D2.4 ICOS Architectural Design (IT-2)

Document Identification			
Status	Final	**Due Date**	30/06/2024
Version	1.0	**Submission Date**	08/07/2024

Related WP	WP2	Document Reference	D2.4
Related Deliverable(s)	D2.1, D2.2, D3.1, D4.1	Dissemination Level (*)	PU
Lead Participant	UPC	Lead Author	Jordi García
Contributors	ATOS, BSC, CEADAR, ENG, IBM, NCSRD, SixsQ, TUBS, UPC, XLAB, ZSCALE	Reviewers	Marina Giordanino (CRF)
			Marta Łakomiak (L-PIT)

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

Document Information

List of Contributors	
Name	Partner
Jordi Garcia	UPC
Xavi Masip-Bruin	UPC
Montse Farreras	UPC
Sergi Sánchez	UPC
Ester Simó	UPC
Marisa Zaragozá	UPC
Francesco D'Andria	ATOS
Alex Volkov	ATOS
Manuel Gallardo	ATOS
Francesc Lordan	BSC
Àlex Barceló	BSC
Andrés L. Suárez-Cetrulo	CEADAR
Jaydeep Samanta	CEADAR
Ricardo Simón Carbajo	CEADAR
Sebastián Cajas Ordoñez	CEADAR
Gabriele Giammatteo	ENG
Maria Antonietta Di Girolamo	ENG
Kalman Meth	IBM
George Xiorius	NCSRD
John White	SixsQ
Konstantin Skaburskas	SixsQ
Admela Jukan	TUBS
Fin Gentzen	TUBS
Jasenka Dizdarevic	TUBS
Marc Michalke	TUBS
Hrvoje Ratkajec	XLAB
Daniel Nikoloski	XLAB
Aleš Černivec	XLAB
Ivan Paez	ZSCALE

Document History			
Version	Date	Change editors	Changes
0.1	03/07/2024	Jordi García (UPC)	First version of the document
0.2	05/07/2024	Jordi García (UPC)	Updated with the peer reviewers' comments
1.0	08/07/2024	Carmen San Román (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Jordi García (UPC)	05/07/2024
Quality manager	Carmen San Román (ATOS)	08/07/2024
Project Coordinator	Francesco D'Andria (ATOS)	08/07/2024

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
COE	Container Orchestration Engine
CLI	Command Line Interface
CNI	Container Network Interface
CRUD	Create, Replace, Update, Delete
DM	Deployment Manager
FL	Federated Learning
GUI	Graphical User Interface
InfDev	Infrastructure Device
IAM	Identity and Access Management
JM	Job Manager
MM	Matchmaker
OCM	Open Cluster Management
OWASP	Open Web Application Security Project
OM	Onboarding Manager
OO	Onboarding Operator
PM	Policy Manager
TA	Threat actors
TB	Trust Boundaries
VM	Virtual Machine

Executive Summary

This deliverable presents the final architecture for the complete ICOS System leveraging and extending the work carried out by the Consortium within the first project iteration (M1-M18), aimed at reviewing and updating the preliminary architecture introduced in the document “D2.2 - ICOS Architectural Design (IT-1)” [1], that was defined to satisfy the requirements elicited in the document “D2.1 - ICOS ecosystem: Technologies, requirements and state of the art (IT-1)” [2]. The proposed architecture includes the whole set of functionalities needed to achieve the project’s objectives and to address the project’s use cases’ needs.

Specifically, this document presents the ICOS System architecture from two different points of view:

i) a logical view, that presents the static structure of the system decomposed into functional components, and; ii) a distinct point of view where the basic functionalities offered by the system to the user are identified and formalised.

The logical architecture has been organised to meet the different roles identified in ICOS, namely, the ICOS Controller, the ICOS Agent, the ICOS Shell and the ICOS Lighthouse. The ICOS Controller, the most complex actor, has been designed through a three-layer architecture, including the Meta-Kernel Layer, the Intelligence Layer, and the Security Layer, enriched by an additional transversal Data Management Layer. For each role and layer, the complete set of components considered toward a successful development are deeply described.

The primary objective of this deliverable is to align all technical partners toward a common set of principles and components for the design of the envisioned ICOS system, along with a common vocabulary and methodology to describe the system. This will undoubtedly allow 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 because partners will work in separated teams in the technical Work Packages during the implementation of the system.

1 Introduction

1.1 Purpose of the document

This document presents the outcomes of the work carried out as part of the task "T2.4 - Architectural Design". The main purpose of this task is the design of the final ICOS architecture and the definition of the functional design of the ICOS platform. The design presented in this document is being developed as part of WP3, WP4 and WP5, turning into the proposed ICOS releases to be validated in WP6.

The deliverable presents an overview of the ICOS system, and clearly describes the architecture as well as the architectural components, and describes their functionalities, for each of the defined system roles: The ICOS Controller, the ICOS Agent, the ICOS Shell and the ICOS Lighthouse.

The document presents the architecture under two different perspectives:

- A logical view, that shows the static system structure for each ICOS suite, decomposed in architectural components with well-defined functionalities and interfaces.
- A functional view, that describes the behaviour of the system at runtime through sequence diagrams, and describes the interactions and the synchronisation mechanisms between the different architectural components.

This document is the result of a thoughtful review of the initial architecture presented in deliverable D2.2 "ICOS Architectural Design (IT-1)" and corrects and includes those components and functionalities that have been missing in the initial ICOS architecture.

All the ICOS Consortium has collaborated on the definition of the architecture, either as technical partners contributing on the design, or as pilot partners contributing on the detection of design gaps or missing functionalities.

1.2 Relation to other project work

This deliverable presents the final ICOS architecture logical and functional views, and is a review of the initial architecture design presented in deliverable D2.2. The inputs received are the following:

- Deliverable D2.2: ICOS Architectural Design (IT-1) [1], submitted in M09, that is the main reference for this deliverable, as this is an update of D2.2;
- Deliverable D2.1: ICOS ecosystem: Technologies, requirements, and state of the art (IT-1) [2], submitted in M06, that summarises the work conducted since the beginning of the project in the task "T2.1 – Ecosystem identification: Baseline technologies" concerning the description of use cases, elicitation of requirements, and the analysis of the state of the art for baseline technologies;
- Deliverable D3.1: Meta-Kernel Layer Module Integration [3], submitted in M13, which provided valuable input to correct some missing details for the Meta-Kernel Layer;
- Deliverable D4.1: Data management, Intelligence and Security Layers [4], submitted in M13, which provided valuable input to detail and correct some missing details for the Intelligence and Security Layers.

This document has also been fed from the outcomes of WP3, WP4 and WP5 tasks that have implemented the original design of the ICOS architecture and identified components with missing functionalities and solved technological challenges that had not been foreseen during the preliminary ICOS architecture design in IT-1.

Additionally, this 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.

1.3 Structure of the document

The deliverable has been organized in 5 main sections:

1 **Introduction** (this section), presents the purpose of the document, identifies the relationships to other project work, introduces its content and structure, and defines the glossary adopted in this document.

2 **Differences with respect to D2.2**, highlights the main differences of the final ICOS architecture with respect to the original architecture presented in M09.

3 **Architecture: Logical view,** presents the architecture from a logical view, including the components description in the 3 layers of the ICOS Controller, the ICOS Agent, the ICOS Shell and the Lighthouse.

4 **Architecture: Functional view**, describes the ICOS operation through a comprehensive set of sequence diagrams, and detailing the Cloud Continuum Operations lifecycle supported by ICOS.

5 **Conclusions**, summarizes 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

- **Cloud Continuum.** An aggregation of heterogenous resources (CPU, memory, storage, networks, IoT devices, intelligence) managed seamlessly end-to-end that may span across different administrative/technology domains in multi-operator and multi-tenant settings.
- **Deployment Manager.** Deployment Manager is part of ICOS agent that deploys through specific orchestrators that is OCM or Nuvla and manages the deployment accordingly.
- **ICOS Agent.** The ICOS software component that needs to be installed on each node with actual underlying orchestrator (OCM or Nuvla) that manages the infrastructure where the jobs are executed. ICOS Agents are distributed throughout the continuum.
- **ICOS Application Descriptor.** A yaml file consisting of ICOS specific application header that includes the required meta information of each component followed by list of the application components manifest.
- **ICOS Node.** Any resource, physical or virtual, that is running either the ICOS software (can be either an ICOS Controller or ICOS Agent).
- **ICOS Controller.** The ICOS Controller is responsible for managing the ICOS continuum, which consists of a set of agents based on proximity criteria, by providing control, intelligence decision-making and lifecycle management.
- **ICOS Lighthouse.** The ICOS software component that handles the distribution and registration of multiple controllers addresses of ICOS ecosystem while serving as the system controller registry.
- **ICOS Shell.** The ICOS shell is an interface that manages the user's interaction to the ICOS system by providing application details plus the infrastructure availability through GUI. Additionally, it provides the capability of user's identity and authentication login.

2 Differences with respect to D2.2

The preliminary ICOS architecture presented in deliverable D2.2 [1], as part of the logical view, has been enhanced and evolved into the ICOS final architecture. ICOS has been designed as a three-layered architecture, including the Distributed Meta-Kernel Layer, the Intelligence Layer, and the Security Layer, as well as a transversal Data Management Layer. The next section provides a complete description of the final ICOS architecture; however, this section highlights and describes the main differences between the initial and final ICOS architectures.

The original Distributed Meta-Kernel Layer was defined in D2.2 as in Figure 1 where all changes have been highlighted in red:

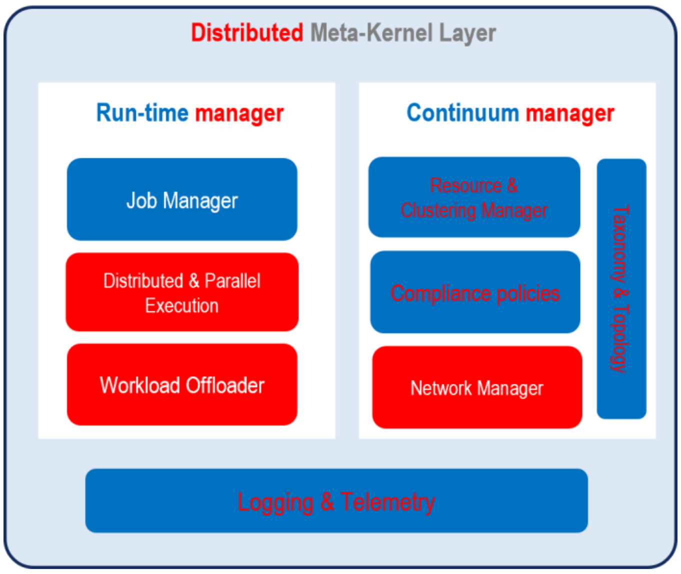


Figure 1. Old ICOS Distributed Meta-Kernel architecture

The changes are the following:

- The general architecture has been mapped to the **ICOS Controller** and the **ICOS Agent**, so two different architectures will be presented (sections 3.2 and 3.3 respectively).
- Some minor renaming has been done for clarity and consistency purposes:
- Distributed Meta-Kernel Layer → Meta-Kernel Layer
- Run-time manager → Runtime management
- Continuum manager → Continuum management
- The **Resource & Clustering Manager** module has been renamed as **Resources Onboarding**, and limits its scope to the management of the onboarding procedures.
- The **Compliance policies** module has been renamed to **Policy Manager**, and moved to Runtime management, as the tasks performed in this module are related to applications' execution.
- The **Taxonomy & Telemetry** module has been renamed as **Aggregator**, and has simplified the original tasks leveraging the capability of the collected telemetry.
- The **Network Manager** module has been removed from the architecture. Instead, the networking capabilities are considered in the match making process to find an optimal layout.
- The match making functionality of the **Job Manager** module has been extracted into a new module: the **Matchmaker** module.
- The Logging & Telemetry module has been renamed as **Telemetry Controller** To highlight the control tasks to be performed.

- The **Distributed & Parallel Execution** and **Workload Offloader** modules were defined in the original architecture to be part of the ICOS Agent. However, we've decided to remove these from the generic part of the architecture and offer them as a service that can be added at the application level. This decision was made based on the following three pillars:
  1. Not all applications require these features: Including them by default would add unnecessary overhead for applications that do not need parallel, distributed computation, or offloading capabilities.
  2. Flexibility in choosing the best-suited libraries: multiple libraries providing parallel, distributed computation, and offloading functionalities are available. By not embedding these components into the ICOS Agent, we allow users the freedom to choose the library that best fits their needs without requiring significant adaptations.
  3. Knowledge of application deployment requirements: These libraries often impose specific requirements on ICOS regarding the deployment knowledge of applications. To facilitate the integration and configuration of advanced computation capabilities at the application level, we have introduced the App Setup Manager module. This component ensures that applications can efficiently manage their setup processes and configurations by providing them with all the necessary information from the meta-OS in a standardised and controlled manner.

  With this change, the ICOS Agent remains lightweight and efficient while providing the necessary flexibility and support for advanced computation capabilities through external libraries as needed. The App Setup Manager thus plays a pivotal role in maintaining the modularity and adaptability of the ICOS framework.

With respect to the original Intelligence Layer, it was defined in D2.2 as in Figure 2, where changes have been highlighted in red:

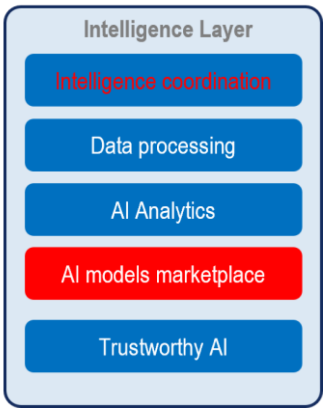


Figure 2. Old ICOS Intelligence Architecture

The changes are the following:

- Some minor renaming has been done for clarity and consistency purposes:
  - Intelligence coordination → Intelligence Layer Coordination API
- The **AI models marketplace** has been renamed as **AI models repository** and removed from the Controller (it will be located in a repository in the cloud).

Although the Intelligence Layer components are in summary the same as in the initial architecture, all components' functionalities have been defined in more detail in the following sections.

And finally, the original Security Layer was defined in D2.2 as in Figure 3, where changes have been highlighted in red:

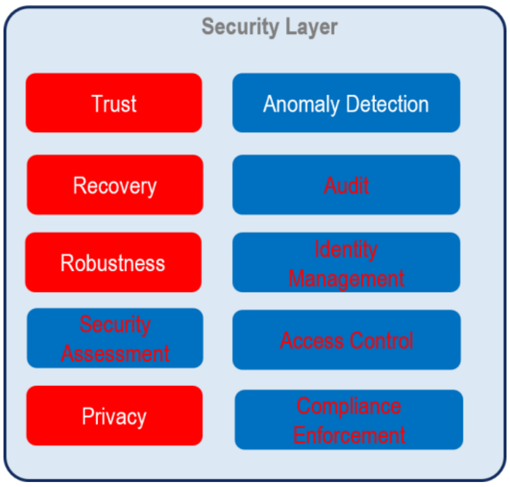


Figure 3. Old ICOS Security Architecture

The changes are the following:

- A new module, named **Security Layer Coordination**, has been added (will be described in subsection 3.2.3.1).
- The Audit module has been distributed into the Controller and the Agent, creating the **Audit agent** and **Audit server**.
- The **Security Assessment** and the **Compliance Enforcement** modules have been combined into a single module, the **Security Scan agent/server**.
- The **Identity Management** and the **Access Control** modules have been combined into a single module, the **Identity and Access Management**.
- The Security Vulnerability Mitigation tasks (**Recovery** and **Robustness** modules) have been removed, as they are implemented as part of the Meta-Kernel layer tasks.
- The Trust and Privacy modules have been removed, as they are an architecture-wide functionality in ICOS.

In the next section, the final ICOS architecture is presented and fully described.

3 Architecture: Logical view

3.1 System Overview

ICOS has been conceived as a dynamic and elastic metaOS platform distributed across the continuum. The primary tasks of ICOS are twofold: a detailed management of heterogeneous resources in real-time, and an efficient deployment of applications on multiple nodes along the continuum. The ICOS design has been defined through two different roles: the **ICOS Controller** and the **ICOS Agent**.

On the one hand, the ICOS Controller is responsible for managing the continuum (tracking the current system topology and availability) and managing the runtime (deploying and monitoring application execution on demand). On the other hand, the ICOS Agents are responsible for executing the offloaded users' applications, taking care of code execution, data access, telemetry collection and, eventually, runtime communication with other Agents. There is an ICOS Agent running on each node of the continuum (whether a complex cluster or a constrained device at the edge), and it is the only ICOS software running on the remote infrastructure.

ICOS has been designed as a distributed, multi-controller system, in which all relevant decisions are made at the Controller level. ICOS Agents receive instructions from ICOS Controllers and translate them into infrastructure-specific tasks. With this organization, ICOS becomes a technologically independent platform, where abstract decisions are made without technological restrictions (at the Controller level) and are implemented by leveraging the capabilities of remote infrastructure technologies (at the Agent level).

A typical ICOS scenario is illustrated in Figure 4:

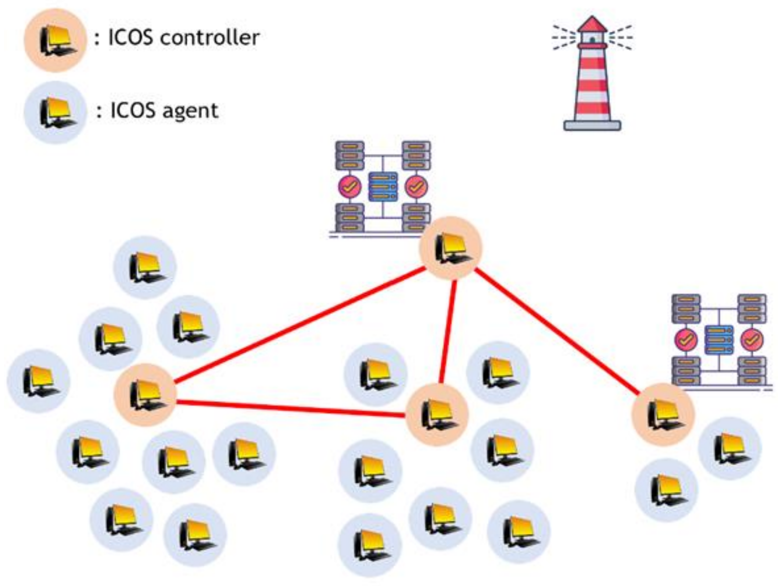


Figure 4. Typical ICOS scenario

ICOS Controllers are distributed along the continuum to leverage locality, providing fast response time and supporting scalability. ICOS Agents are deployed at every node of the continuum. They can range from powerful clusters to constrained computing devices, and provide processing and data accessing capabilities. Each ICOS Controller is in charge of managing a number of ICOS Agents based on locality principles. Upon request for application execution, the Controller will attempt to find locally an appropriate set of nodes within its scope to execute the application according to the application requirements; in case the request cannot be satisfied with the available resources, the Controller will coordinate horizontally with other Controllers to find a global solution for the application request.

In the following sections, a detailed architectural description of the different ICOS roles is performed, named the **ICOS Controller**, the **ICOS Agent**, the **ICOS Shell** and the **ICOS Lighthouse**.

3.2 ICOS Controller

The ICOS Controller has been designed as a three-layer architecture. As shown in Figure 5, the **Meta-Kernel Layer** implements all tasks related to the continuum management, as well as the runtime decision-making and management. This layer is also responsible for collecting telemetry data from the infrastructure through the Agents. The **Intelligence Layer** is fed by the telemetry data, and it is responsible for providing intelligence to the Meta-Kernel layer (both for the continuum and runtime management decisions) as well as providing predictive monitoring to forecast different runtime events (resource utilisation, network load, or security risks, among others). This layer is also responsible for launching training and retraining processes through federated learning to keep the intelligence model updated. Finally, the Security Layer is responsible for providing identity and access management, trusted and encrypted communication channels, and detecting security-related risks (vulnerabilities, threats, or anomalies) and events (audits).

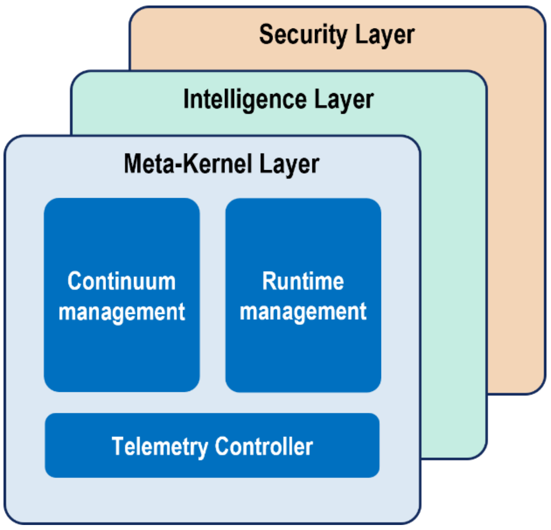


Figure 5. Three-Layer Architecture of ICOS Controller

3.2.1 Meta-Kernel Layer

The ICOS Meta-Kernel Layer is the responsible for the continuum and the runtime management, as well as the telemetry collection and management. Figure 6 shows the ICOS Controller Meta-Kernel Layer architecture, which is basically organised in three main blocks: the **Continuum management**, the **Runtime management**, and the **Telemetry Controller**.

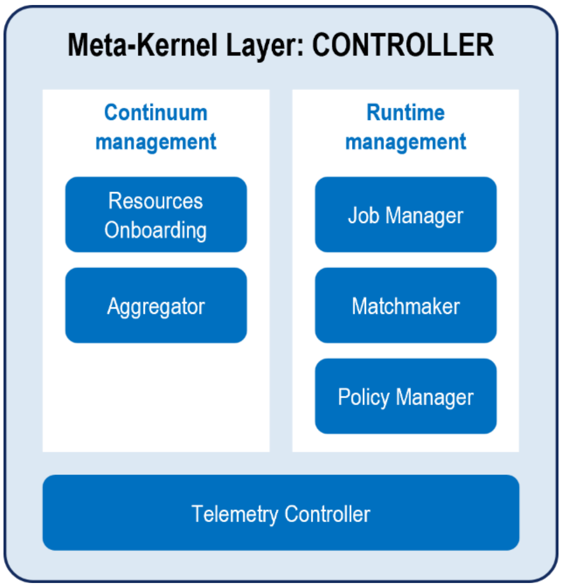


Figure 6. The ICOS Controller Meta-Kernel Layer

In the following sections, each block, together with the components that implement them, are presented and fully described.

3.2.1.1 Continuum management

The Continuum management has been designed through two components: the **Resources Onboarding** and the **Aggregator**.

3.2.1.1.1 Resource Onboarding

Resources that constitute part of the ICOS continuum must undergo an onboarding process. While the specifics may vary from component to component, the main steps are as follows:

- Obtaining the resource credentials from the root of trust.
- Establishing a secure connection with other services.
- Engaging in the specific resource onboarding workflow.

According to the ICOS continuum architecture, the resources and components subject to onboarding include:

- ICOS Controller
- ICOS Agent
- Infrastructure services

The ICOS Controller is onboarded by adding it to the ICOS Lighthouse. The ICOS Agent is onboarded by connecting it to a specific ICOS Controller. Infrastructure services, which include Edge and Cloud resources, are onboarded by integrating them into a specific ICOS Agent. These Edge and Cloud resources can become part of the ICOS continuum either via an Edge and Cloud Orchestrator (e.g., Open Cluster Manager or Nuvla) or by directly onboarding a specific Container Orchestrator Engine (COE) instance.

3.2.1.1.2 Aggregator

The Aggregator component is tightly coupled with the observability stack of the ICOS project. It serves as an abstraction layer for the monitoring and telemetry information gathered across the distributed computing resources and software that are part of the ecosystem.

It is based on a predefined infrastructure taxonomy that serves as a data structure that all consuming components will expect when interacting with the Aggregator. Internally, monitoring and telemetry data are queried from the Telemetry Controller and wrapped in the infrastructure taxonomy data structure.

The data structure served by the Aggregator results in a list of infrastructure components divided into Controllers and Agents, forming a topology of ecosystem nodes with its metadata (name, type, identifier…), vulnerabilities, location, static metrics (CPU architecture, CPU cores, CPU max frequency, storage and so on), dynamic metrics (CPU temperature, free RAM memory, network metrics, etc), network interfaces, and connected IoT devices.

3.2.1.2 Runtime Management

The Runtime management has been designed through three components: the **Job Manager**, the **Matchmaker**, and the **Policy Manager**.

3.2.1.2.1 Job Manager

The Job Manager component represents the core module of the ICOS Controller runtime. This module is responsible for the runtime management and provides control, persistence and the coordination between ICOS components. Such a component enables the continuum to be consistent within real time, furthermore becoming the centre of the truth in the mentioned continuum.

For ICOS to become trustable at runtime, and consistently manage ICOS execution in such a diverse continuum, Job Manager provides the notion of a Job.

A Job defines the minimal executable unit to be managed by ICOS Controller at runtime. For this unit to become executable by the different ICOS Agents, more importantly, without considering their underlying runtime technology. Specific criteria for job's granularity are defined taking into account the following key ICOS concepts:

- **Application Component Description**: It describes, following ICOS syntax, the components an application is composed of, as well as their requirements to be met and policies to be enforced.
- **Target**: Defines the underlying infrastructure able to execute a single job, taking into consideration the mentioned requirements and the capacity the infrastructure piece provides, since appropriate quality of service must be enforced. This target is selected by the Matchmaker (described in the next section) for each job that comprises an application.
- **Resource**: Abstract representation of the actual application component executed within a single target. This representation is retrieved from the orchestrator at runtime during the execution of the corresponding job.
- **ICOS Agent**: The actual underlying orchestrator that manages the infrastructure where the job is executed. From a Job Manager's perspective, whenever an agent takes a job for execution, it becomes the owner of such a job, meaning that only the mentioned agent can manage this job's life cycle.

The Job Manager provides job lifecycle management operations (CRUD) that enable consistent and efficient execution of a job, furthermore, managing both the state of the job and the state of the actual resource within the agent that owes that job.

When an application is composed of multiple components it becomes a set of jobs, in other words, a job group. Job group holds all the information regarding the application, including all the components(jobs) that compose the application, the relationship between the different jobs (application topology) and other information such as requirements and policies such application must meet.

3.2.1.2.2 Matchmaker

The Matchmaker (MM) is the runtime component responsible for the optimal mapping of the different application components on the available ICOS infrastructure. The MM is triggered by the Job Manager (as part of the runtime management) when a new application has to be executed, and uses the Aggregator (as part of the continuum management) to retrieve the availability of the ICOS infrastructure.

The MM receives as input the application description from the Job Manager and the infrastructure description from the Aggregator. The application descriptor consists of a list of the application components to be deployed (including the component manifest, if any) plus an ICOS specific application header that includes the relevant information that the MM should consider to derive the optimal application mapping, for instance, the application constraints, the hardware requirements, the optimization metrics, or some other user-defined policies.

The MM parses the application descriptor, extracts the requirements (computational resources and limits, data requirements, and security and performance related policies) for each component, and filters these requirements from the ICOS infrastructure description. From the set of candidate nodes that fulfil each application components requirements, and considering the interdependencies between components, the MM selects the best set of nodes where the application should be deployed and returns this list to the Job Manager.

Note that as part of the ICOS infrastructure information received from the Aggregator, it also contains the current state of the nodes (i.e., CPU load, RAM utilization, network capabilities, etc), the forecasted state of the nodes (as described in the Intelligence Layer), and the security scoring per node (as described in the Security Layer), which will also be considered in the optimal mapping process.

3.2.1.2.3 Policy Manager

As presented in the deliverable D2.2 [1], the Policies Manager component is responsible for the management of technical and business performance policies associated with user applications and ICOS nodes. The main scope and objectives for this component remained unchanged with regards the first version of the architecture and can be seen in more detail in D2.2.

The Policy Manager is involved in the management of the continuum and the orchestration of user applications. It generates notifications when policies are violated. These notifications are received by other ICOS components (e.g., the Job Manager) and trigger the execution of remediation actions that have the goal of re-establishing the desired behaviour for nodes and user applications. The component uses the Telemetry Controller module (described in the next section) as a source of data to "sense" the current status of the nodes and the applications through the metrics that are collected within the system, evaluate the policies and verify if they are enforced or not.

The Policy Manager defines a policy as composed of three parts:

1 The **subject**: uniquely identify the object to which the policy should be applied. It can be a user application's component (defined by its name, the application name and instance id) or an ICOS node (identified by its agent's id and the node's id);
2 the **spec**: defines the specification of the policy. In particular it defines the metrics and the thresholds to use for its measurement;
3 the **action**: defines the action expected to be executed in case of violation of the policy.

Policies related to an application (or a single component of an application) are defined in the Application Descriptor along with the definition of the application components and requirements.

3.2.1.3 Telemetry Controller

As presented in the deliverable D2.2 [1], the Telemetry Controller (named Logging and Telemetry module in D2.2) implements the ICOS cloud-native monitoring and observability strategies. The main scope and objectives for this component remained unchanged with regards to the first version of the architecture, and can be seen in more detail in D2.2.

The implementation of this module, done for the first development iteration, allowed us to refine the architecture of this module to better address the requirements for ICOS' observability. In particular, the Logging and Telemetry module has a highly distributed architecture that adapts to the distribution of ICOS resources.

As shown in Figure 7, this module is constituted by three different sub-modules that play a different role and needs to be deployed in different parts of the system:

1 Telemetry Controller. It is deployed in the ICOS Controller and receives and aggregates data (metrics and logs) from all the nodes registered to that controller. This component is responsible for the long-term storage of the metrics and provides tools to analyse the data collected.

2 Telemetry Gateway. It is deployed in the ICOS Agent and is responsible for collecting all metrics and logs for that agent.
3 Telemetry Agent. It is deployed in the computational resources (e.g., edge devices, clusters) available to ICOS. This component is responsible for the collection of metrics and logs.

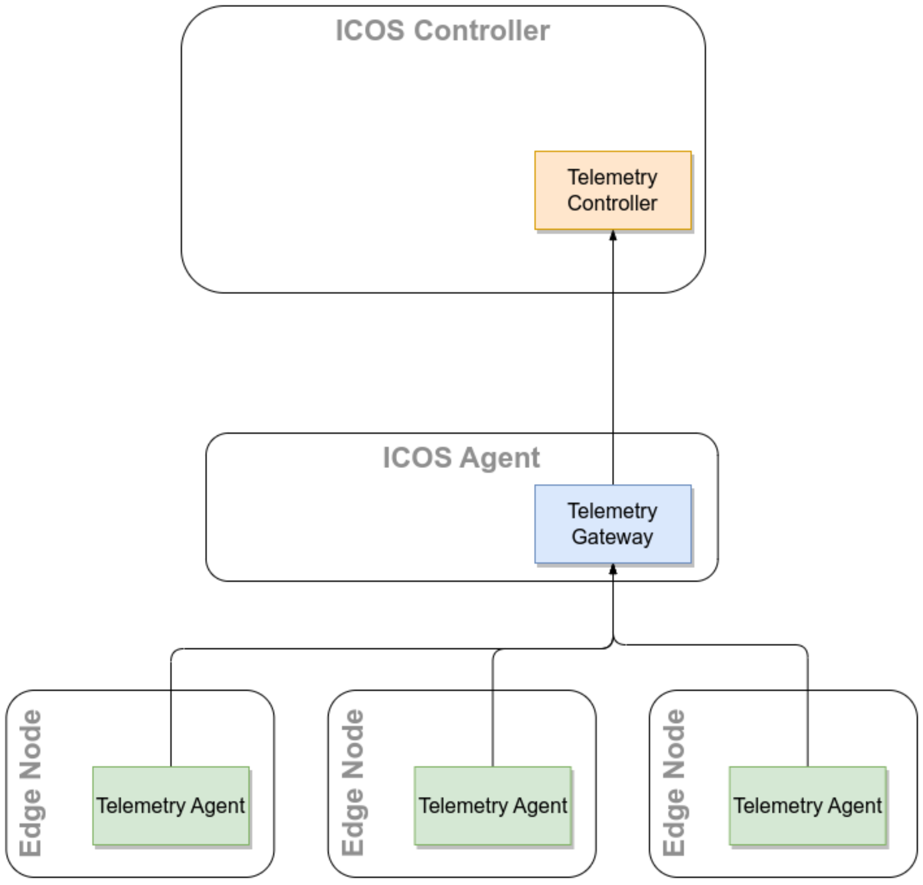


Figure 7. Architecture of the Telemetry Controller

The **Telemetry Controller** represents the main source of data for the other ICOS components in the Controller to know how the ICOS nodes and applications behave. In particular:

- The Aggregator queries the data from the Telemetry Controller to build a system topology (nodes are available with their characteristics and relationships). The topology built by the Aggregator is at the basis of the matchmaking and orchestration mechanisms in ICOS.
- The Policy Manager queries the data from the Telemetry Controller to evaluate the enforcement of the active policies.
- The ICOS Shell can submit queries to the Telemetry Controller on behalf of the user to build charts and dashboards.

The **Telemetry Agent** and the **Telemetry Gateway** are described in more detail in section 3.3 as part of the ICOS Agent description.

3.2.2 Intelligence Layer

This section describes the Intelligence Layer, which facilitates AI models' training, testing, deployment, maintenance, and updating across the edge/cloud spectrum. Its primary goal is to enhance the functions and efficacy of the security and Meta-Kernel layers while trustworthily adhering to data and model usage policies. An initial design was proposed in deliverable 2.2. This is summarised and updated in this section.

Figure 8 shows the architecture of the layer at the ICOS Controller. The Controller stores all the modules described above except the cloud-based model repository. The Intelligence Layer in the Controller suite is accessed through its API, which is pingable inside the Controller itself and by other ICOS Agents living in the same network topology.

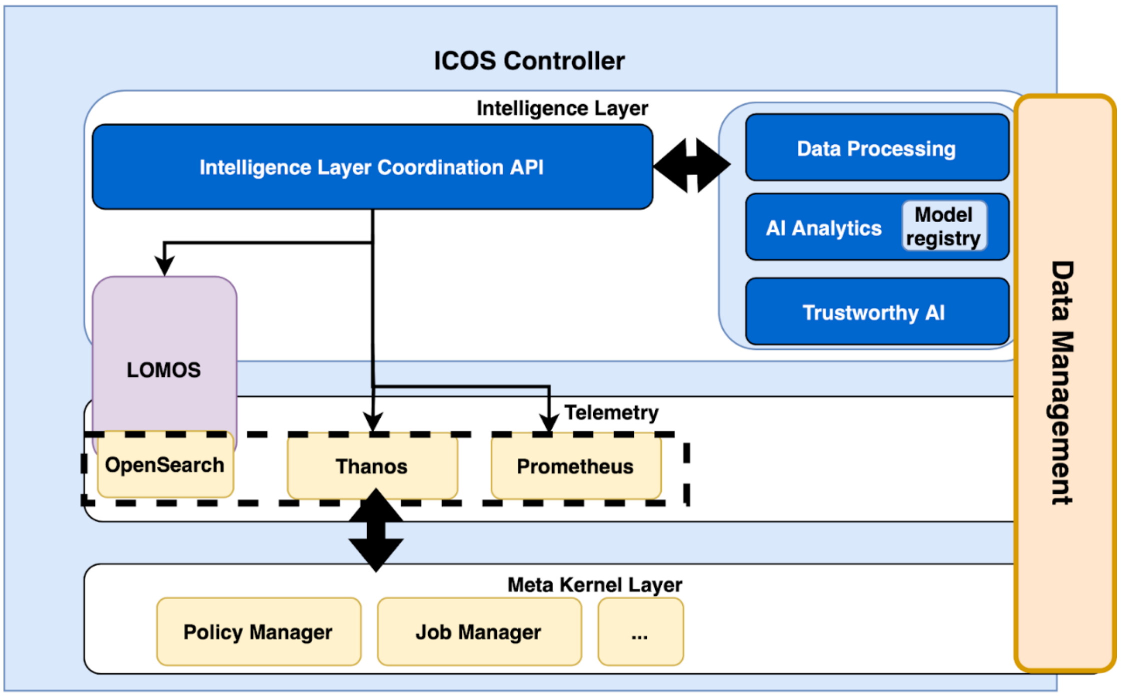


Figure 8. Intelligence Layer at the ICOS Controller

Figure 8 also shows relationships between modules, components, and interfaces at the Controller, which are centralised in the Intelligence Coordination module. Intelligence results are pushed to the Telemetry. LOMOS is used for anomaly detection and runs in a different environment, pushing results to the Telemetry by itself; its sequence diagram is provided in subsection 4.12.1. The Meta-Kernel Layer uses any metrics or forecasts generated by Intelligence by querying the Telemetry. The sequence diagram in subsection 4.12.1 provides a detailed view of the Telemetry, the Meta-Kernel Layer, and Intelligence interactions.

In addition, Figure 8 shows data management as a transversal layer to Intelligence and the meta kernel. It enables access to data within and outside the current node and can also be used to offload AI training tasks to other ICOS nodes. ICOS agents may receive such workloads and train AI models on their premises. The reason to do this is twofold:

- Making use of data locality, thus avoiding data transfers and related privacy concerns.
- Offloading computationally intensive tasks to other ICOS nodes.

The offloading or data access enabled by Data Management is represented by an orange transversal layer in Figure 9.

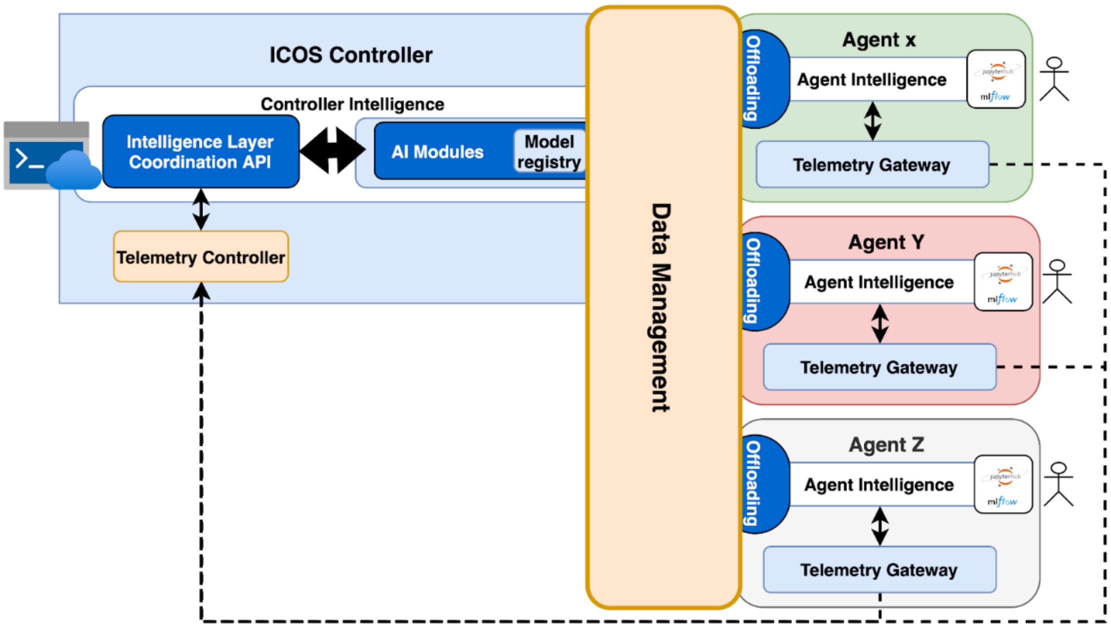


Figure 9. AI offloading and logging between ICOS Controller and Agents

In more details, Figure 9 shows the ICOS Controller on the left-hand side and multiple agents on the right-hand side of the diagram. Agents collect data and push it with Intelligence predictions into the Telemetry Gateway. The Intelligence layer at the controller can also access this data through the Telemetry Controller. While Intelligence for the Meta Kernel Layer is coordinated from the ICOS Controller, AI model training may be offloaded to agents when suitable through the Data Management Layer.

Figure 10 shows the architecture of the layer at ICOS Agents. Intelligence for ICOS Agents owns its own local model registry. This allows Agents:

1 **Training and running their own AI workloads detached from a controller**. This can be done to alleviate the request workload into the Controller, and benefit in use cases where there is no stable connection to an ICOS Controller at all times.
2 **Have personalised AI models**, by downloading models previously trained at the Controller and fine-tuning them with agents' specific data.
3 **Allow ICOS to learn collaboratively in a Federated Learning fashion**. For this purpose, the Controller's model registry will be considered the global model registry that aggregates learning from agents. The Intelligence local endpoint in Figure 10 is present to allow Federated Learning.

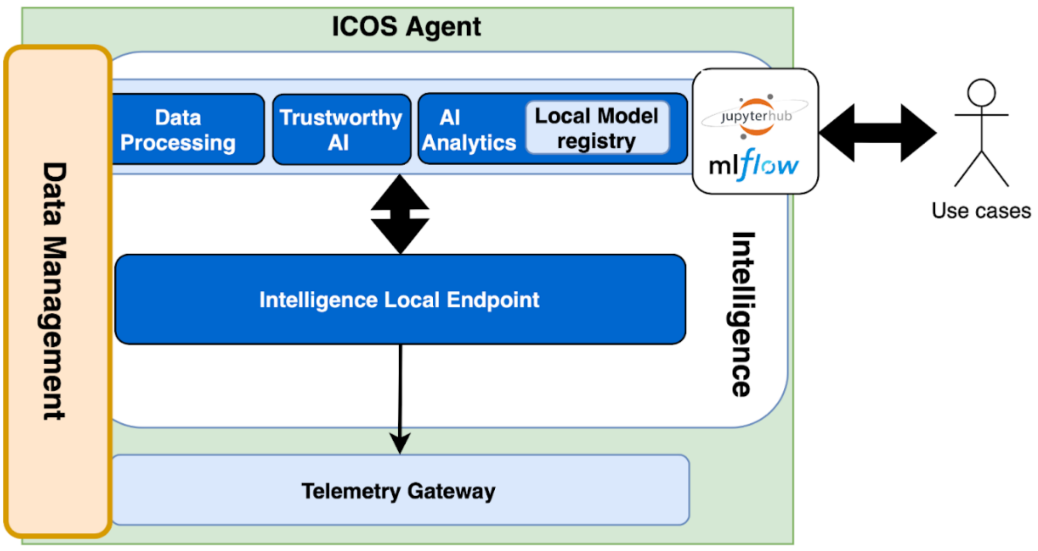


Figure 10. Intelligence at ICOS Agent

Intelligence at the agents also allows offloading AI training workloads to other ICOS nodes using Data Management. It also allows use cases and second-day ICOS users to benefit from the ICOS intelligence layer using the libraries, environment, and model registry available at the node.

The Intelligence Layer contains the following modules: **Intelligence Layer Coordination**, **Data Processing**, **AI Analytics**, **Trustworthy AI**, and **AI Models Repository**. These modules, already described in deliverables 2.2 and 4.1, are summarised in the following subsections.

3.2.2.1 Intelligence Layer Coordination

Intelligence Layer Coordination lives at the ICOS Controller and is the entry point for all AI-related requests. It acts as a point for communication with the other modules of the Intelligence Layer. It coordinates the Meta-Kernel and user layers, providing and requesting services through an API located in an ICOS Controller, which can forward AI workloads to ICOS Agents.

The Intelligence Layer offers a REST API interface for using AI as a service. This interface allows ICOS applications in the Security, Meta-Kernel, and User layers to interact with it and use AI as a service. Deliverable D4.1 describes the technical implementation of this API and its goals in detail, and Deliverable D2.2 gives a more detailed description of this module at the architectural level.

3.2.2.2 Data Processing

The Data Processing module allows data access and storage, as well as pre-processing using the Data Management Layer that allows offloading computing to ICOS Agents. This module is triggered after a request is made to the Intelligence Coordination API. A sequence diagram for this is specified in section 4.12.1.

This module aggregates functions to read and transform data into an AI model format. Deliverable D4.1 provides specifics about AI model pipelines, including data cleaning and processing.

3.2.2.3 AI Analytics

AI Analytics combines techniques and algorithms to train and use AI models to enhance meta-OS and security decision-making tasks. It aims to forecast and create metrics for meta-OS modules like the Job Manager and the Policy Manager. Use cases can benefit from techniques and models deployed in the Intelligence Layer through the ICOS Agent suite. The AI Analytics module owns a model registry with models trained in the ICOS node; Controllers have a registry with a global view, while Agents can have personalised models.

The sequence diagram for model inference and training, showing the interactions between this and the other modules, is specified in section 4.12.1. More details of this module are provided in D2.2 [1] and D4.1 [4] at architectural and technical levels, respectively.

3.2.2.4 Trustworthy AI

The Trustworthy AI module is designed to address data privacy concerns, adding a federated learning function to the Intelligence Layer. It also provides functionality to monitor model metrics and data and explain model predictions through the AIOps frameworks deployed in the Intelligence Layer.

As updated in the deliverable D4.1[4], this module has the following goals:

- *“to ensure trustable, secure & robust model training via federated learning techniques and*
- *to provide model explainability through a series of AI interpretability algorithms to aid the decision-making process of the models while understanding inputs and outputs.”*
- Similarly, a toolset to monitor incoming data and continuously evaluate models is being added to the API to ensure that models can be trusted during potential data shifts. *“Its goal is the automated identification of changes and anomalies in the feature space and model performance”.*

A sequence diagram to understand the federated learning process is provided in section 4.12.2.

Explainable AI visualisations are provided as part of the AIOps technologies that help technical users understand model behaviour and identify bias. This is done to meet Article 13 of the AI Act [5], which requires models to be transparent enough to allow their output to be interpretable and to understand their limitations.

This module was previously introduced in D2.2 and conceptually updated in D4.1. Further technical updates will be presented in deliverable D4.2.

3.2.2.5 AI Models Repository

AI Models Repository is a cloud-based repository that aims to open collaboration between ICOS adopters. It stores previously trained models in ICOS for the Meta-OS, security, and user layers, making them available to the public and users of the Meta-OS.

This cloud-based repository differs from model registries inside the AI Analytics module.

- Model registries have a version of the current models running on the device.
- The AI models repository aims to have versions of prior successful models trained for the meta kernel, security, and user layers by ICOS partners and users.

The AI Model Repository has previously been defined in D2.2. A detailed technical update to this component will be provided in deliverable D4.3.

3.2.3 Security Layer

As introduced in the deliverable D2.2 [1], 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, assessing the security of resources and applications and suggesting remediation or mitigation actions, proactive discovery of anomalous behaviours and security-sensitive events, and verification of the compliance of resources and applications. Trust (identity validation) and Privacy (anonymization and encryption) are included as architecture-wide functionalities and not specific modules.

Figure 11 provides an overview of the Security Layer Architecture. Central Security Layer components are located in the ICOS Controller: **Security Layer Coordination**, **Security Scan server**, **Audit server** and the **Anomaly detection API-2**.

Security Scan agent and **Audit agent** are located in the ICOS nodes and in the K8s cluster where the ICOS Agent is deployed. They provide the low-level security data (vulnerabilities, compliance statuses, kernel events…) for the Security Scan server and Audit server in the ICOS Controller. The Security Scan server processes the data into security scan metrics and transfers them to the Telemetry Controller using the Security Layer Coordination API. The Audit servers process the data into audit metrics and push them directly to the Telemetry Controller.

The **Anomaly Detection module** is located in a different environment and receives data (logs) from the Telemetry Controller and outputs results back to it. Intelligence Coordination API calls the Anomaly detection API-2, located in the ICOS Controller, to obtain these results.

Identity and Access Management module is located outside the ICOS Controller. It is used by ICOS components and ICOS users to authenticate and authorise requests. There is also trusted and secure communication between ICOS components located inside the ICOS Controller and ICOS Agent and between ICOS Controller and ICOS Agent as well as trusted and secure communication between ICOS users and ICOS components.

In the progress towards final Security Layer Architecture, additional work was done to assess the security posture of the ICOS system using Threat Modelling Process from OWASP (Open Web Application Security Project) Foundation. The Threat Modelling Process is a structured approach to application threat modelling that enables to identify, quantify, and address the security risks associated with an application or system. Threat modelling looks at a system from a potential attacker's perspective, as opposed to a defender's viewpoint, and can help increase system security [1].

The result of the Threat Modelling Process is the identification of Threat actors (TA) that could harm ICOS, their attack vectors passing through trust boundaries (TB), threat types and ICOS assets they attack as well as defences in the form of existing or new countermeasures to repel these attacks.

Using this process, new countermeasures were identified that would need to be implemented in ICOS final release and which correspond to the functionalities provided by the Audit module, Security Scan module and Trust.

The results of the process are presented in the form of a Threat modelling diagram Figure 12 and a Threat modelling Table 1.

[1] Threat Modelling Process | OWASP Foundation

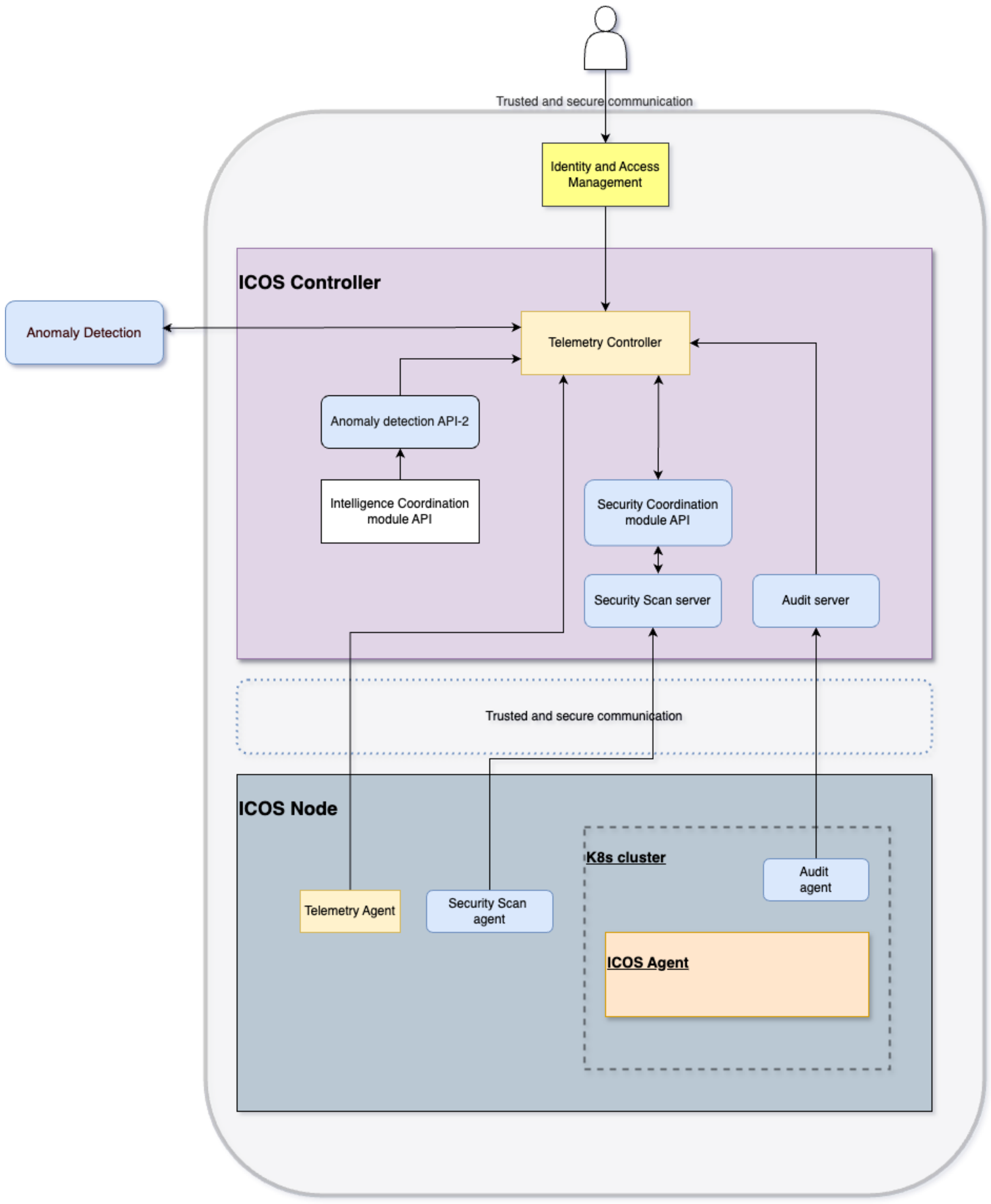


Figure 11. Security Layer Architecture

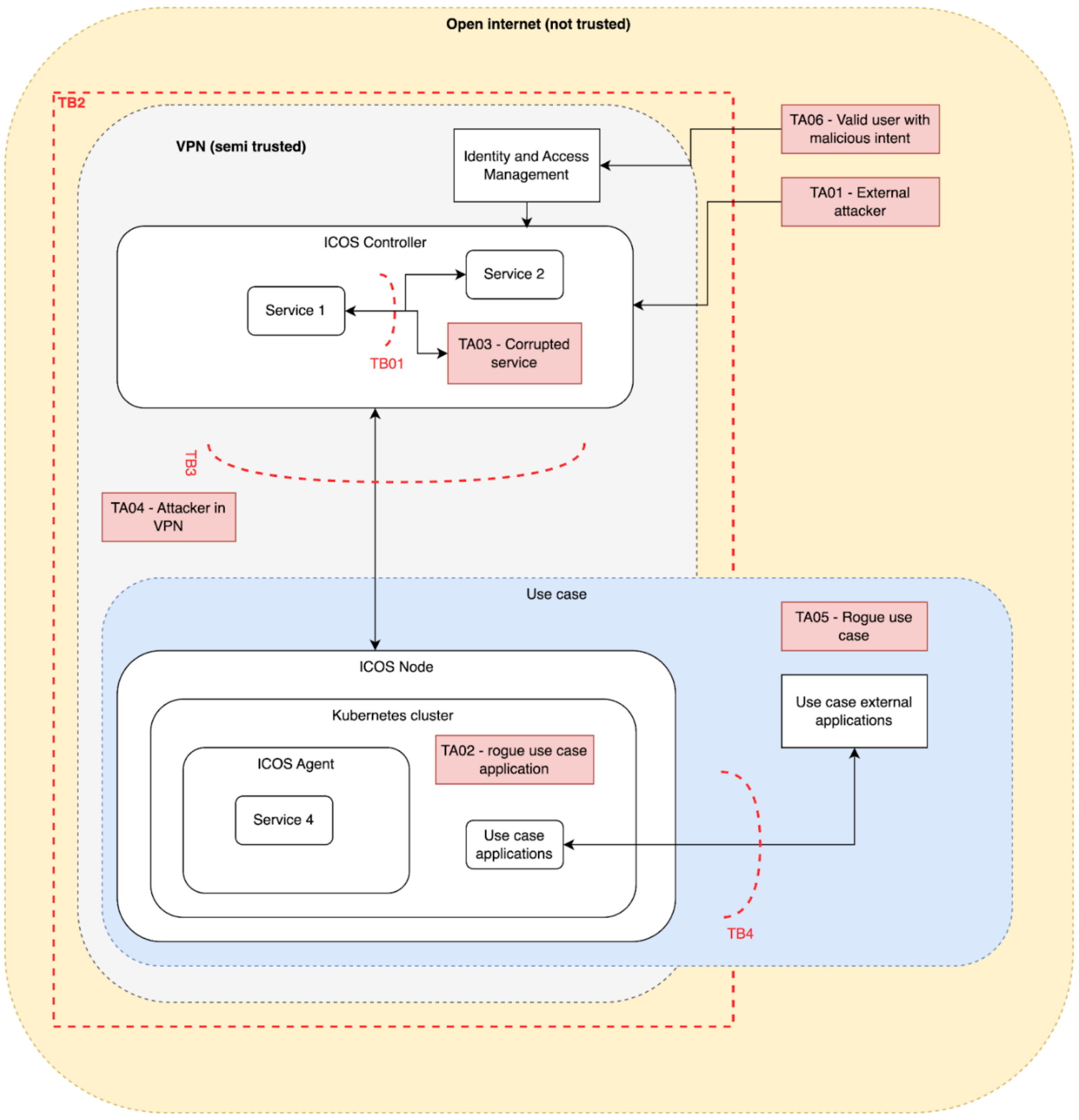


Figure 12. Threat Modelling Diagram for ICOS

Table 1 Threat Modelling Table

ID	Description	Threat	Threat type	Targeted asset	Existing countermeasure	Vulnerability exposed	Risk		Countermeasures to implement
							Likelihood	Impact	
TA01	External attacker	Pretend to be an ICOS service -> access to ICOS Controller	Tampering	Ability to run and manage ICOS services in the ICOS Controller	VPN	VPN Key leaks	Low	High	
			Repudiation		/	No events monitoring	Low	High	Audit
			Spoofing		IAM authentication	Keycloak key leak	Low	High	
TA01	External attacker	Pretent to be an ICOS user -> access to ICOS Controller		Ability to run and manage ICOS services in the ICOS Controller					
			Spoofing		IAM authentication	Stolen credentials	Medium	Medium	
					IAM authentication	Intercepted token (man in the middle)	High (if no HTTPS)	Medium	Trust (certificates)
					IAM authentication	Credentials stuffing	Medium	High	Audit
TA03	Corrupted ICOS service in ICOS Controller	Pretend to be an ICOS service in the ICOS Controller	Tampering	Ability to communicate with and manage services in ICOS Controller and ICOS agents and ICOS nodes	CI/CD pipeline with Trivy and Sonarqube	Infected Docker container with malware	Low	Medium	Code (IaC) and images security scanners
			Repudiation			Malicous dependencies			
			Denial of service						Audit
			Elevation of privilege						
TA02	Corrupted use case application in the Kubernetes cluster		Tampering	Ability to obtain and modify data in the Kubernetes cluster (ICOS services and the use case application)	Use case CI/CD pipeline	Infected Docker container with malware	Low/Medium (Medium as we don't control the use case CI/CD)	Medium	Code security scanners
			Information disclosure						Audit
TA04	Attacker in VPN	Pretend to be an ICOS Controller or an ICOS Agent (man in the middle)	Information disclosure	Read all traffic between ICOS Controller and ICOS Agent	/	VPN Key leaks	Low	High	Mutual certifications between ICOS Controller and ICOS Agent
TA05	Rogue use case	Schedule all jobs on the rogue device	Denial of service	Availability of ICOS Node	Security scan detects what is going wrong with the infra	Metrics coming only from Telemetry producers	Low	High	(Telemetry agent quality metrics, if needed)
TA06	Valid user with malicious intent	Perform operations and deny that the user performed them	Repudiation	Ability to run and manage ICOS services in the ICOS Controller	IAM authentication	/	High	Low	Audit

3.2.3.1 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.

The Security Layer Coordination module API acts as a reverse proxy service for inbound traffic from Intelligence or the Meta-Kernel Layer and as a proxy for outbound communication. This provides additional scheduling and security functionalities (time scheduling and CRON jobs). A complete description of this component can be found in the deliverables D2.2 and D4.1.

3.2.3.2 Anomaly Detection

The Anomaly Detection module is responsible for detecting anomalies in system and application logs. In the final version of the architecture, this module in the Security Layer implements the first workflow, described in D2.2: log-based anomaly detection, consisting of log template extraction, learning normality, and anomaly detection. The module communicates with the API provided by the Intelligence Layer Coordination module in the Intelligence layer to train new models and request inference in already trained models. The logs for training and inference are acquired from the Telemetry component (Meta-Kernel). The result of the inference is the anomaly score, which the Intelligence Layer Coordination module retrieves by calling the Anomaly detection module API.

A complete description of this component can be found in the deliverable D2.2.

3.2.3.3 Security Scan

In the final architecture, this module combines the functionality of two modules from D2.2: Security Scan and Compliance enforcement. Regarding the functionalities of the Security Scan, it provides functionalities of vulnerability detection, security analytics, intrusion detection, file integrity monitoring and malware detection for the ICOS node. Additionally, it provides the functionality of vulnerability detection of application Docker images.

Regarding the functionality of Compliance enforcement, it is able to check compliance of ICOS nodes and application Docker images with main security configuration standards and best practices (e.g., CIS Benchmarks).

A complete description of this component can be found in the deliverables D2.2 and D4.1.

3.2.3.4 Audit

As presented in the deliverable D2.2, the objective of this module is to perform lightweight audit checks to identify potential vulnerabilities and risks in the ICOS node and provide recommendations for improving the security level. The output of audit observability policies is used in the security mitigation mechanism. A complete description of this component can be found in the deliverable D2.2.

3.2.3.5 Security Vulnerability Mitigation

As presented in the deliverable D2.2, the Security Vulnerability Mitigation module operates based on static rule strategy, wherein rule triggering initiates the execution of certain remediation or mitigation processes. These can either be fully independent and automated or require the user's input.

In the final architecture, the functionalities of the Security mitigation module were moved to the Meta-kernel layer, concretely to the Policy manager - Job manager - Deployment manager. The Policy manager uses the Telemetry Controller module as a source of security metrics, provided by the Audit module. It then generates notifications for the Job manager when the security policies are violated. The Job manager triggers the execution of the remediation or mitigation action which is then implemented by the Deployment manager on the ICOS node.

The basic rules and responses are pre-defined in static maps. Additionally, the Intelligence layer's AI Analytics component will be used to offer further mitigation strategies and rules to produce intelligent recognition and amendments of potential system issues.

On the other hand, remediation actions regarding infrastructure on which ICOS is running are out of scope of automated mechanisms and have to be executed manually by the users (e.g., infrastructure provider). This also includes compliance enforcement actions.

A complete description of this component can be found in the deliverable D2.2.

3.2.3.6 Identity and Access Management

As presented in deliverable D2.2, the Identity and Access Management (IAM) module has the objective of ensuring that the right people will have access to the right resources in the system through 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 IAM to ensure the received requests are properly authenticated and authorised. The IAM module is responsible for managing ICOS users and not the users of the user’s applications running in ICOS. A complete description of this component can be found in the deliverable D2.2.

3.2.3.7 Trust and Privacy

As presented in the deliverable D2.2, Trust is an architecture-wide functionality in ICOS, present at three levels:

1 In identification and encrypted communication between ICOS components located inside the Controller and ICOS components located inside the Agent using CNI (Container Network Interface) with VPN encryption technologies (e.g., Wireguard).
2 In identification and encrypted communication between ICOS components in the ICOS Controller and in the ICOS Agent using certificates (e.g., x509) and mTLS.
3 In identification and encrypted communication between ICOS users and ICOS components using VPN technologies (e.g., Wireguard).

Privacy relies on Data Management and Intelligence layers to anonymise data and to use Federated learning to train the AI models at the Edge (ICOS node), eliminating the need to move the (sensitive) raw data across devices or to unauthorised third parties.

A complete description of Trust and Privacy functionalities can be found in the deliverable D2.2.

3.2.4 Data Management Layer

As presented in the deliverable D2.2, the Data Management Layer is responsible for supporting the data needs of the other layers in the ICOS architecture.

In order to accomplish this objective, Data Management services will be running on both the Controllers and the Agents. The metadata will be stored within the continuum and the Data Management Layer will provide transparent access to the data, abstracting the low-level details of the storage system (e.g., network topology, data structure, storage distribution, transport layers, etc.).

The presence of these Data Management services allows horizontal communication with event-based pub/sub mechanisms, plus support for distributed queries, decoupled from the topology complexities of the continuum. This feature is used in the App Setup Manager (see subsection 3.3.3) where a bus is used for horizontal communication, a feature provided by the Data Management Layer.

The interface of Data Management can be used for storing data and performing task offloading, as shown by the Intelligence Layer (see 3.2.2). This feature of the Data Management Layer allows different intelligence components to interact with data models and perform offloading of AI tasks (as shown in Sequence Diagram in subsection 4.12.1).

Internal Data Management components will leverage network and security components from ICOS to ensure that there is connectivity between Data Management and services and that this communication is secure, such as TLS/mTLS and support for access control, user authentication, and encryption.

3.3 ICOS Agent

The ICOS Agents are responsible for executing the offloaded users' applications, taking care of code execution, data access, telemetry collection and, eventually, runtime communication with other Agents. As shown in Figure 13, the ICOS Agent has been designed including the Onboarding Manager, responsible for onboarding the Agent (and attached infrastructure), the Deployment Manager and the App Setup Manager components, which are responsible for deploying an application, and the Telemetry Gateway, which is responsible for collecting the telemetry and forwarding it to the Controller.

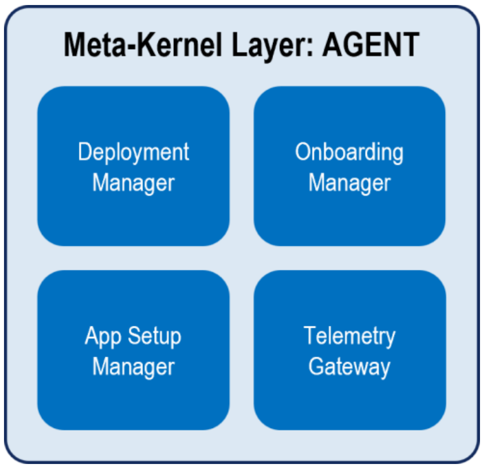


Figure 13. The ICOS Agent Architecture

In the following subsections, the different components of the ICOS Agent are presented and described in detail for each layer: the **Onboarding Manager**, the **Deployment Manager**, the **App Setup Manager**, and the **Telemetry Gateway**.

3.3.1 Onboarding Manager

The Onboarding Manager (OM) is a service that runs on the ICOS Agent. The OM receives and sends requests from the ICOS CLI in the Agent and Infrastructure Services onboarding phases. It is responsible for deployment and lifecycle management of the components running as part of ICOS Agent.

In the ICOS Agent onboarding phase, the "*icos-cli agent join*" CLI provisions the ICOS Onboarding Manager (as Kubernetes Operator) on the ICOS node. The OM contacts the ICOS IAM with a certificate signing request that, when signed and returned, establishes the identity of the ICOS Agent. The OM, after contacting ICOS IAM, creates a local secret with the signed credentials. Using this credential, OM contacts the ICOS Lighthouse to find a suitable ICOS Controller. Once a suitable ICOS Controller is determined, the OM deploys a set of required Agent services to the Kubernetes installation on the ICOS Agent node e.g., Telemetry Gateway and Deployment Manager.

In the Infrastructure services onboarding phase, the OM consumes the infrastructure profile (list of required services) from the ICOS CLI "*icos-cli edge join*". From this profile a manifest is generated that is suitable to the Container Orchestrator Engine running on the infrastructure device. This manifest, containing the list of services and the target, is sent to the Application Deployment API on the ICOS Controller.

3.3.2 Deployment Manager

The Deployment Manager (DM) component represents the so-called driver that provides the abstraction and compatibility between the Job Manager and the underlying orchestration technology of the Agent (any). For this, the DM provides a mapping between the CRUD operations described within the Job Manager and the lifecycle operations it implements. Once the mapping is implemented, the orchestrator becomes compliant with ICOS, thus can be considered an ICOS Agent.

The abstraction enables an ICOS Agent to become capable of executing any job that the Job Manager decides to offload to this Agent. Furthermore, the Deployment Manager implements a scheduler that is constantly fetching jobs for execution. Whenever a job is deemed executable, the Deployment Manager receives it and performs the following three tasks:

- **Request Ownership:** the ownership concept is introduced to prevent other ICOS Agent instances from picking up the same job, thereby preventing race conditions and improving security aspects of ICOS.
- **Execute the Job:** (1) The Agent executes the job by taking the application component description (from the job) and performing the requested operation, i.e., deploy; and (2) the Agent updates the state of the job to the actual state of the resulting resource.
- **Sync with Job Manager**: finally, asynchronously reports any changes on the state of the resources the Deployment Manager is running, until the mentioned resources are removed.

The following Figure 14 displays the current execution flow between the Job Manager and the Deployment Manager as they are tightly related to each other.

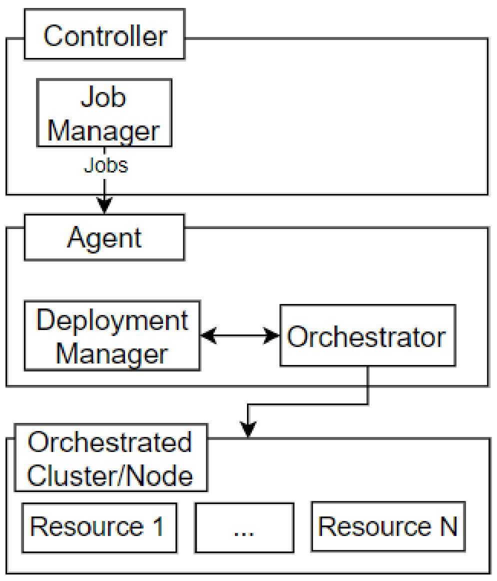


Figure 14. Deployment Manager Component within the ICOS Architecture

3.3.3 App Setup Manager

The App Setup Manager is a crucial component acting as an intermediary layer to facilitate the seamless interaction between applications and the underlying meta-OS. This component ensures that applications can efficiently manage their setup processes and configurations obtaining information from the meta-OS in a standardized and controlled manner.

As depicted in the diagram in Figure 15, the data bus serves as a communication backbone that facilitates the exchange of information between the App Setup Manager and the applications overseen by the ICOS Agent. By leveraging this data bus, the App Setup Manager can send configuration commands and updates to the applications. This interaction is crucial for maintaining up-to-date configurations, as it allows for real-time adjustments. Through the data bus, the App Setup Manager automates the deployment of settings across multiple replicas of the same component simultaneously removing the need for manual configuration. Additionally, the data bus provides a standardized communication protocol, ensuring compatibility and consistency across different applications and systems.

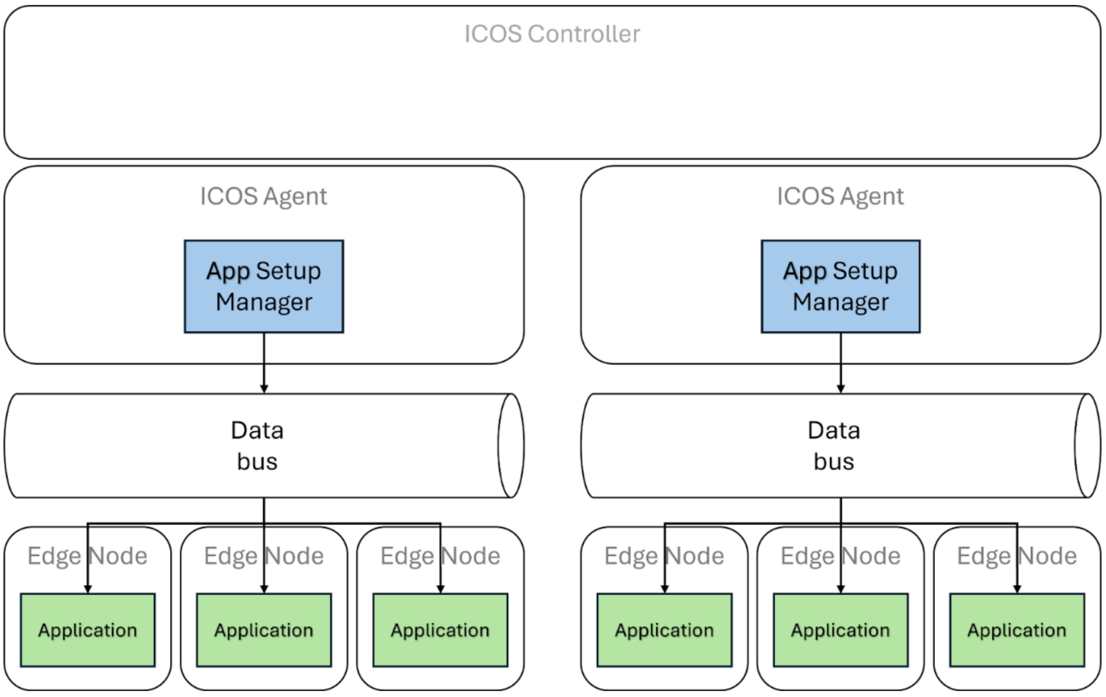


Figure 15. Integration of the App Setup Manager into the ICOS Architecture

The ICOS's data bus is flexible and can run above the Data Link, Network, or Transport Layer of the OSI model, accommodating IP and non-IP networks. The ICOS's data bus has the potential of integrating a wide set of devices using different kinds of networking interfaces like WiFi, Bluetooth, Ethernet, and Serial. This flexibility covers a wide range of common scenarios, especially in robotics, vehicles, and maritime/agricultural/industrial devices, besides the conventional computer network technologies that are commonly used.

3.3.4 Telemetry Gateway

As anticipated in subsection 3.2.1.3, at ICOS Agent level, telemetry data is collected and processed by two sub-modules: the Telemetry Agent and the Telemetry Gateway.

The Telemetry Agent is deployed in the computational resources (e.g., edge devices, VMs) and is responsible for the measurement of the metrics and the reading of the logs directly where they are produced. Thanks to ad-hoc modules and third-party plugins it is able to measure different aspects of the nodes where it is installed:

- resources characteristics, capacity and performance (e.g., CPU, memory, network);
- energy efficiency of the nodes and the applications (e.g., power consumption of nodes and/or processes);
- security of the nodes and the applications (e.g., security assessment results, auditing, security logs).

The Telemetry Agent is highly configurable to activate/deactivate features based on the environment where it is deployed. This aspect is very important since the Telemetry Agent should be able to run all the ICOS devices; also the ones with very constrained resources (e.g., low power, memory or network devices).

Telemetry data is not stored by the Telemetry Agent, instead it is forwarded to a Telemetry Gateway. The Telemetry Gateway is a component deployed in the ICOS Agent that is responsible for:

1. receiving data from all the Telemetry Agent components deployed in the node registered to the ICOS Agent;

2. aggregate, filter and enrich data received (e.g., adding labels to identify the source of the data - like the "agent_id" label);
3. temporarily store (optionally) the data to be used by other ICOS Agent components (e.g., Federated Learning in the Agent);
4. transmit data to the Telemetry Controller in the ICOS Controller.

The Telemetry Gateway component has been introduced in the second iteration of the ICOS Architecture. The main reasons for its introduction are:

- being able to use telemetry data in the ICOS Agent;
- overcome networking issues (e.g., edge devices not being able to reach the Telemetry Controller directly because of firewall or NATting);
- aggregate and optimise the processing of telemetry data (e.g., discarding some metrics, reducing the amount of data sent to the controller).

3.4 ICOS Shell

The ICOS shell consists of three components; the CLI, the GUI and the Shell Backend. These are located at different parts of the system; the Backend lives on the Controller, while the CLI and GUI can be deployed on a user's device. The shell represents the interface for any user to interact with the ICOS system for application maintenance, creation and deletion, as well as representation of users' related metrics, telemetry and security data. The ICOS shell manages the user's identity and authentication by storing an authentication token from the IAM after successful login and attaching it to all successive requests. Depending on the nature of the issued command, these requests are then forwarded to the respective component of ICOS; Meta-Kernel, Intelligence or Security as depicted in Figure 16.

Under the umbrella of the shell, a small set of DevOps tools is provided, which facilitates and automates system actions like device/node onboarding and registration.

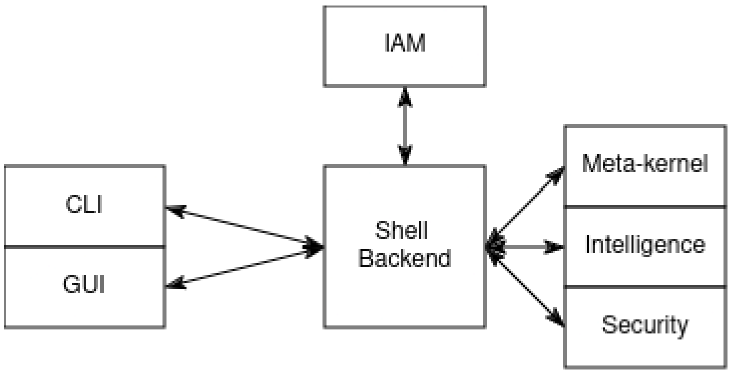


Figure 16. The ICOS Shell

3.4.1 Shell CLI

The ICOS shell is packaged as a single binary to be executed on a user's device. It interacts exclusively with the Shell Backend by using its API. It is written to be easily extensible whenever new functionalities and commands are added to the system.

3.4.2 Shell GUI

The shell GUI, written in react, interacts primarily with the ICOS Shell Backend. It mirrors all features of the ICOS CLI while also extending its functionalities through the introduction of graphical presentation of metric and telemetry data as well as security alarms. While its main purpose is the exclusive interaction with the ICOS Shell Backend, it has the capability of being easily extended to embed either external sites or interacts with other system components.

3.4.3 Shell Backend

The ICOS Shell Backend is the primary entry point for all users to interact with the ICOS system and serves as a proxy for incoming requests. Its API endpoints are generated adhering to the OpenAPI standard (formerly Swagger) through the OpenAPI generator that also generates matching code elements for the ICOS CLI which does not only allow for automatic documentation but also for flexibility.

3.5 ICOS Lighthouse

To allow for communication between controllers, they have to be under the umbrella of the same ICOS Lighthouse. This node is a fixed part of the ICOS system and handles the distribution and registration of controller addresses, serving as the system's controller registry.

The main functionalities of the lighthouse are the following:

- Keeping track of all active ICOS Controllers in the system (joining, leaving, timeout).
- Provide controller address list to nodes joining or rejoining the cluster.
- Provide controller list to all other controllers.
- Provide controller list to the ICOS shell whenever a new session is established.

In summary, the lighthouse has the main role of being the known and fixed contact point for most devices in the ICOS system, always available, to be contacted during the node onboarding stage or in case of loss of connectivity with the system.

4 Architecture: Functional View

This section elaborates on the functional view of the architecture, detailing the Cloud Continuum Operations lifecycle supported by ICOS. The overview of the operation is illustrated in Figure 17. We can distinguish three functional categories. With blue we categorise the CC Operations: On-boarding of i) ICOS Agent; ii) (Edge, IoT, Cloud) Infrastructure; iii) ICOS Controller; and in the case of Multi-Controller, the Multi-Controller Coordination. The orange categorises the operations related to the Deployment of a Vertical Application in the CC using ICOS. The operations categorised as such are Deployment i) Creation; ii) Update; iii) Replace; iv) Stop and v) Delete. Finally, the green colour categorises the operations related to the deployed vertical application execution. The operations under this category are: i) Anomaly Detection Policies; ii) Security and iii) Intelligence. Each one of the above is further broken down to sub-operations. The operation in this category may define the operational state a deployment may be observed at, based on set policies, security incidents and anomalies detected as well as other metrics controlled by the Intelligence Layer (e.g., energy consumption etc).

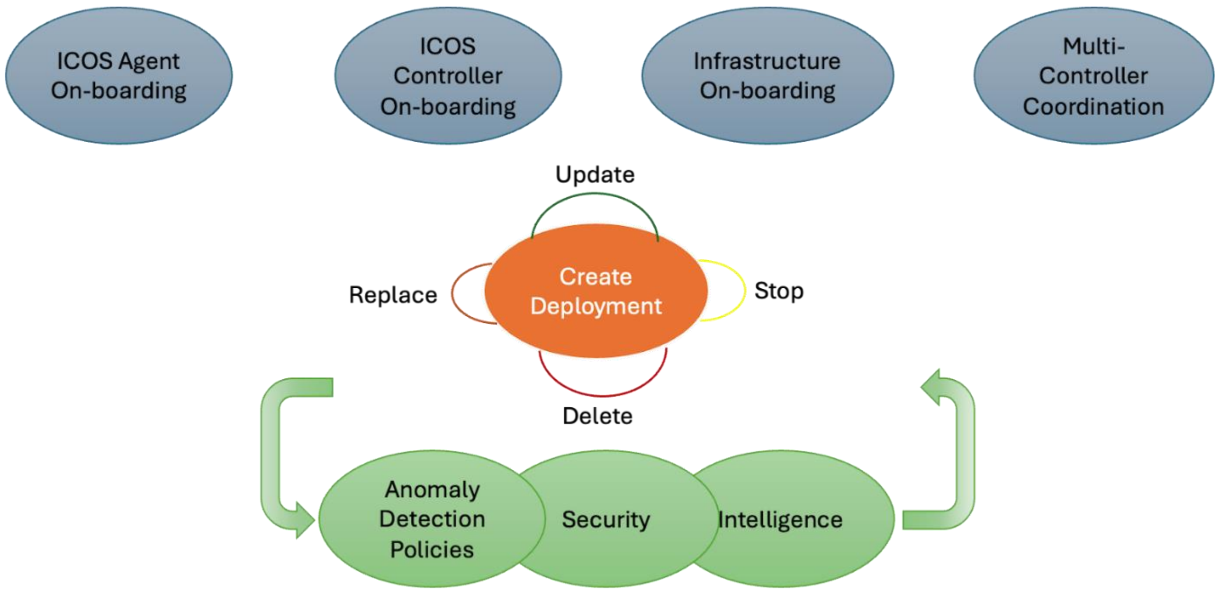

ICOS Agent On-boarding
ICOS Controller On-boarding
Infrastructure On-boarding
Multi-Controller Coordination
Update
Replace
Create Deployment
Stop
Delete
Anomaly Detection Policies
Security
Intelligence

Figure 17. ICOS Functional View

4.1 Onboarding: ICOS Agent

The onboarding of an ICOS Agent must be done by an ICOS Agent Operator persona. An ICOS Agent needs to be added or "onboarded" to an ICOS Controller within an organisation's vertical deployment. Before an ICOS Agent can be added some steps must be completed.

The owner of the ICOS Controller adds the Controller details to the ICOS Lighthouse. See the process described in the “Onboarding: ICOS Controller” section above.

The ICOS Agent operator needs to obtain credentials from: the Orchestrator of the ICOS node; the Kubernetes credentials from the installation on the ICOS Agent node; request a join token from the ICOS IAM. These credentials are passed as a configuration to the ICOS command line that will launch the "ICOS agent join" action.

The "ICOS agent join" contacts the ICOS Onboarding Operator (OO) on the ICOS node. The ICOS OO contacts the ICOS IAM with a certificate signing request that, when signed and returned, establishes the identity of the ICOS Agent. The ICOS OO, after contacting ICOS IAM, creates a local secret with the signed credentials. Using this credential, the ICOS OO contacts the ICOS Lighthouse to find a suitable ICOS Controller. Once a suitable ICOS Controller is determined, the ICOS OO deploys a set of required Agent services to the Kubernetes installation on the ICOS Agent node e.g., telemetry gateway, Deployment Manager.

The ICOS Agent services that are required to contact the ICOS Controller use the local secret to contact the ICOS IAM to obtain a short-lived token. The ICOS Deployment Manager contacts the Orchestrator, using the Orchestrator credentials, to begin accepting workloads.

ICOS Agent onboarding sequence diagram is presented in Figure 18.

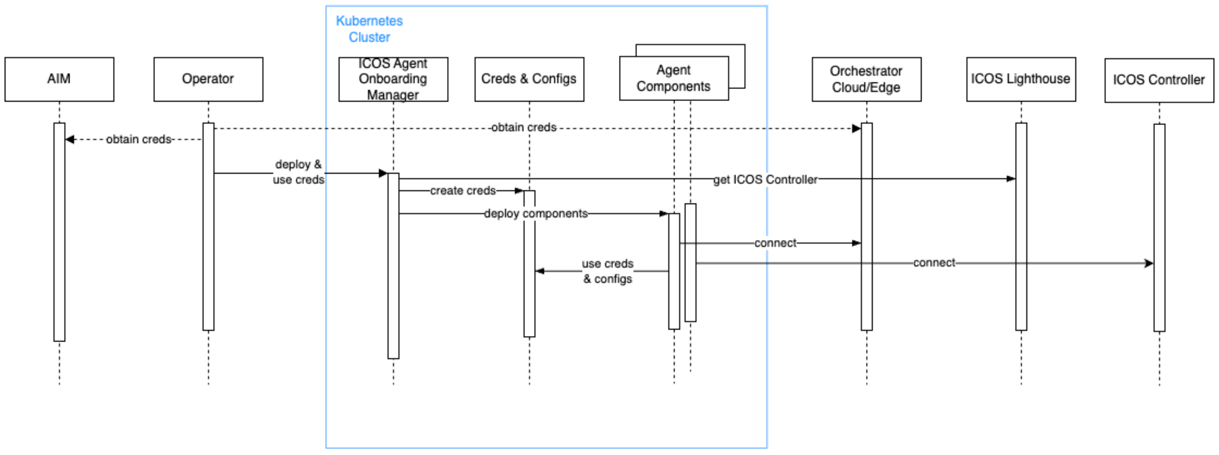


Figure 18. ICOS Agent onboarding sequence diagram

4.2 Onboarding: Infrastructure

From the ICOS point of view, Infrastructure is where the workloads of ICOS users will run.

An Infrastructure Device (InfDev) could be an Edge/IoT device, physical server or virtual machine in a Cloud infrastructure. The InfDev must be connected to the local Orchestrator (e.g., OCM, Nuvla or other) and the ICOS Agent that communicates with the Orchestrator.

The InfDev is owned by an Operator that will perform the onboarding.

As a prerequisite, the Operator must possess valid credentials for: the Orchestrator, local container orchestrator engine on the InfDev (e.g., Kubernetes or Docker); ICOS Agent credentials (obtained from the ICOS IAM). This set of credentials are required for contacting the ICOS Controller, Orchestrator and container orchestration engines.

The Operator must also define a profile that defines the extra services needed beyond the essential services that will run on the InfDev. The Operator runs "icos-cli edge join" from the command-line on the InfDev which then follows a sequence.

An entity for an Orchestrator Agent is created on the Orchestrator (e.g., NuvlaEdge). The Orchestrator Agent is deployed to the InfDev and commissioned to the Orchestrator. At this point, the InfDev can be joined to the ICOS Agent through the ICOS Onboarding Manager (OM). The InfDev profile is passed to the OO.

The OM deploys the InfDev services as a manifest to the controller application deployment API. At this point the services to be deployed and the target endpoint (InfDev) are known. The Agent Deployment Manager pulls the job from the Controller application deployment API. At this point, the standard ICOS deployment of workloads is followed and the standard (e.g., telemetry) and optional (e.g., ClusterLink) services are deployed to the InfDev. The InfDev is now able to send telemetry and receive workloads from the ICOS Agent.

Infrastructure onboarding sequence diagram is presented in Figure 19.

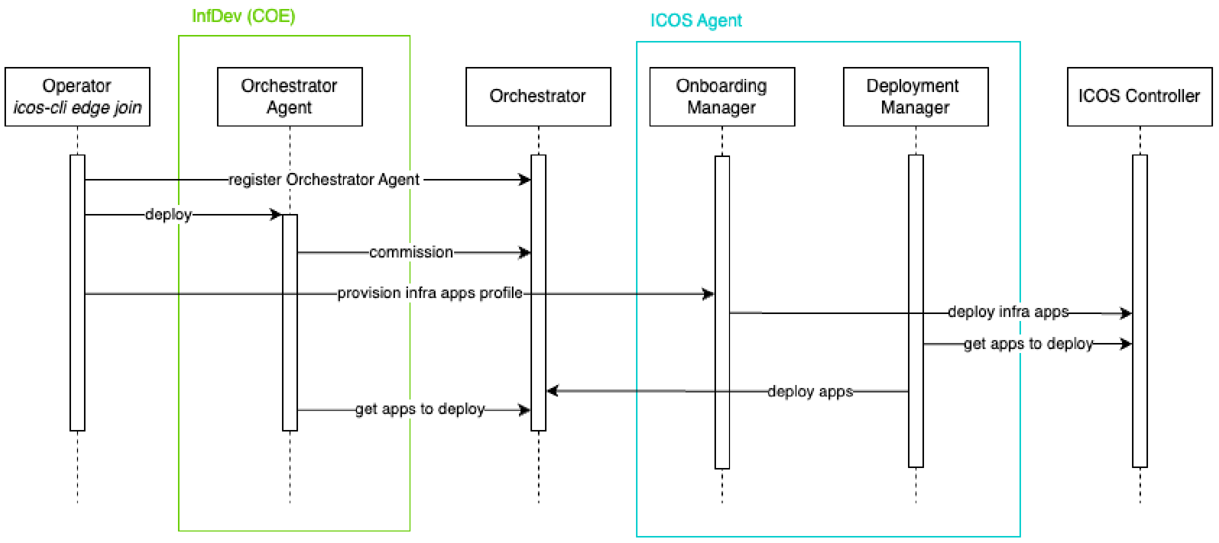


Figure 19. Infrastructure onboarding sequence diagram

4.3 Onboarding: ICOS Controller

An organisation using ICOS creates an "ICOS vertical" that includes: IAM Lighthouse, Controller, Agents, and Infrastructure devices. The ICOS Controller sits at the top of the vertical and, therefore, needs to be bootstrapped from scratch. The ICOS Controller suite of services is installed via Helm from the ICOS repository by the ICOS Controller owner.

The ICOS Controller owner adds the Controller details to the ICOS Lighthouse, including information such as the IP address, endpoints, capabilities, and indications of ICOS Agents eligible to join the Controller.

The credentials to identify the ICOS Controller are generated by the vertical owner using their local IAM. Trust between the ICOS Controller and IAM (part of the vertical) is assumed since, when an ICOS Agent is onboarded, it will use a credential from the ICOS IAM. The ICOS Controller credentials and ICOS Agent credentials are used to set up mTLS for subsequent communications.

ICOS Controller onboarding sequence diagram is presented in Figure 20.

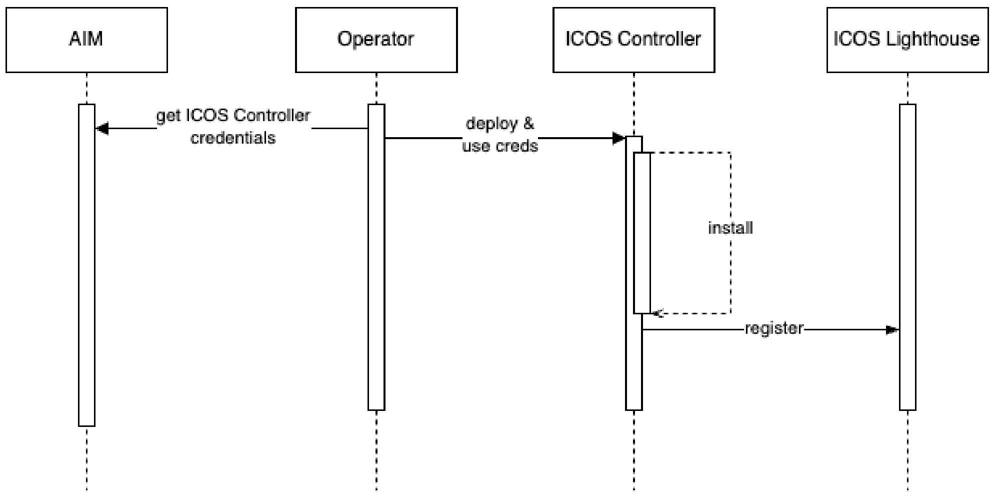


Figure 20. ICOS Controller onboarding sequence diagram

4.4 Create Deployment

When an operator requests a new application deployment, this deployment is created by following the application description provided. The Deployment Manager creates the group of resources to be applied to the managed cluster. This involves creating a resource manifest that contains the necessary resource definitions in a format that can be interpreted by the Orchestrator. Next, the Deployment Manager submits this definition to the orchestrator's API, which applies the workload and returns metadata, which is used to track the resource status. Finally, the metadata is sent to the Job Manager to be persisted in its database.

Create Deployment sequence diagram is presented in Figure 21.

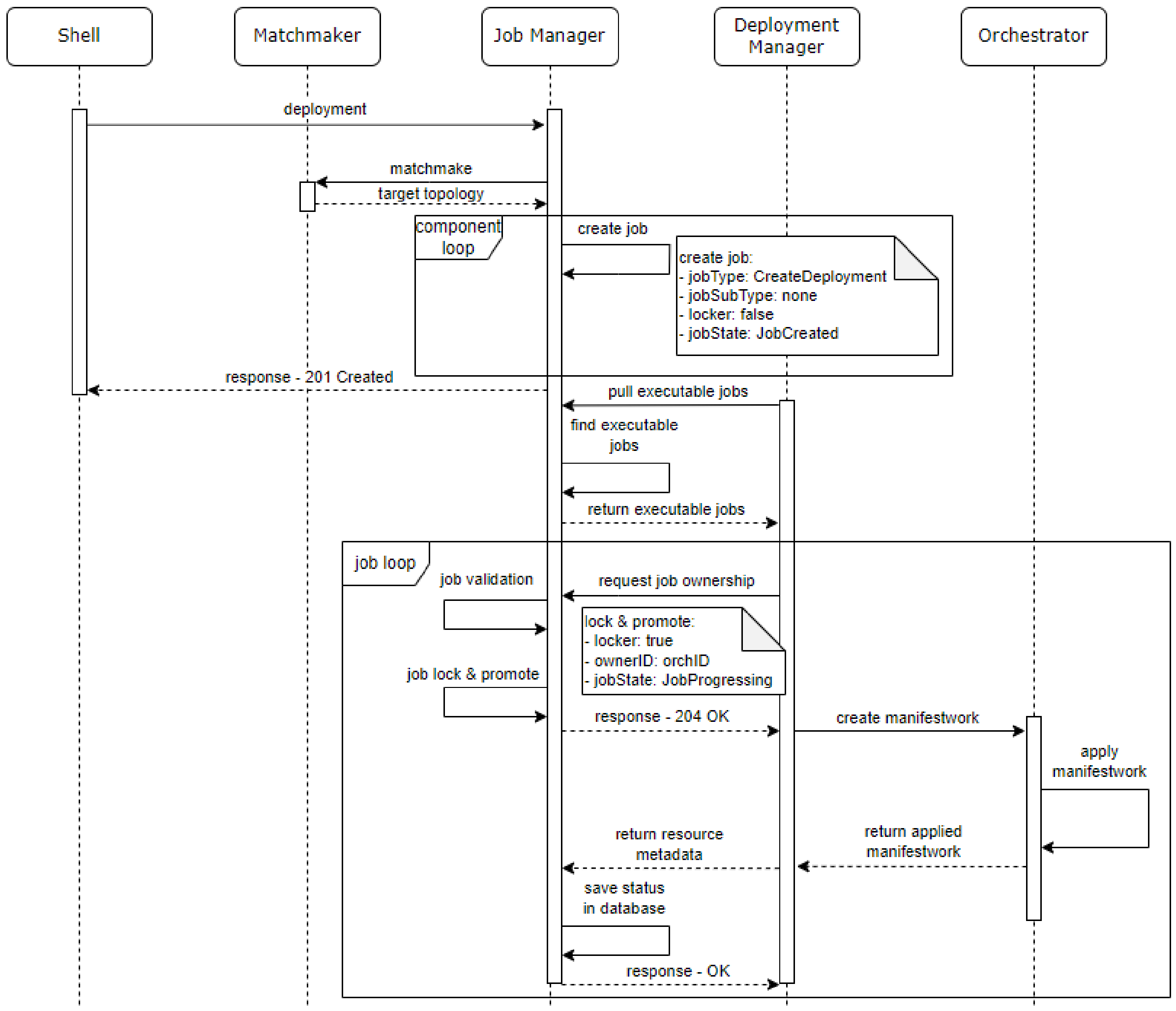


Figure 21. Create Deployment sequence diagram

4.5 Update Deployment

Updating a deployment involves adjusting resource requirements or the number of replicas to meet specific criteria set by the policy manager. When the policy manager detects an incompliance, it informs the Job Manager of the necessary actions to be taken. The Deployment Manager currently supports the following actions:

- **Scale-out**: adds N replicas to a deployment.
- **Scale-in**: removes N replicas from a deployment.

- **Scale-up**: adds more resources (CPU/RAM) to a deployment.
- **Scale-down**: removes resources (CPU/RAM) from a deployment.
- **Reallocation**: moves a stateless application component to a different ICOS Agent by removing the current workload and creating a new deployment that will follow the create deployment sequence described in section 4.4.

Figure 22 presents Update Deployment Scaling sequence diagram, whereas Figure 23 shows Update Deployment Reallocation sequence diagram.

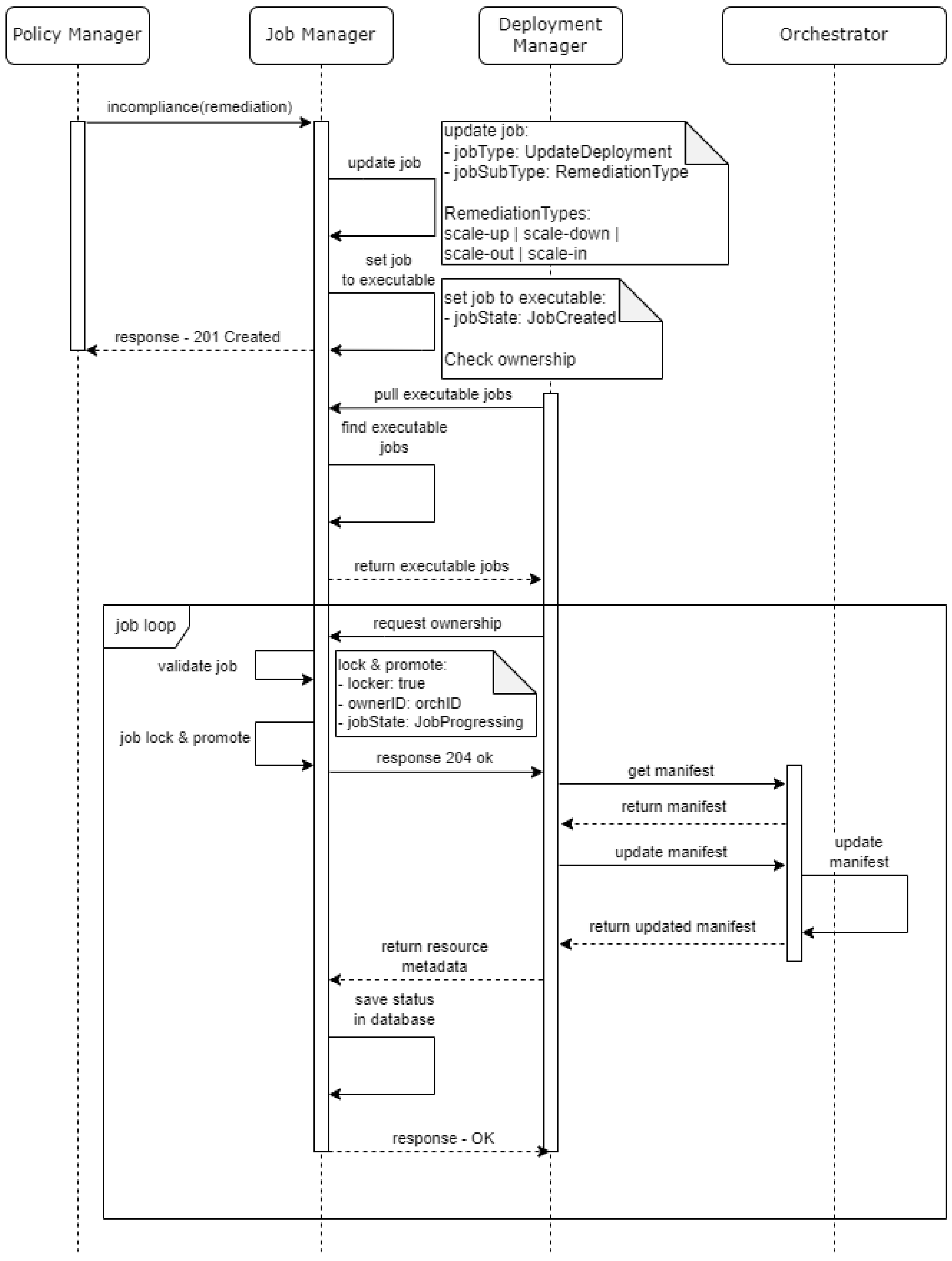


Figure 22. Update Deployment Scaling sequence diagram

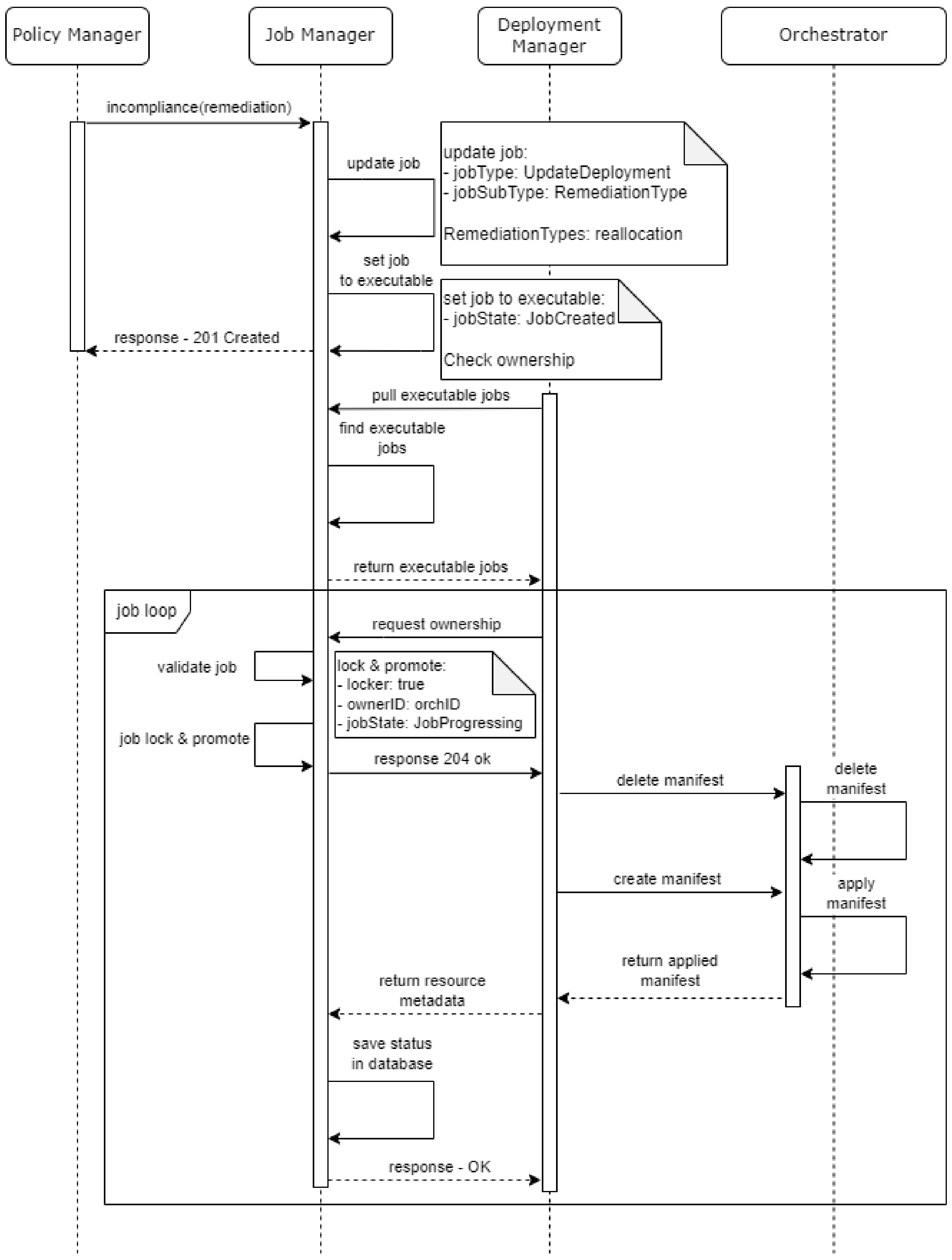


Figure 23 . Update Deployment Reallocation sequence diagram

4.6 Replace Deployment

Replacing an existing deployment involves updating the deployment with new parameters provided by the operator. The operator submits a request containing the ID of the job group to be replaced and can modify various parameters, such as the application name and job manifests. Once the replacement request is validated, the job manager makes the job executable. Instead of creating a new resource, the Deployment Manager updates the existing one with the new parameters. This process ensures that the updated configurations are applied correctly, maintaining continuity and consistency in the deployment management workflow. The updated metadata is then sent to the job manager for database persistence. Figure 24 shows the Deployment Replacement sequence diagram.

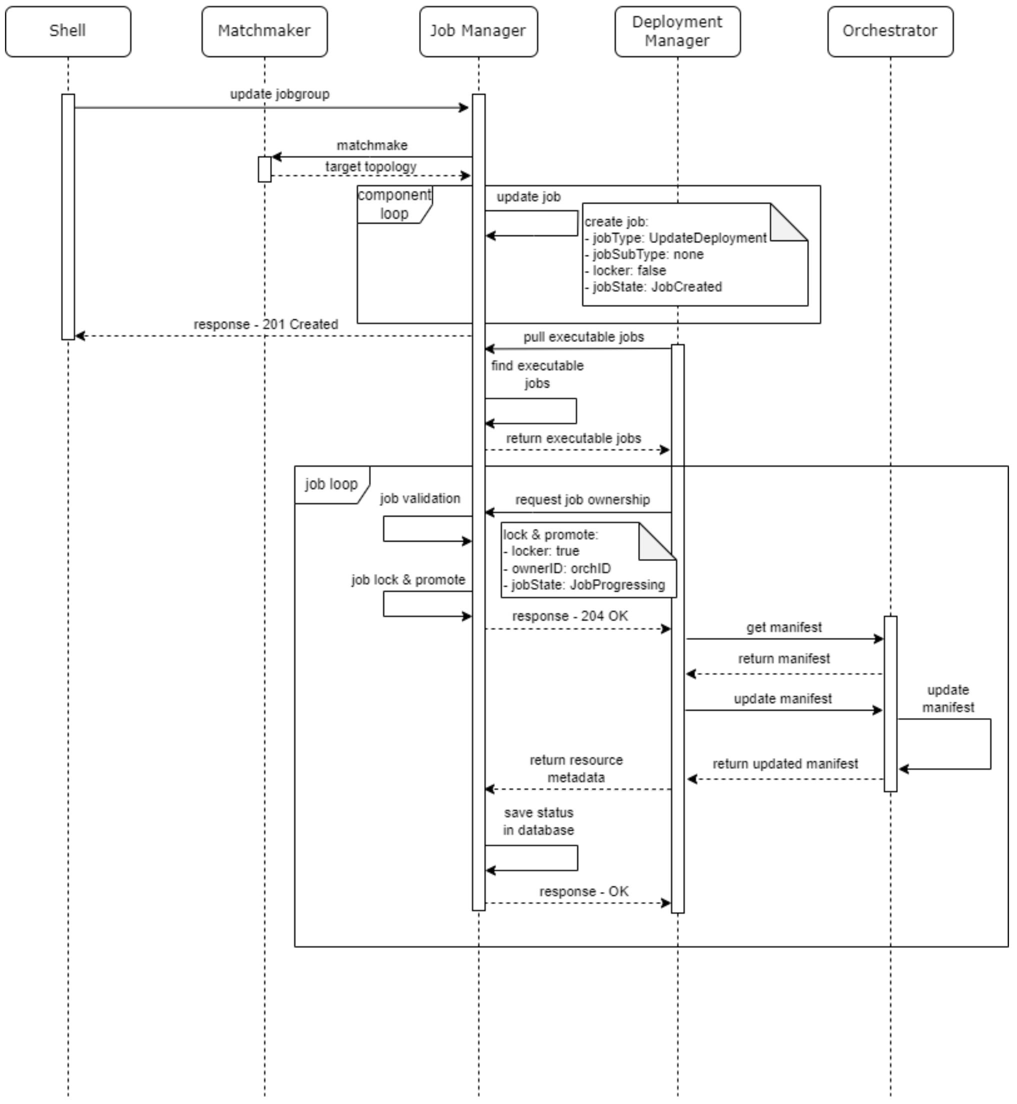


Figure 24. Deployment Replacement sequence diagram

4.7 Stop Deployment

This process refers to the action of deleting a workload that is currently running in a managed cluster. This action is typically necessary when an application deployment no longer suits the needs of an operator, it requires a restart or it must be deployed on a different cluster. By stopping a deployment, all resources associated with the workload are freed, ensuring efficient use of cluster resources and maintaining the desired state of the managed environment. Figure 25 shows the Stop Deployment sequence diagram.

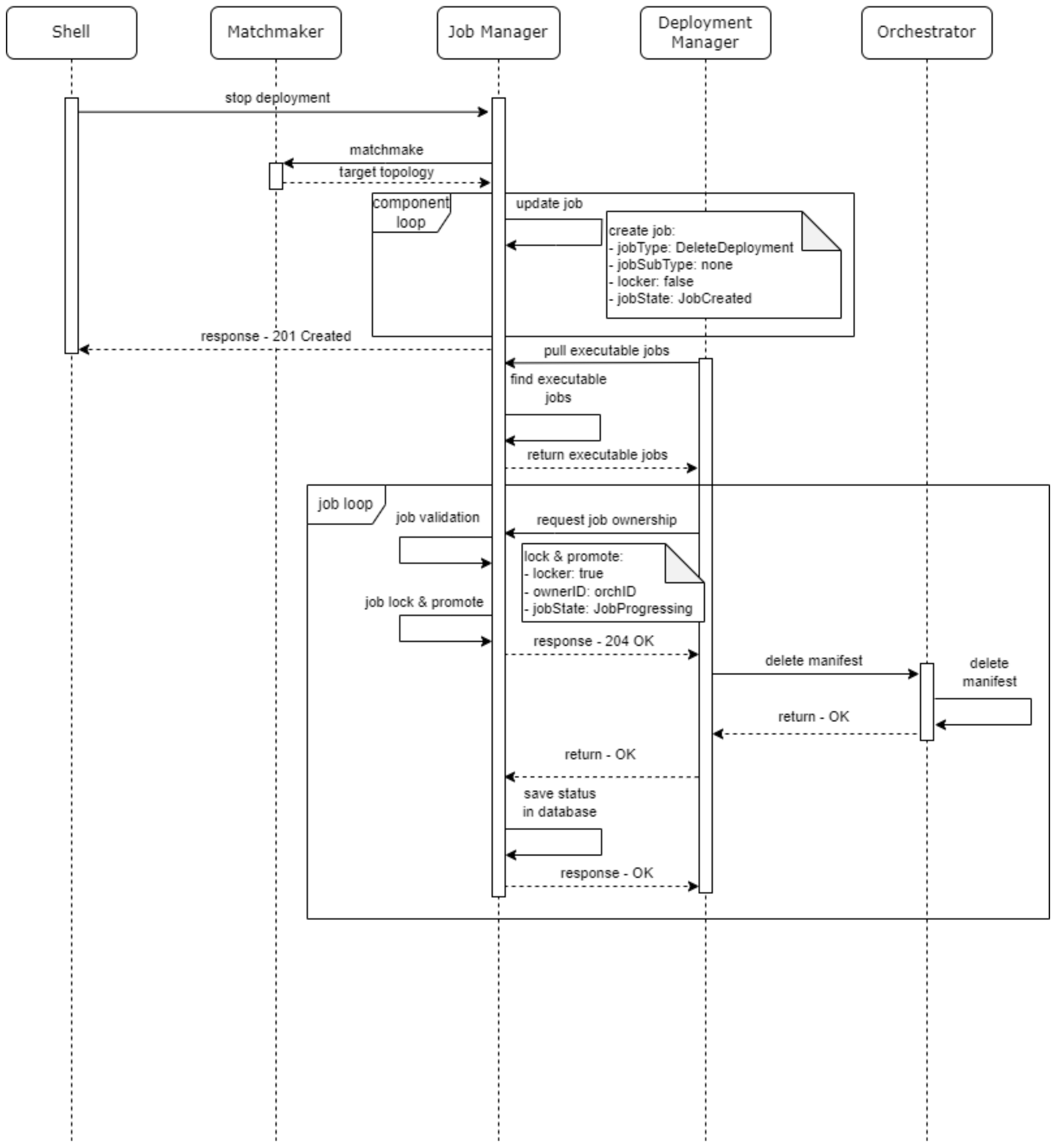


Figure 25. Stop Deployment sequence diagram

4.8 Delete Deployment

Deleting a deployment involves removing the resource from the orchestrator and resetting the job that was managing the workload to its default state. This process is similar to reallocation but focuses on completely removing the resource rather than moving it. Once the deployment is deleted, the job managing the workload is set to a "job created" state, ensuring it is ready for future assignments. Figure 26 shows the Delete Deployment sequence diagram.

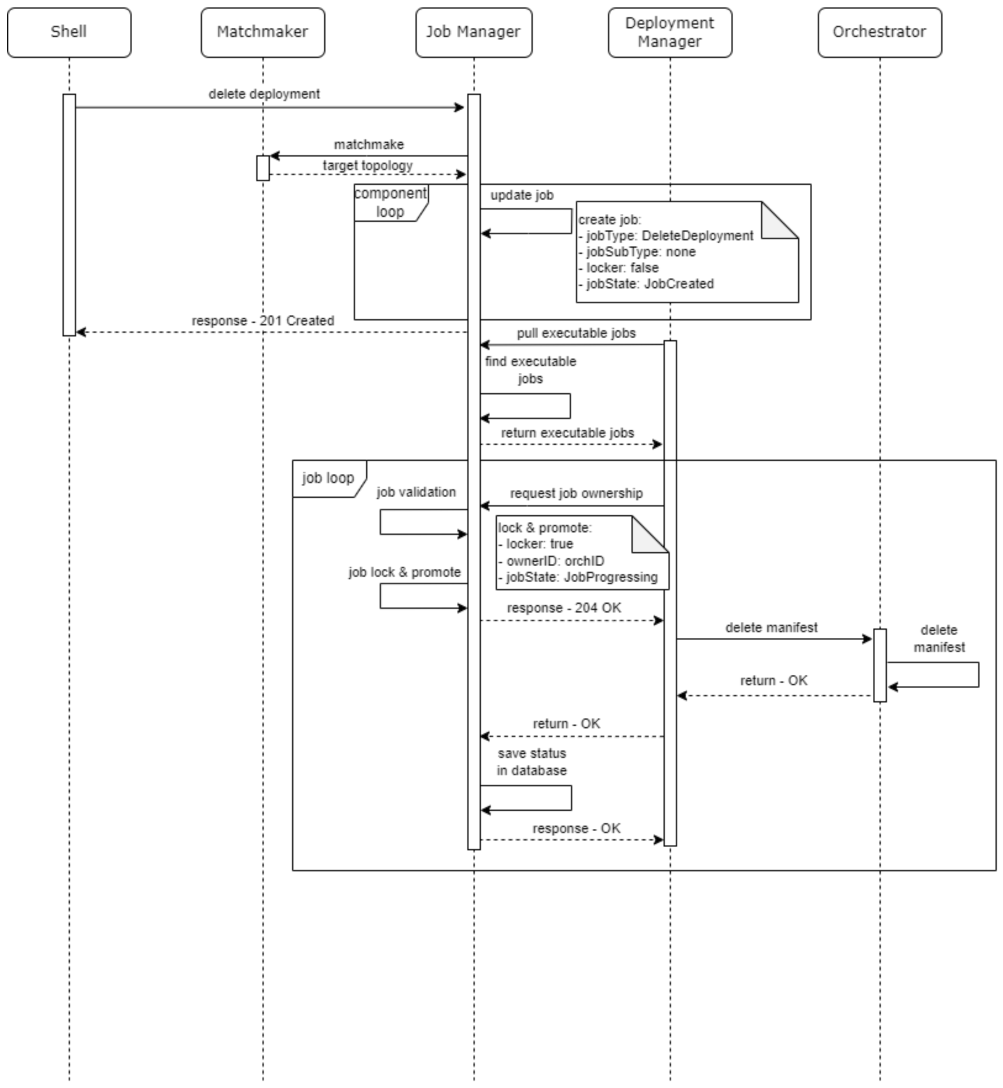


Figure 26. Delete Deployment sequence diagram

4.9 Anomaly Detection Policies

The enforcement of technical and performance policies at runtime can be decomposed in three steps depicted in the diagram in Figure 27.

The first step is the Set-Up of Policies. When a new application deployment is submitted to the system, the Job Manager, after the creation of deployment jobs, calls the Policy Manager passing the Application Descriptor (where also policies to enforce for the application are defined). The Policy Manager translates the policies in measurable expressions based on metrics (e.g., "cpu_usage < 0.8"). Then, it configures the Telemetry module to send alerts when these expressions are violated.

During the application runtime, the Telemetry module starts receiving telemetry data from the ICOS nodes where the application components are deployed and evaluate the expressions configured by the Policy Manager.

If any of the configured expressions fails, the Telemetry module sends an alert to the Policy Manager. At this point the Policy Manager computes a policy violation based on the policy context stored in its backend and sends a notification of "Policy Violation" to the Job Manager. The Job Manager performs an application update (Update deployment, see diagram in section 4.5) based on the type of violation received.

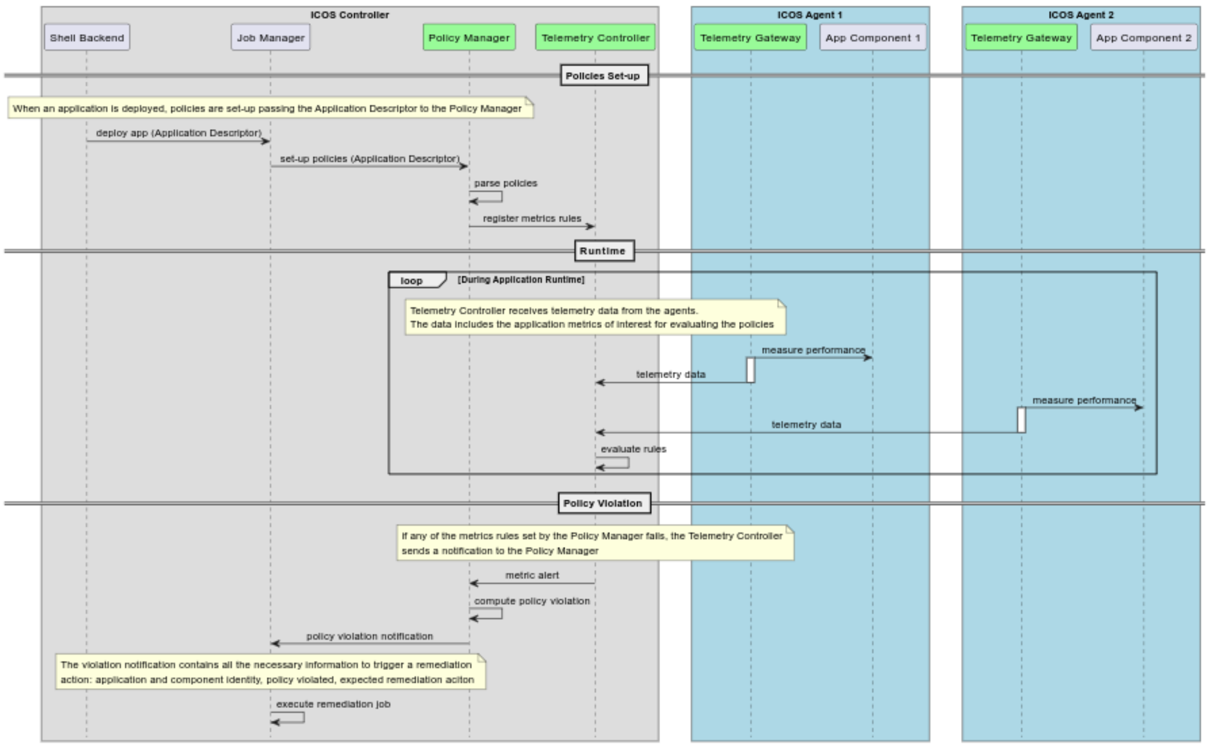

ICOS Controller
Shell Backend
Job Manager
Policy Manager
Telemetry Controller
ICOS Agent 1
Telemetry Gateway
App Component 1
ICOS Agent 2
Telemetry Gateway
App Component 2
Policies Set-up
When an application is deployed, policies are set-up passing the Application Descriptor to the Policy Manager
deploy app (Application Descriptor)
set-up policies (Application Descriptor)
parse policies
register metrics rules
Runtime
loop
[During Application Runtime]
Telemetry Controller receives telemetry data from the agents.
The data includes the application metrics of interest for evaluating the policies
measure performance
telemetry data
measure performance
telemetry data
evaluate rules
Policy Violation
If any of the metrics rules set by the Policy Manager fails, the Telemetry Controller sends a notification to the Policy Manager
metric alert
compute policy violation
policy violation notification
The violation notification contains all the necessary information to trigger a remediation action: application and component identity, policy violated, expected remediation action
execute remediation job

Figure 27. Anomaly detection policies

4.10 Multi-controller coordination

As described in the System Overview (section 3.1) ICOS has been defined as a multi-controller meta-OS. Each ICOS Controller is in charge of managing and coordinating the tasks (continuum and runtime) in a specific scope, and when an application execution request cannot be satisfied within the bounds of the corresponding Controller scope, then a multi-controller coordination is required. Figure 28 shows the multi-controller sequence diagram.

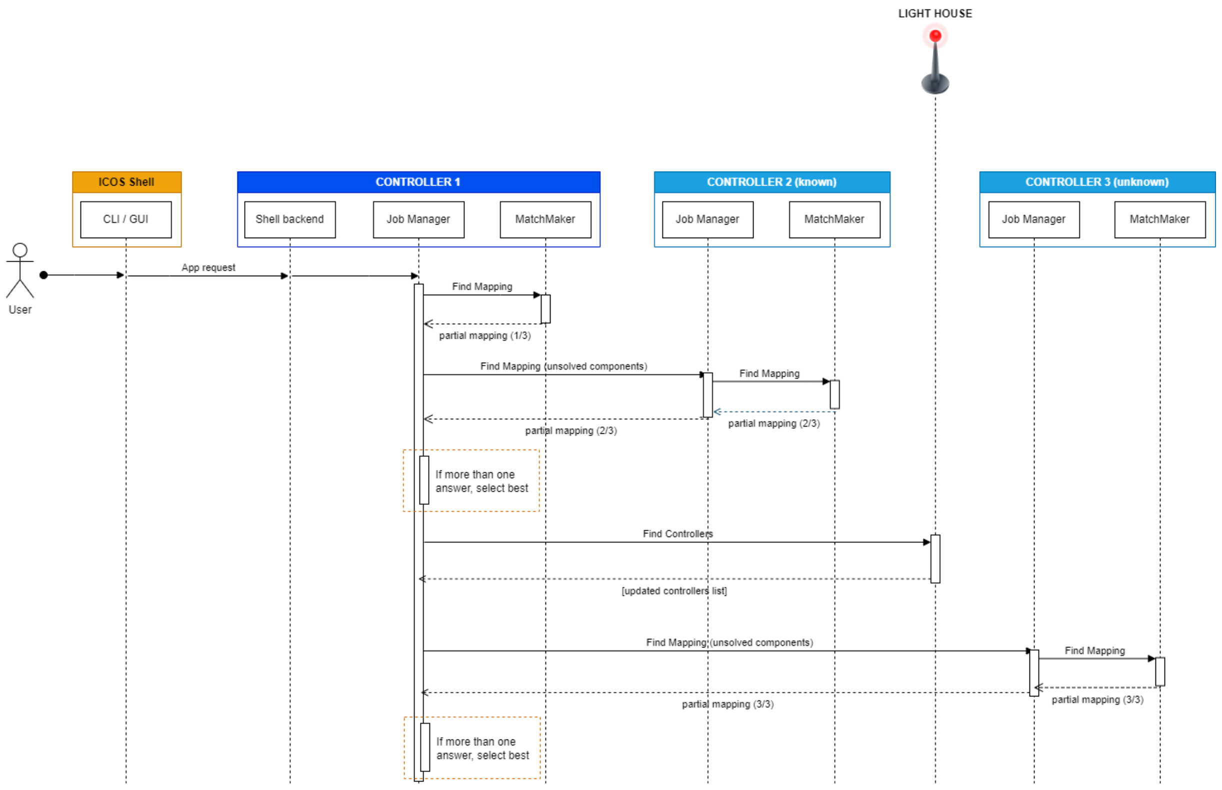


Figure 28. Multi-controller coordination sequence diagram.

The application is launched by a user (either manually or automatically) through the ICOS Shell. The request is received through the Shell Backend and moved to the Job Manager (from a specific Controller), as already described in the Create Deployment sequence diagram (section 4.4).

The Job Manager (JM) invokes the MatchMaker (MM) to find the best application matching according to the application requirements and constraints. Assume that only some application components could be mapped within the scope of the current Controller (for instance, some components need specialised hardware that is not in the infrastructure managed by the Controller), then the MM will only provide a partial mapping. The JM will then forward the unsolved application components to the list of known Controllers (those that have been contacted previously) that will try to match the remaining components. If some of them can satisfy the request, then the mapping is done. However, if there are yet some components that couldn't be mapped, then the Controller will check with the ICOS Lighthouse if there are new Controllers (controllers that haven't been contacted before), and will try again with them. Finally, if all application components could be mapped, then the application can be executed across Controllers; however, if any component couldn't be mapped, then the application cannot be executed.

4.11 Security Sequence Diagrams

Sequence diagrams in this section present three security workflows:

1. How anomalies are detected in ICOS using AI log monitoring (modified from the original presentation in D2.2)
2. How security issues are detected on ICOS nodes using a rule-based approach and how remediation actions on the ICOS nodes are applied
3. How security issues are detected on ICOS Agents and how remediation actions on ICOS Agents are applied

Other sequence diagrams regarding security (e.g., Identity and Access Management) can be found in D2.2.

4.11.1 Anomaly Detection

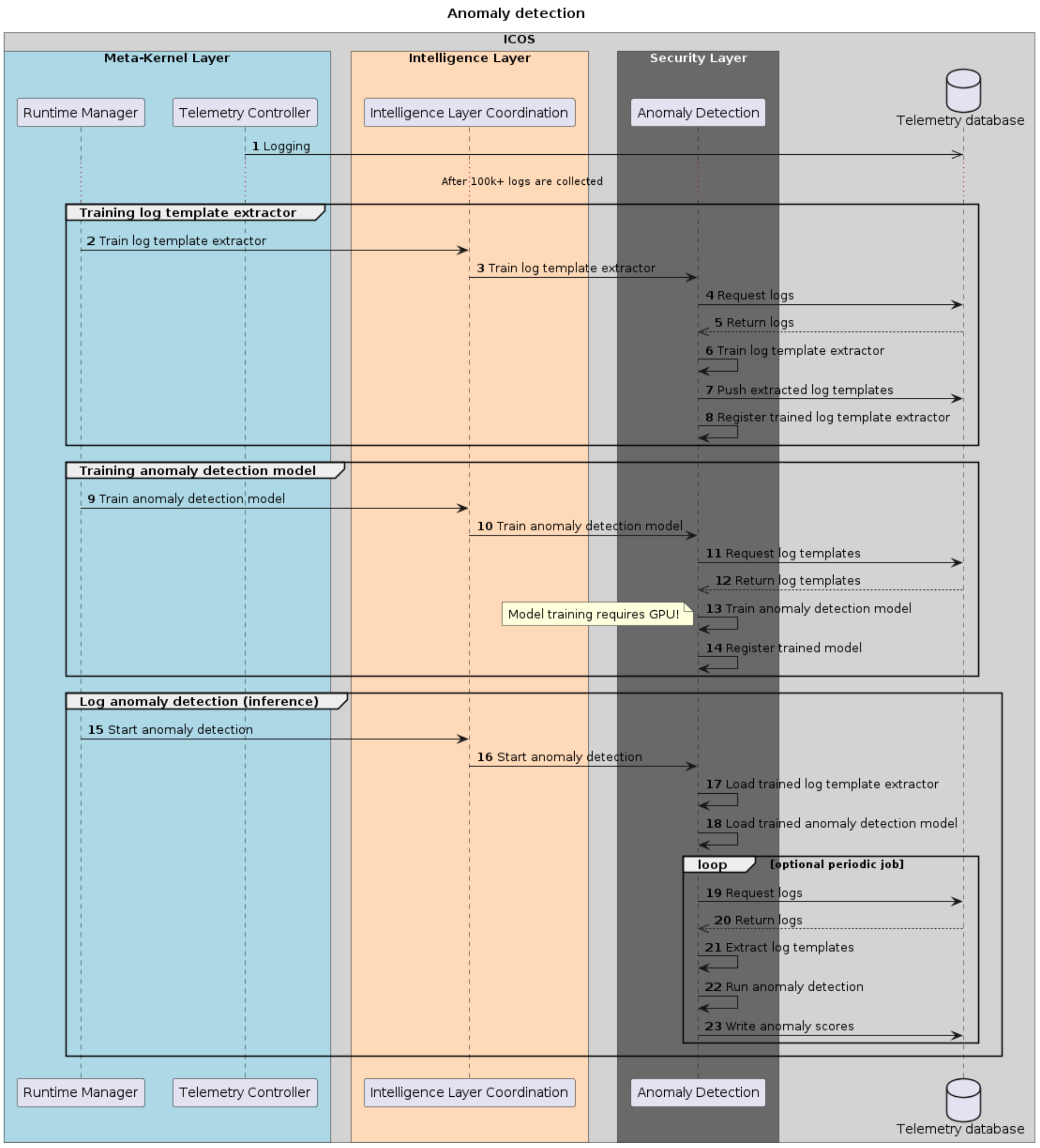
As it is described in deliverable D2.2 [1], the Anomaly Detection works in three main phases:

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

Figure 29 shows the anomaly detection sequence diagram. 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.



Anomaly detection
ICOS
Meta-Kernel Layer
Intelligence Layer
Security Layer
Runtime Manager
Telemetry Controller
Intelligence Layer Coordination
Anomaly Detection
Telemetry 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

Figure 29. Sequence diagram for anomaly detection

4.11.2 Detection of security issues on the ICOS nodes and remediation actions

Figure 30 shows the detection of security issues on the ICOS nodes and remediation actions sequence diagram. The workflow presents starting the Security Scan module to scan for security issues (e.g., vulnerabilities, file integrity, compliance checks…) on the ICOS node (baremetal, VM…). Security Scan is deployed using Agents on the ICOS Nodes to scan the node and Server to get the scan results. The Security Coordination module calls the Server to combine the results of the scans into security scan metrics, which are then visualised in the Grafana for the ICOS user (e.g., infrastructure provider). The latter can inspect the metrics, and decide on and apply the appropriate remediation action on the ICOS node (patching, updating…).

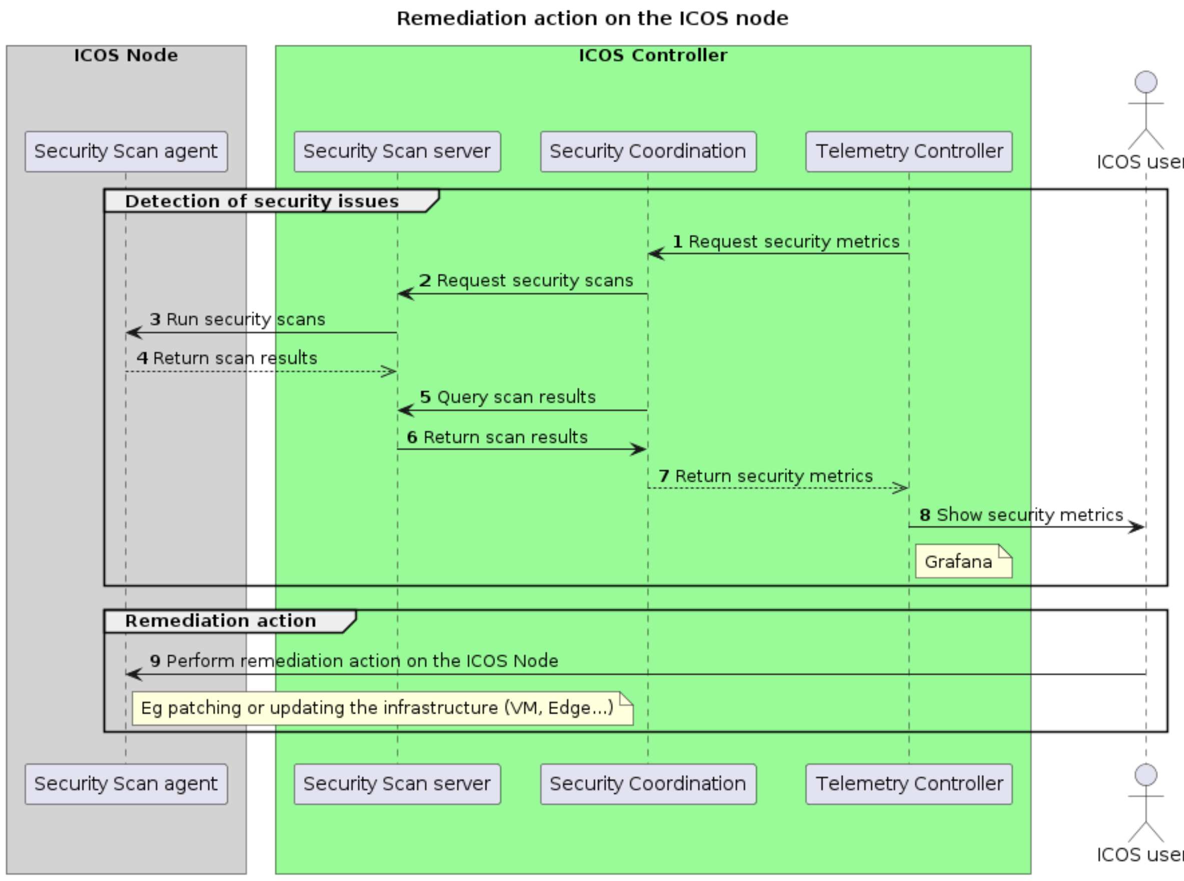


Figure 30. Sequence diagram for detection of security issues on the ICOS nodes and remediation actions

4.11.3 Detection of security issues on the ICOS Agent and remediation actions

Figure 31 shows the detection of security issues on the ICOS Agent and remediation actions sequence diagram. The workflow involves all three ICOS layers: Meta-Kernel, Security, and Intelligence.

Detection of security issues is done in runtime with the Audit module. The module deploys agents to monitor the kernel events on the Kubernetes cluster on which the ICOS Agent is running. The agents push the event logs to the Audit Server, which processes these logs and pushes the processed logs and metrics to the Telemetry controller. After security issues are detected, two types of remediation action are possible: rule-based and intelligence-based.

a) Rule-based remediation action includes security policies with defined remediation actions. The Policy Manager translates the policies into measurable expressions based on metrics and configures the Telemetry Controller to send alerts when these expressions are violated.

b) When the Telemetry Controller detects failure of audit metrics, it sends an alert to the Policy Manager. At this point, the Policy Manager computes a policy violation based on the policy context stored in its backend and sends a notification of "Policy Violation" to the Job Manager. The latter computes a remediation action based on the received notification and applies the action on the Kubernetes cluster using the Deployment manager and selected orchestrator (OCM or Nuvla).
Intelligence-based remediation action using predictions of security metrics. This workflow relies on the training and inference of prediction models by the Intelligence Layer. The sequence of training the models is similar to the sequence of **AI training, inference, and re-training,** as presented in the Intelligence sequence diagrams section. In short, the Policy Manager calls the Intelligence Coordination API. Once received, the AI layer searches Telemetry for audit data

and prepares it for model training. Once the model training is completed, the trained model file and the model metrics are stored at the Intelligence coordination API, ready to be used for inference.
In inference, the Policy Manager calls Intelligence Coordination API to run the model and provide predictions. The Intelligence Coordination API obtains the latest audit data from the Telemetry Controller, runs the model with this data, and provides predictions to the Policy Manager. From this point onward, a workflow similar to the rule-based remediation action is executed. The Policy Manager uses the prediction data to compute the policy violation and notifies the Job Manager, which computes and applies the action on the K8s cluster using the Deployment manager and selected orchestrator (OCM or Nuvla).

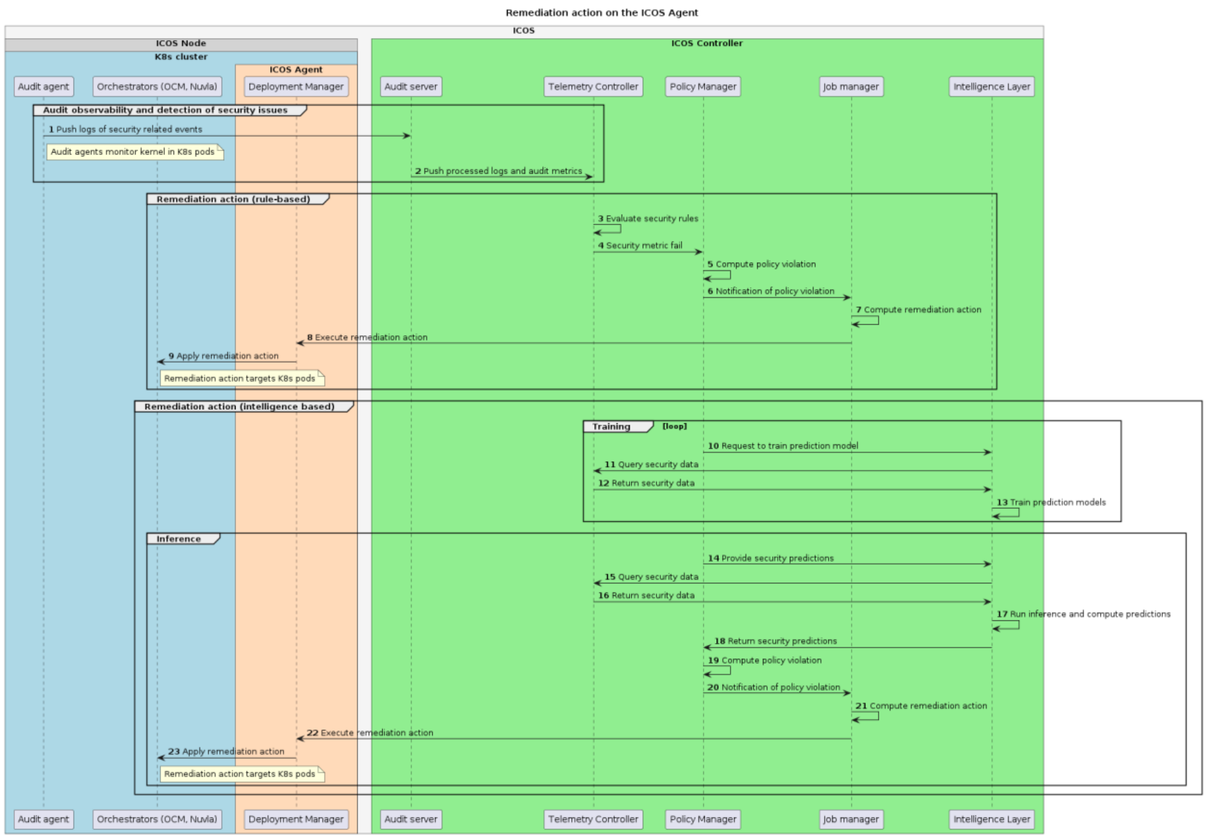

Remediation action on the ICOS Agent
ICOS
ICOS Node
K8s cluster
ICOS Agent
ICOS Controller
Audit agent
Orchestrators (OCM, Nuvla)
Deployment Manager
Audit server
Telemetry Controller
Policy Manager
Job manager
Intelligence Layer
Audit observability and detection of security issues
1 Push logs of security related events
Audit agents monitor kernel in K8s pods
2 Push processed logs and audit metrics
Remediation action (rule-based)
3 Evaluate security rules
4 Security metric fail
5 Compute policy violation
6 Notification of policy violation
7 Compute remediation action
8 Execute remediation action
9 Apply remediation action
Remediation action targets K8s pods
Remediation action (intelligence based)
Training
[loop]
10 Request to train prediction model
11 Query security data
12 Return security data
13 Train prediction models
Inference
14 Provide security predictions
15 Query security data
16 Return security data
17 Run inference and compute predictions
18 Return security predictions
19 Compute policy violation
20 Notification of policy violation
21 Compute remediation action
22 Execute remediation action
23 Apply remediation action
Remediation action targets K8s pods

Figure 31. Sequence diagram for detection of security issues on the ICOS nodes and remediation action

4.12 Intelligence sequence diagrams

This section contains sequence diagrams related to intelligence, which are used to train, infer, and retrain models collaboratively in a Federated Learning fashion. Sequence diagrams are presented in the following two subsections.

4.12.1 AI training, inference, and re-training sequence diagram

This section shows the interactions followed in ICOS:

- to train an AI model for a new metric in MetaOS and then offload that request to the Data Management layer,
- for model inference, and
- for model re-training.

The action to predict a new metric might be a request from the Meta-Kernel layer, such as the Policy Manager, via an API call to the Intelligence coordination API. Once received, the AI layer searches the Telemetry Controller for data on the new measure and prepares it for model training. The model training request is subsequently routed to the Data Management layer. The Intelligence coordination API initiates this request to offload. Once the model training is completed, we receive the trained model file and the model metrics at the Intelligence coordination API, where the trained model and metrics are stored in a repository and ready to be used for inference. Furthermore, the trained model may require retraining due to performance degradation over time, and this re-training is started by the AI layer, which queries telemetry data and uses the most recent data to improve model performance. Finally, upon request, the model inference generates prediction results and sends them back to the Telemetry controller at a set frequency. This continues until it is alerted to stop.

Figure 32 depicts the sequence diagram initiating AI model training offload and model inferencing for a new metric.

We will now look at the steps involved in this approach.

1. **Request to train a new model:** This process begins with the Meta-Kernel layer making a request to the Intelligence layer to predict a new metric in the ICOS system, which can, for instance, be the policy manager.
2. **Data to train the model**: The Intelligence layer then queries the data in the Telemetry controller to train new metrics and does the necessary data preprocessing.
3. **Offload model training:** The Intelligence coordination API offloads the model training request to the Data management layer via Dataclay, which could be either the same or a different device, depending on the architecture.
4. **Saving the trained model:** The trained model and its metrics are then returned to the Intelligence layer, where the model is saved in the repository.
5. **Deploying the model:** The trained model is then deployed and ready for production to generate model predictions.
6. **Model re-training:** This step involves the Intelligence layer querying new data from Telemetry and re-training the model to improve its performance.
7. **Model Inference:** The Meta-Kernel layer requests the Intelligence layer to generate model predictions. The Intelligence layer then periodically uploads model predictions in the Telemetry Controller, and this continues in a loop until notified to stop.

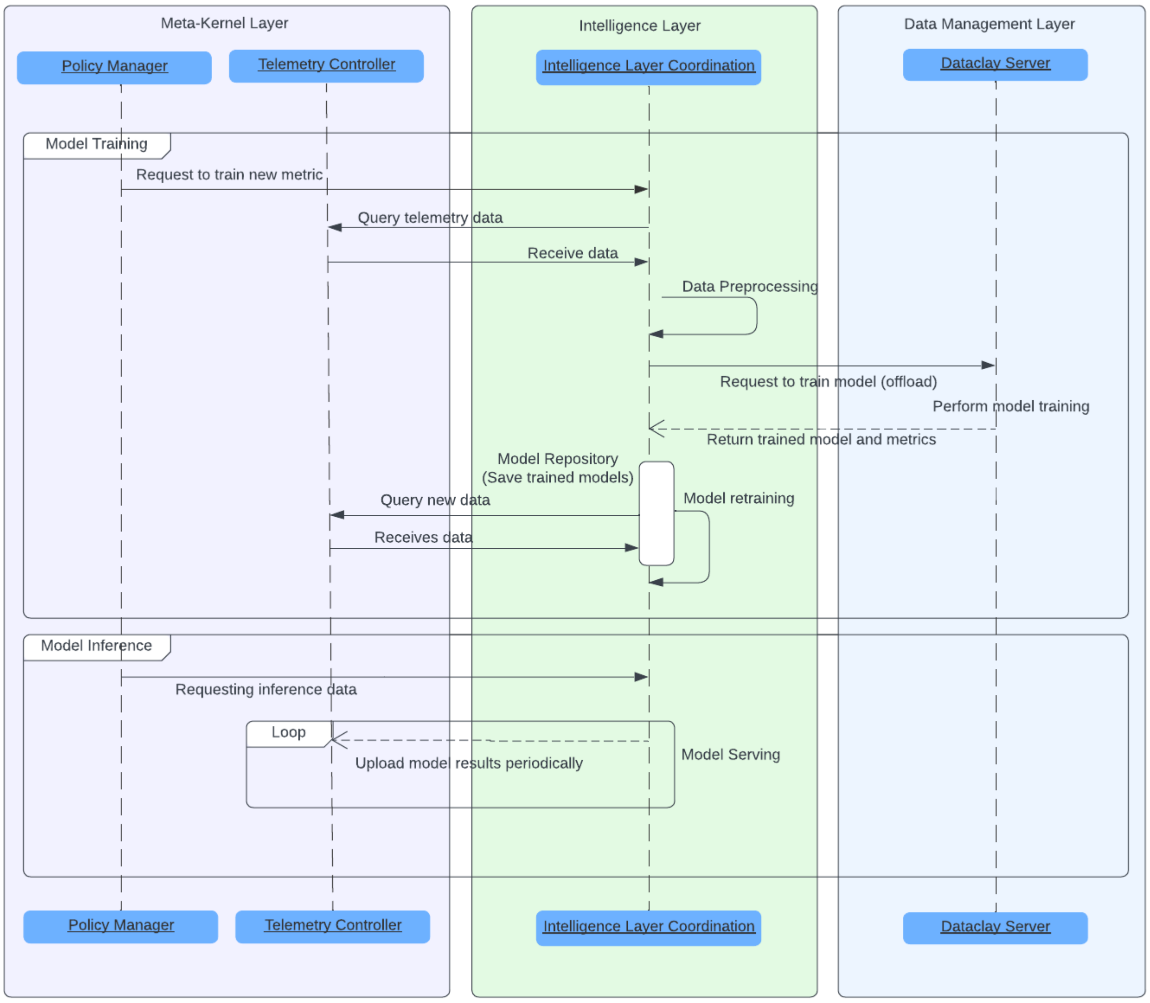


Figure 32. Sequence diagram for AI training offloading.

4.12.2 Federated Learning sequence diagram

In this subsection, we demonstrate the process of constructing the Federated Learning (FL) model within the ICOS system. FL is a decentralized approach to machine learning where multiple clients (or nodes) collaboratively train a shared model while keeping their data localized. This paradigm enhances privacy, reduces data transmission costs, and leverages the computational power of multiple agents. Figure 33 depicts the sequence diagram that implements the FL training process for a system architecture with two ICOS agents, detailing the interactions between various components involved in the training process. Note that many FL ICOS agents can follow the same principles and steps.

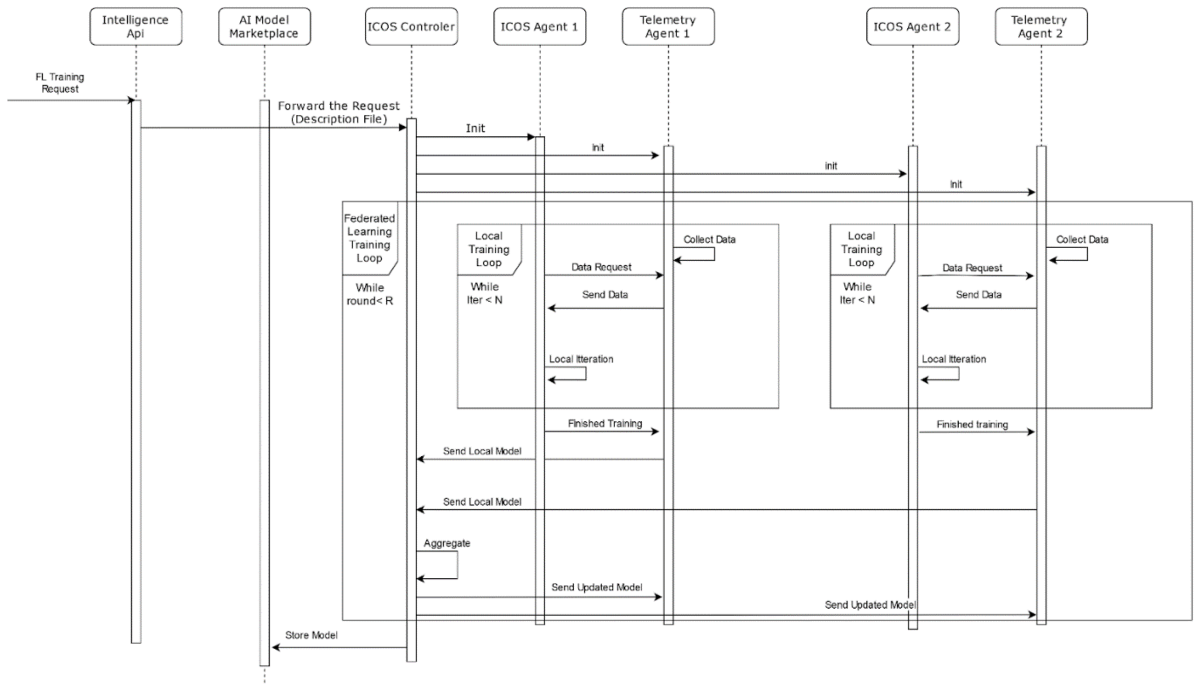


Figure 33. The sequence diagram for end-to-end training FL schemes.

The following steps describe how the FL training process unfolds:

- **FL Training Request:** The process begins with an FL training request through the Intelligence API. This request typically includes the specifications and requirements for the federated learning task as arguments.
- **Forward the Request:** The Intelligence API forwards the request and a description file to the ICOS controller to initiate and control the FL process. This description file contains the details necessary for initiating the federated learning process.
- **ICOS Controller Initialization:** The ICOS Controller orchestrates the federated learning process. Upon receiving the request, the ICOS Controller initiates the process by sending initialization commands to the ICOS Agent(s) and Telemetry Agent(s).
- **Initialization of ICOS and Telemetry Agents:** ICOS Agent 1, Telemetry Agent 1, and ICOS Agent 2 and Telemetry Agent 2 are initialized to prepare for local training and data collection.
- **Federated Learning Loop:** The ICOS Controller enters a loop for federated learning, iterating for a predefined number of rounds (*R*).
- **Separate Local Training Loops for ICOS Agent 1/2 and Telemetry Agent 1/2:** During each round, ICOS Agent 1 and 1 enter a local training loop. They request data from local sources (through Telemetry Agents), perform local iterations of training for a specified number of iterations (*N*), and update the local model.
- **Data Request and Collection:** The Telemetry Agents may handle data requests, sending the necessary data to their respective ICOS Agents for local training iterations. This step ensures the data remains localized while allowing the training to proceed.
- **Sending Local Models:** Once the local training loops are completed for the current round, ICOS Agents return their locally trained model parameters to the ICOS Controller.
- **Aggregation and Updating the Global Model:** The ICOS Controller aggregates the received local models to form an updated global model. This aggregation typically involves –but is not limited to techniques like averaging the weights of the local models.
- **Sending Updated Model:** The ICOS Controller sends the updated global model back to the ICOS Agents for the next round of local training.

- **Completion of Federated Learning Process:** The federated learning loop continues until the predefined number of rounds (*R*) is completed. After the final round, the ICOS Controller stores the final global model, completing the federated learning process.
- **Storing the Final Model:** The ICOS Controller stores the final federated learning model in a designated storage, making it available for future inference, further training, or re-usage if needed.

5 Conclusions

This deliverable presents the final ICOS System architecture as the main outcome of project's task T2.4, “Architectural Design". The architecture has been defined through a thoughtful review of deliverable D2.2 "ICOS Architectural Design (IT-1)" [1] that presented the initial ICOS architecture, and leveraging several valuable contributions from the work developed under WP3, WP4 and WP5 within the first project iteration (IT-1).

The logical architecture has been specifically designed and adapted to each ICOS suite: the ICOS Controller, the ICOS Agent, the ICOS Shell and the ICOS Lighthouse. The ICOS Controller has been further designed through the Meta-Kernel Layer, the Intelligence Layer and the Security Layer, plus the transversal Data Management Layer. For each suite and layer, a set of logical components has been presented and carefully described in order to better manage its complexity and analyse its design.

An analysis of the ICOS System from an operational point of view has been conducted and reported in this document. The operational architecture has also been presented and discussed through a comprehensive set of sequence diagrams, considering the Cloud Continuum Operations lifecycle supported by ICOS. The sequence diagrams have been organised into three functional categories: operations related to the continuum management, operations related to the deployment of a vertical application, and operations related to the observation of a deployed application that are mainly involved in the intelligence and security layers.

This deliverable is of utmost importance to fuel the implementation process of the technical work in Work Packages WP3, WP4 and WP5, supported by the system design and implementation specification stated in this report, which will be responsible for shifting the single components into 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 second period (from M19 to M36) of the project in the technical Work Packages.

All project partners collaborated toward the definition of the final ICOS system architecture following a well-defined methodology that started by formalising common guiding principles aimed at 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.

6 References

[1] ICOS. D2.2 – "ICOS Architectural Design (IT-1)", Gabriele Giammatteo, 2023.

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

[3] ICOS. D3.1 – "Meta-Kernel Layer Module Integration", Konstantin Skaburskas, 2023.

[4] ICOS. D4.1 – "Data management, Intelligence and Security Layers", Hrvoje Ratkajec, 2023.

[5] The AI Act, https://artificialintelligenceact.eu/the-act/ retrieved 2024/06/30.