[55] as repositories to store released artefacts produced in the project.

[55] https://jfrog.com/artifactory/

Additional tools will be considered in the context of WP5 activities, especially for the testing and validation activities, with the goal of automating and enlarging the coverage of quality assessment in the ICOS software.

6.3 Release Plan

ICOS adopts an iterative and incremental development process to improve the definition and the implementation of the system continuously during the project lifetime. There are two main iterations in the project (depicted in Figure 41) each of them composed by three distinct phases: research, implementation, integration, and validation.

This strategy allows to refine and improve the ICOS System in the second iteration of the project based on the feedback received from the development, integration and validation tasks executed in the first iteration.

From the software point of view, during the project there will be three different releases of the ICOS System (Figure 41):

- ICOS Alfa release at M15 (November 2023).
- ICOS Beta release at M22 (June 2024) and
- ICOS Final release at M32 (April 2025).

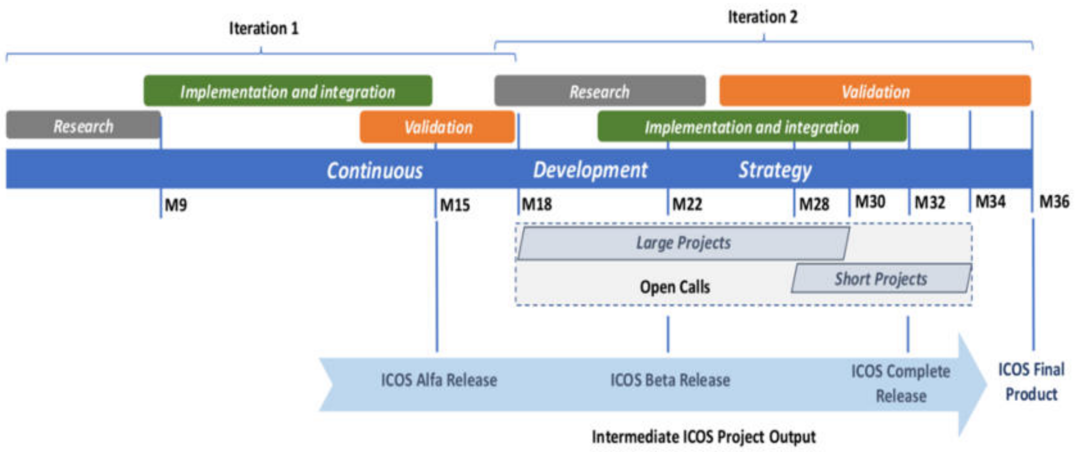


Figure 41: ICOS System Release.

Not all functionalities of ICOS will be delivered in the first release at M15, but they will be delivered incrementally in the three releases. An initial **plan for the releasing** of the functionalities is presented in Figure 41. In addition to the planned functionalities, ICOS Beta and ICOS Final releases will also contain improvements and refinements based on the feedback provided by the testing and validation activities in the first iteration as well as new/modified functionalities defined in the second version of the architecture (at M22, deliverable D2.4 [3]) and the feedback from the "Financial Support to Third Parties" (FSTP) projects.

It is worth noting that this plan has been created with the current knowledge and status of the project and it is highly probable that minor aspects will change (especially for Beta and Final releases) over the course of the project to adapt to unexpected events in the projects, new or modified functionalities in the second iteration and feedback received from the validation and the FSTP projects. Updates to this plan will be included in the second version of the architecture deliverable (D2.4 [3]) as well as in the integration work package deliverables (D5.1 [12] and D5.2 [13]).

Table 9: Functionalities ICOS Beta and ICOS Final releases.

Layer	Alfa Release	Beta Release
	M15	M22
Distributed Meta-Kernel	Node On-Boarding (*SUC_CC_1*) Application Descriptor Definition (*SUC_RT_1*) Basic Application Deployment (*SUC_RT_4*) Collect and visualise Metrics and Logs (*SUC_RT_6*, *SUC_RT_7*)	Visualise Continuum Topology (*SUC_CC_5*) Deployment offload to a different or multiple Controllers (*SUC_RT_5*) Review and apply deployment optimisations and recovery actions (*SUC_RT_8*, *SUC_RT_9*, *SUC_RT_10*)
Data Management	Store and access data (*SUC_DM_1*, *SUC_DM2*)	Execute Data Processing Functions (*SUC_DM_3*)
Security	User AuthN/AuthZ, Users Management (*SUC_SC_1*, *SUC_SC_2*) Secure Connections and Trust between Nodes (*SUC_SC_9*, *SUC_SC_10*) Security Coordination APIs	Service-to-Service and Controller-to-Controller AuthN/AuthZ (*SUC_SC_1*, *SUC_SC_2*) Compliance Analysis (*SUC_SC_5*) Security Assessment (*SUC_SC_8*) Audit Analysis (*SUC_SC_4*)
Intelligence	Basic Models Management (*SUC_AI_1*) Basic models training (*SUC_AI_2*, *SUC_AI_3*) Intelligence Coordination APIs	Automated models training deployment and execution (*SUC_AI_2*, *SUC_AI_5*) Nodes Classification Models (*SUC_AI_7*) Anomalies Detection Models (*SUC_AI_8*) Policies violation prediction Models (*SUC_AI_9*)
ICOS Shell	Basic CLI	Complete CLI Basic GUI

7 Conclusions

The document presented the initial ICOS System architecture as the main outcome of project's task "T2.4 - Architectural Design". The architecture has been defined starting from the analysis of User Stories and Requirements collected during the first months of the project and reported in the deliverable "D2.1 - ICOS ecosystem: Technologies, requirements and state of the art (IT-1)" [1]. That deliverable was also the primary source of information for the identification of the actors of the system as well as the technological stacks on top of which the ICOS System will be based.

All the project's partners collaborated to the definition of the architecture following a well-defined methodology that started with the formalisation of common guiding principles for evaluating the architectural decisions (i.e., Functional Suitability, Performance Efficiency, Scalability, Robustness, Trust, and Privacy). Then, the work continued with a series of activities conducted collaboratively by all partners with virtual and physical meetings, brainstorming sessions, proposal and review of models and diagrams. The resulting architecture has been described following the ISO/IEC/IEEE 42010:2022 standard to ensure that all the stakeholders and concerns are sufficiently addressed. In particular, the architecture is described from different point of views following the viewpoints proposed by the "Kruchten's 4+1 Views" framework: from a functionality point of view (presented in section 3), from a logical/structural point of view (presented in section 4) and from an operational/runtime point of view (presented in section 5). In addition, an initial and preliminary analysis from the implementation and physical point of views is provided in section 5.8.

From the functionality point of view, the User Stories and Requirements have been formalised in a set of forty-one different System Use Cases that represents atomic functionalities that the system will provide to its users and that will generate a value for them. System Use Cases have been numbered to allow traceability in all the next phases of design and implementation.

The ICOS System has been decomposed on a set of logical components to better manage its complexity and analyse its design. Two types of ICOS Nodes have been identified:

i) the **ICOS Controller** (responsible for managing the continuum and the run-time) and
ii) the **ICOS Agent** (responsible for executing offloaded users' services).

The ICOS software (that will run on all the ICOS Nodes) has been logically decomposed in multiple layers:

- The **Distributed Meta-Kernel Layer** responsible for the management of the Continuum and the Runtime Manager which is responsible for the execution of user's applications.
- The **Security Layer** responsible for guaranteeing the security aspects in a proactive way for all operations and at all levels: from devices and users' authentication and authorisation to application behaviour anomaly detection and compliance enforcement.
- The **Intelligence Layer** responsible to provide artificial intelligence capabilities to ICOS operations and decisions like optimisation of application deployment and resources usage, suggestion of recovery actions and predictive analysis of monitoring data.
- The **Data Management Layer**: a horizontal component that will be used by the other layers to store, exchange, and process data within the system in a secure and distributed manner.
- The **ICOS Shell Layer**: responsible for exposing all the system functionalities to the users providing graphical and command line interfaces, management, and DevOps tools.

The composition of each layer is analysed in the document identifying its internal decomposition in functional components and their relationships.

An analysis of the ICOS System from an operational point of view has been conducted and reported in the document. The Consortium focused in particular on the functionalities considered of higher priority in the first iteration and that could hinder risks for the progress of the design and the implementation of the system. The ones analysed are the i) on-boarding of ICOS Agents, ii) the deployment of user' services, iii) the execution of user' services, iv) the optimisation of applications resources, v) management of policies, vi) the identity and access management, vii) the anomalies detection through application logs and viii) the usage of the system functionalities through the ICOS Shell. For each of these functionalities a deep analysis has been conducted on how it is realised, what is the role of each component of the architecture in its realisation and the interactions between ICOS components needed to achieve it. This allowed to early identify the main integration points between components, the communication paradigms and the protocols facilitating the following design and implementation phases.

Finally, in order to prepare the technical work in the Work Packages WP3, WP4 and WP5 and hand-over the design and implementation of the system to them, some initial decisions are provided on i) the technological stack that will be adopted for the implementation, integration and testing of the system, ii) the responsible partners for the different components and iii) the integration process that will lead from the single components to a uniform ICOS System release.

The architecture defined in this document will be the basis for all the technical work that will be carried out in the first period (up to M18) of the project in the technical Work Packages. However, it is expected that, as the analysis and design activities will progress, the architecture of the system will evolve and improve. In addition, the feedback received by the evaluation of the system in the project's use cases (WP6) will also a source of improvement for the system architecture. For this reason, in the second project period a new iteration will start and a new, final, version of the ICOS System architecture will be delivered at M22 and documented in deliverable "D2.4 - ICOS architectural design (IT-2)".

8 References

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

[2] ISO/IEC/IEEE 42010:2022- "*Software, systems and enterprise -Architecture description"* (November 2022), https://www.iso.org/standard/74393.html retrieved on May 8th, 2023.

[3] ICOS. D2.4 –" *ICOS Architecture Design IT-2 at M22"*. Grant-Agreement-101070177-ICOS.

[4] ISO/IEC 25010-"*Software product quality models"*, (November,2011), http://www.iso25000.it/styled-8 http://www.iso25000.it/styled-8/http://www.iso25000.it/styled-8/retrieved on May 16th, 2023.

[5] ISO/IEC/IEEE 42010:2011- "*System and Software engineering – Architecture Description",* (December,2012), https://www.iso.org/standard/50508.html retrieved on May 8th, 2023.

[6] Nick Rozanski, Eoin woods -*"Applying Viewpoints and Views to Software Architecture"*, published by Addison Wesley, (2005) https://www.viewpoints-and-perspectives.info/vpandp/wp-content/themes/secondedition/doc/VPandV_WhitePaper.pdf, retrieved on May 8th, 2023.

[7] Philippe Kruchten - *"Architectural Blueprints-The "4+1" View Model of Software Architecture"* (November 1995), https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdfhttps://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf retrieved on May 12th, 2023.

[8] C4model - *"The C4 model for visualising software architecture"*, https://c4model.com/ , retrieved on May 12th, 2023.

[9] John A. Zachman - *"The Concise definition of the Zachman Framework",* https://www.zachman.com/about-the-zachman-framework retrieved on May 12th, 2023.

[10] Lothar Bormann, & Frances Newbery Paulish, Working Conference on Software Architecture, WICSA 1999, Software Architecture , pp 529-543, *Software Architecture at Siemens: the challenges our approaches, and some open issues,* https://link.springer.com/chapter/10.1007/978-0-387-35563-4_31https://link.springer.com/chapter/10.1007/978-0-387-35563-4_31 retrieved on May 12th, 2023.

[11] The Opengroup, TOGAF, https://www.opengroup.org/togaf retrieved on May 12th, 2023.

[12] ICOS. D5.1 - *First ICOS release: ICOS Alfa Version at M15.* Gran-Agreement-101070177-ICOS.

[13] ICOS. D5.2 - *Second ICOS Release: ICOS Beta Version at M22*. Gran-Agreement-101070177-ICOS.

[14] Scott Rose, Oliver Borchert, Stu Mitchell, Sean Connelly, (August 2020)-NIST Special Publication 800-207, "*Zero Trust Architecture*", https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

[15] Liam Collins, Hamed Hassani, Aryan Moktari, Sanjay Shakkottai, (May 2022), "*FedAvg with Fine Tuning: Local Updates Lead to Representation Learning",* https://arxiv.org/abs/2205.13692, retrieve on May 16th, 2023.

[16] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, Virginia Smith, (April 2020) ,"*Federated Optimization in Heterogenous Networks",* https://arxiv.org/abs/1812.06127https://arxiv.org/abs/1812.06127, retrieved on May 16th, 2023.

[17] Manoj Ghuhan Arivanzhagan, Vinay Aggarwal, Aaditya Kumar Singh, Sunav Choudhary,(December 2019), "*Federated Learning with Personalization Layers",* https://arxiv.org/abs/1912.00818

[18] J. Antic et al., 19th International Conference on the Design of Reliable Communication Networks (DRCN), Vilanova I la Geltru, Spain, 2023, pp. 1-5, *"Runtime security monitoring by an interplay between rule matching and deep learning-based anomaly detection on logs"*, (April 2023), https://ieeexplore.ieee.org/abstract/document/10108105

[19] F. Aguiló–Gost, E. Simó–Mezquita, E. Marín–Tordera, & A. Hussain,"*A Machine Learning IDS for Known and Unknown Anomalies."* (April 2022), https://zenodo.org/record/6473378#.ZGZEDXZBxhG

[20] Lordan, F., Tejedor, E., Ejarque, J. *et al*: J Grid Computing, *"ServiceSs:An Interoperable Programming Framework for the Cloud.",* Journal of Grid Computing ,12(1), pp. 67–91 (March 2014), https://doi.org/10.1007/s10723-013-9272-5