D3.2-Meta-Kernel Layer Module Developed IT-2

Document Identification			
Status	Final	**Due Date**	31/10/2024
Version	1.0	**Submission Date**	31/10/2024

Related WP	WP3	Document Reference	D3.2
Related Deliverable(s)	D3.1, D2.4	Dissemination Level (*)	PU
Lead Participant	BSC	Lead Author	Francesc Lordan (BSC)
Contributors	ATOS, NCSRD, PSNC, L-PIT, SUITE5, XLAB, ENG, UPC, BSC, TUBS, NKUA, CRF, IBM	Reviewers	Izabela Zrazinska (Worldsensing)
			Marc Michalke (TUBS)

Keywords:
Cloud, Edge, IoT, MetaOS, development, integration

Document Information

List of Contributors	
Name	Partner
Francesc Lordan	BSC
Alex Barceló	BSC
Marc Michalke	TUBS
Gabriele Giammatteo	ENG
Maria Antonietta Di Girolamo	ENG
Manuel Gallardo	ATOS
Ivan Paez	ZSCALE
Jordi Garcia	UPC
Xavi Masip	UPC
Montse Carreras	UPC
Kalman Meth	IBM

Document History			
Version	Date	Change editors	Changes
0.1	30/09/2024	BSC	First draft version of ToC.
0.2	24/10/2024	ALL	Main content for all sections
0.3	25/10/2024	BSC	Aggregation, restructuring and formatting
0.4	30/10/2024	TUBS, WSE, BSC	Internal review
0.5	31/10/2024	BSC	Draft version for quality review
1.0	31/10/2021	ATOS	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Francesc Lordan (BSC)	31/10/2024
Quality manager	Carmen San Roman	31/10/2024
Project Coordinator	Francesco D'Andria	31/10/2024

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
CLI	Command-Line Interface
CRD	Custom Resource Definition (Kubernetes term)
DPM	Dynamic Policy Manager
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
GUI	Graphical User Interface
IT-x	Development Iteration x
JM	Job Manager
OCM	Open Cluster Management (software)
PM	Policy Manager
REST	REpresentational State Transfer
WP	Work Package

Executive Summary

This document titled “D3.2 Meta-Kernel Layer Module Developed (IT-2)” focuses on the modifications and improvements that the ICOS project Meta-Kernel Layer Module has experienced during its transition from IT-1 into IT-2. Several parts of the architecture were already discussed in previous deliverable “D3.1 Meta-Kernel Layer Module Integrated (IT-1)” [1]. The development of the Meta-Kernel Layer Module for IT-2 leverages on the outcome of “D2.4 ICOS Architectural Design (IT-2)” [2]. Both the development work described in this document as well as the document itself build upon the foundation laid by D2.4. Furthermore, the scope of this document covers the conceptual design with concrete implementations of the components, with a special focus on the changes born from the transition from IT-1 into IT-2. The primary purpose of this document is to provide an account of the design and implementation of the ICOS Meta-Kernel Layer Module (for IT-2), which is a critical component of the ICOS project. This module plays a major role in the larger framework of the ICOS Meta Operating System.

The current document is an essential resource for all stakeholders involved in the ICOS project, offering a detailed account of the Meta-Kernel Layer Module's development and integration for IT-2. It encapsulates the collective efforts on changes and improvements in this period (since IT-1), bringing the project one step closer to its ultimate objectives.

1 Introduction

1.1 Purpose of the document

This document is a report on the design, development, and implementation of ICOS Meta-Kernel Layer Module at the end of development iteration 2 (IT-2).

1.2 Relation to other project work

This document uses results of "D2.4 ICOS Architectural Design (IT-2)" [2]. Namely, it considers the elaborated architecture of ICOS Meta-Kernel Layer, its primary functions and interfaces with other layers of the ICOS Meta Operating System.

This document complements "D3.1 Meta-Kernel Layer Module Integrated (IT-1)" [1] by building on its findings and addressing additional aspects that arose during IT-2 development.

1.3 Structure of the document

The content in this document is divided in 2 major chapters

- **Chapter 2** presents the System User Stories with special focus on novel aspects related to IT-2. This chapter also includes further details on the components and their interfaces.
- **Chapter 3** presents the selection of new tools that have been introduced in the ICOS project for the implementation of components since IT-1.

2 Meta-Kernel Layer Module Design

As shown in Figure 1 (Figure already present and discussed in D2.4 [2]), the Meta-Kernel Layer is now present in both the ICOS Controller and the ICOS Agent, instead of just the ICOS Controller as documents related to IT-1 suggested.

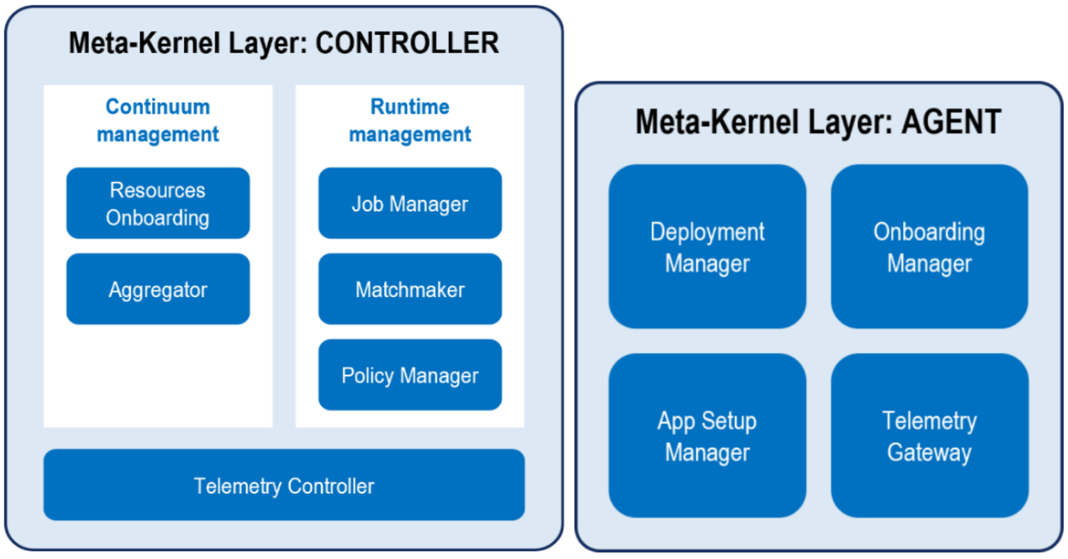


Figure 1: The ICOS Meta-Kernel Layer (Controller and Agent)

This chapter focuses on the new component, changes, improvements and new functionalities for IT-2. For this purpose, the following sections will provide concrete design considerations as well as components and interfaces related to the new IT-2.

2.1 Components and interfaces

2.1.1 user.shell

The ICOS shell now consists of two parts; the command-line interface (CLI) client, which was already described in previous deliverables as well as the ICOS GUI. Both connect to the shell backend's RESTful API for all requests and allow for interfacing with the user. Update and Create commands expect files that adhere to the ICOS application descriptor. Furthermore, the shell holds the login token of the user.

Both components leverage the backend's OpenAPI implementation. While the CLI is a binary executed on any device, the GUI is available as an OCI image that can be run either on a client machine by using a runtime like Docker, or on a remote appliance (including and ICOS controller installation) using container orchestration platforms like Kubernetes.

2.1.2 user.shell-backend

The shell backend exposes a RESTful API at http://${CONTROLLER_ADDRESS}/api/v3 for defined queries which then result in further API calls to different components, i.e. the JobManager, the IAM, the aggregator or components of the intelligence layer. The API calls shown in Table 1 are fully implemented for the current version of the shell backend.

Table 1: API calls for shell-backend

HTTP request	Description
GET /resource/	Returns a list of resources
GET /resource/{resourceId}	Find resource by ID
GET /deployment/	Returns a list of deployments
POST /deployment/	Creates a new deployment
GET /deployment/{deploymentId}	Find deployment by ID
PUT /deployment/{deploymentId}	Updates a deployment
DELETE /deployment/{deploymentId}	Deletes a deployment
DELETE /deployment/{deploymentId}/start	Starts a deployment
DELETE /deployment/{deploymentId}/stop	Stops a deployment
GET /user/login	Logs user into the system
GET /user/logout	Logs out current logged in user session
GET /healthcheck	Health check
GET /controller/	Returns a list of controllers
POST /controller/	Adds a new controller
POST /metrics/train	Trains a model on a set of metrics
POST /metrics/predict	Predict metrics development based on model and input metrics

2.1.3 discovery.lighthouse

The Lighthouse component still shares the code base of the shell backend with the only relevant endpoints being GET /controller/ and POST /controller/. Despite being added to a new suite, this component underwent no changes since D3.1 [1].

2.1.4 runtime.job-manager

The Job Manager component has sustained some refinements to its API interface to enhance its functionality and to align with RESTful architectural principles. These improvements encompass a comprehensive reorganization of endpoints, aiming to create a more modular, maintainable, and scalable API architecture. Specifically, the API has been restructured to implement finer-grained resource representations and more precise endpoint definitions, promoting a clear separation of concerns.

Table 2 lists the resources provided by its RESTful API and its operations. The full schemas are provided in the Annexes.

Table 2: API calls for the Job Manager

HTTP request	Description
GET /jobmanager	Home route
GET /jobmanager/healthz	Healthcheck
GET /jobmanager/jobs	Retrieves all jobs
PUT /jobmanager/jobs	Updates a job
GET /jobmanager/jobs/executable/{orchestrator}?{icos_agent_id}	Lists executable jobs filtered by orchestrator ICOS agent ID
GET /jobmanager/jobs/{job_uuid}	Retrieves a specific job by UUID
DELETE /jobmanager/jobs/{job_uuid}	Deletes a specific job by UUID
POST /jobmanager/groups	Creates a new job group
GET /jobmanager/groups	Retrieves all job groups
PUT /jobmanager/groups	Updates a job group
GET /jobmanager/groups/{group_uuid}	Retrieves a specific job group by UUID
DELETE /jobmanager/groups/{group_uuid}	Deletes a specific job group by UUID
PUT /jobmanager/groups/stop/{group_uuid}	Stops a specific job group by UUID
PUT /jobmanager/groups/start/{group_uuid}	Starts a specific job group by UUID
GET /jobmanager/resources/status/{job_uuid}	Retrieves the resource state by Job UUID
PUT /jobmanager/resources/status	Updates the resource state by UUID
POST /jobmanager/policies/incompliance	Creates a policy incompliance entry

2.1.5 runtime.deployment-manager

The deployment manager component maintains a stable architecture, having undergone no significant modifications. Designed without exposing HTTP endpoints, it prevents direct user interaction and interfaces with the *runtime.job-manager* through its interface (see Table 2 and section 2.1.4).

2.1.6 runtime.app-setup

The App Setup is a new component that is required in order to make runtime information, like topology changes, available to the application and functional components related to job execution. IT-2 architecture design uses this information in the *runtime.execution-manager* and it is also used to maintain the application service connectivity between agents (through ClusterLink). In general, having this information available can be valuable for a wide variety of runtime decisions across the continuum.

This component acts with two main sources of information: Job Manager and Aggregator. On the one hand, information on the start and finalization of jobs is received from the Job Manager through the App Setup API shown in Table 3. On the other hand, information from the topology changes and other continuum modifications are obtained through polling the Aggregator (component which already has an interface for accessing the information required by the App Setup).

Table 3: API calls for the App Setup

HTTP request	Description
PUT /deployment	Notify a new job in the system
DELETE /deployment/{deployment_id}	Notify finalization of a job

The events and information generated by the App Setup are published through a global bus. Once the information is published into this bus, other components are able to consume this information. More detailed design and sequence diagrams are provided in section 2.2.3.

2.1.7 runtime.clusterlink

ClusterLink simplifies the connection between services that are located in different domains, networks, and cloud infrastructures. ClusterLink runs on Kubernetes-enabled clusters. The ClusterLink deployment operator allows easy deployment of ClusterLink to a Kubernetes cluster.

ClusterLink setup operations (create, deploy, delete) are performed using the ClusterLink CLI command. Once the ClusterLink operator is deployed on a cluster, operations (peer, export, import, policies) are performed through Custom Resource Definition objects (CRDs). The creation and manipulation of CRDs is done through the Kubernetes API. Examples of the CRD formats for these operations can be found at https://clusterlink.net/docs/main/tasks/operator/.

2.1.8 runtime.matchmaker

The interface for the runtime matchmaker remains the same in IT-2. The runtime matchmaker is triggered by the job manager, receives the application descriptor YAML file, and calls the continuum aggregator to retrieve the ICOS topology. The runtime matchmaker extracts the resources required for the user application and matches them with the available resources in the ICOS topology to find the best possible solution. It then responds to the job manager for the further deployment of the user application. More details on the design and inner workings of the Matchmaker components are provided in section 2.2.4. The Matchmaker API endpoints can be seen in Table 4.

Table 4: API calls for the Matchmaker

HTTP request	Description
GET /	Healthcheck
POST /matchmake	Receives the description of application. Contacts the Aggregator service in order to retrieve ICOS topology. Matchmaking is performed and the result is returned.

2.1.9 runtime.execution-manager

The purpose and the API of this component remains the same as in IT-1.

2.1.10 runtime.orchestrator-edge-cloud

This component has received maintenance and can be considered stable. It has not experienced significant changes on API nor interface since IT-1. It is now part of *runtime*.

2.1.11 runtime.policy-manager

The Policy Manager (or Dynamic Policy Manager) component has been implemented during IT-2, according to the design previously described in D3.1 [1].

Several API endpoints are provided in this component, as shown in Table 5. For a full definition of the URL and the schema on this REST calls see the Annexes.

Table 5: API calls for the Policy Manager

HTTP request	Description
GET /policies/{id}	Get a policy by id.
GET /policies/	Return the list of the policies created/updated
POST /policies/	Add new policies
GET /policies/status	Read item
POST /alertmanager/	AlertManager Webhooks
POST /registry/icos/	ICOS process app descriptor

2.1.12 continuum.aggregator

This component has received maintenance and can be considered stable. It has not experienced significant changes on API nor interface since IT-1.

2.1.13 edge.node

No changes between IT-1 and current IT-2.

2.1.14 telemetry.agent

The design, functionalities and interfaces for the Telemetry Agent remain constant from IT-1. The only difference at architectural level is that in IT-2 the Telemetry Agents sends data to the new Telemetry Gateway component and not directly to the Telemetry Controller. At implementation level, new metrics and plugins have been added. Further details will be discussed later in section 2.2.5.

2.1.15 telemetry.gateway

The Telemetry Gateway exposes an API to receive metrics from the Telemetry Agents using the OpenTelemetry Protocol (OTLP) format through HTTP and gPRC. The same protocols are used to send data to the Telemetry Controllers.

2.1.16 telemetry.controller

With respect to IT-1, the Telemetry Controller interface has been enriched with an API to manage alerts. The API is compliant with the Prometheus Alerting Rules format and exposes methods to create and delete alerting rules. This functionality is mainly exploited by the Policy Manager component that uses the Telemetry Controller alerts to monitor and enforce the policies.

Table 6 reports the rules API. More information on the implementation of the alerting API is provided in section 2.2.5.

Table 6: API calls for the Telemetry Controller

HTTP request	Description
POST /rules/	Add a new rule. Expect in the body of the request a rule object.
DELETE /rules/{id}	Delete the selected rule.

2.2 Detailed design

Following the description of the new components and interfaces provided in section 2.1, this section focuses on component design including relevant sequence diagrams that define ICOS and its components. This section describes the current status of integration and design for the Meta-Kernel Layer at IT-2.

2.2.1 Job Manager

The *runtime.job-manager* component is the core module of the ICOS Controller. It is responsible for comprehensive runtime management, encompassing control, persistence, and coordination among ICOS components. By providing a persistence layer, the Job Manager serves as the definitive source of truth within the system as described in D2.4 [2].

2.2.1.1 Lifecycle management

The *runtime.job-manager* comprehensive lifecycle management CRUD operations (Create, Read, Update, Delete) ensure the consistent and efficient execution of Jobs. These operations oversee both the state of individual Jobs and the state of the associated Resources within the owning ICOS Agents, thereby maintaining system integrity and performance. By managing the Job lifecycle, the Job Manager ensures that Jobs transition smoothly through various states, while adhering to the specified policies and requirements.

2.2.1.2 Policy Manager integration

The integration with the *runtime.policy-manager* component is a critical aspect of the Job Manager's functionality. The Job Manager notifies the Policy Manager each time an application undergoes actions such as creation, stopping, starting, updating, or deletion. This notification mechanism ensures that all policy-related considerations are continuously monitored and enforced throughout the application lifecycle. Additionally, the *runtime.job-manager* is responsible for addressing policy non-compliances identified by the Policy Manager. Upon receiving a non-compliance request, the Job Manager initiates a remediation process by updating the job's status with the necessary metadata required by the *runtime.deployment-manager* to take appropriate action. The Job Manager supports a wide range of remediation actions, including horizontal and vertical scaling, security operations, and the reallocation of deployments to different clusters or nodes.

2.2.2 Deployment Manager

The Deployment Manager provides an abstraction layer which is functionally analogous to a driver. This ICOS component clears the way for the integration of diverse orchestrator drivers within the system topology. Currently, ICOS includes two distinct deployment manager implementations: one for Nuvla and another for OCM. Both perform core functions: interacting with *runtime.job-manager* and deploying workloads on their respective orchestrators. This design ensures that the essential operations of *runtime.deployment-manager* remain unmodified, enabling the component to support multiple orchestration backends without modifying its core architecture.

2.2.2.1 Interaction with underlying orchestrators

The Deployment Manager is constantly pulling jobs from the *runtime.job-manager* component. For a job to be deemed executable, it must comply with the requirements defined by the query logic that interfaces with the database. Once a job has been pulled, the *runtime.deployment-manager* is given instructions on the operations to perform with the job. These instructions can vary widely, ranging from something as simple as deploying an application to performing some kind of reallocation on an already deployed resource. Just like the underlying orchestrators, the *runtime.deployment-manager* functions in a pull mode, periodically scheduling calls to a specific endpoint exposed by the *runtime.job-manager* API to fetch a list of executable jobs.

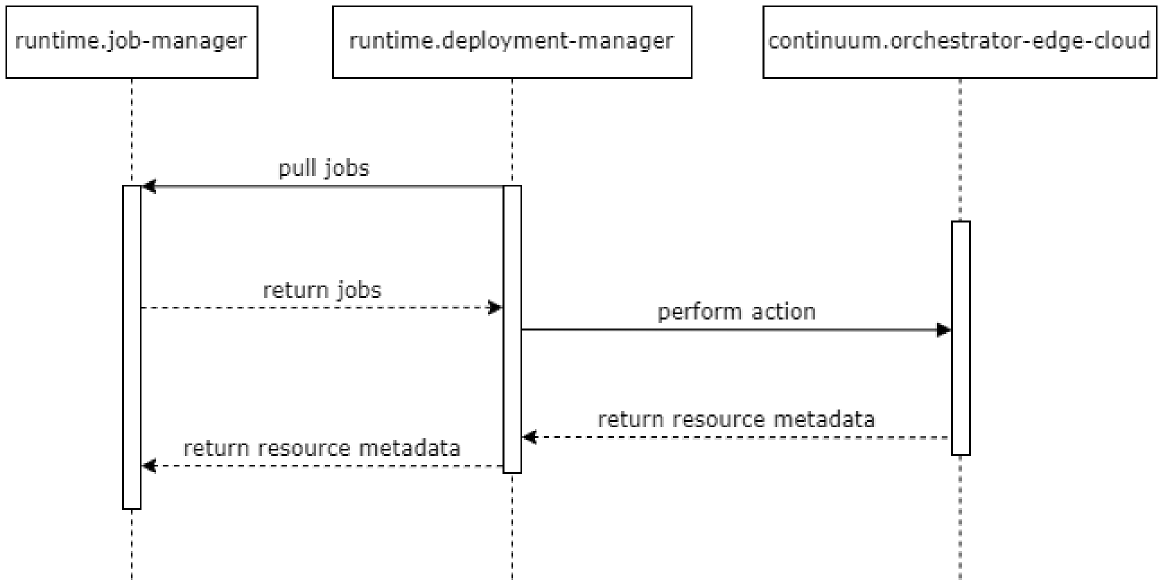


Figure 2: Sequence diagram for the interactions of Deployment Manager

Figure 2 shows the sequence diagram of the Deployment Manager. In this diagram we can see on the left side the interactions between the Job Manager and the Deployment Manager. On the right side we can see the interaction between Deployment Manager and the underlying orchestrators.

Each instance of the *runtime.deployment-manager* is associated with an unique ICOS agent identifier (*icos_agent_id*). This identifier is supplied via runtime configuration settings specific to each ICOS Agent. Having this unique identifier ensures that each deployment manager instance retrieves only the jobs pertinent to its cluster resource manager of choice, thereby maintaining operational isolation and preventing cross-cluster interference.

2.2.2.2 Multi-Cluster Setups

Executing jobs in a scenario with multiple clusters requires several additional considerations. Consider a scenario where an ICOS controller is managing multiple ICOS agents. Each of these agents are deployed in separate environments with their own Deployment Manager instances. To pull executable jobs, they will need to make calls to a route with the following pattern:

- GET /jobmanager/jobs/executable/{orchestrator}?{icos_agent_id}

Each *runtime.deployment-manager* instance is assigned a unique *icos_agent_id*, ensuring that they only interact with jobs intended for their specific cluster.

By using a unique *icos_agent_id* we prevent race conditions in multi-cluster deployments where multiple instances of the same type of deployment manager may be running concurrently. This strategy enforces that each deployment manager retrieves jobs exclusively tied to its appointed ICOS agent. As a result, we eliminate the risk of deploying a workload in the wrong infrastructure.

2.2.3 App Setup

As discussed in section 2.1.6, App Setup has a REST API used by the Job Manager and performs polling unto the Aggregator. When there are relevant changes, those are notified through the bus. Figure 3 shows a sequence diagram to highlight the interaction between those components and a Distributed Parallel Execution –an entity part of the *runtime.execution-manager* which will make use of the runtime information provided by the App Setup.

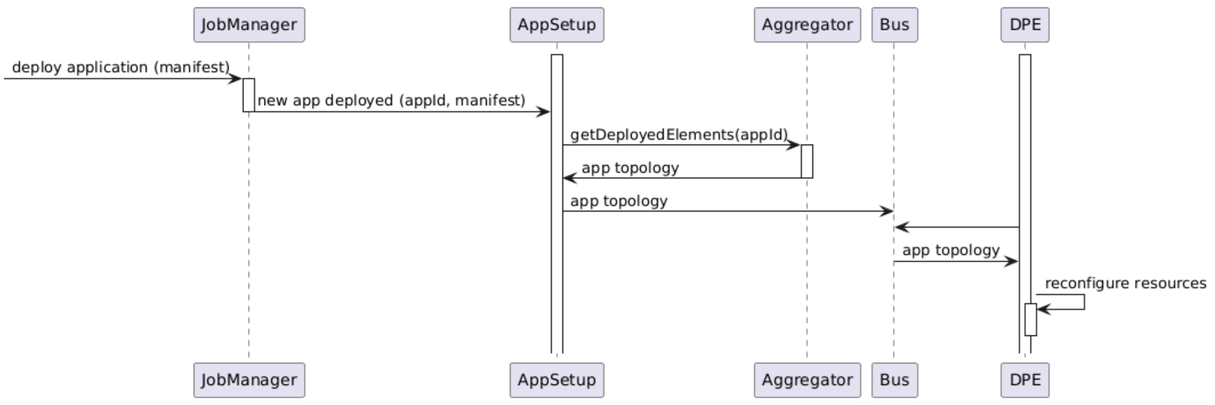


Figure 3: Sequence diagram for the App Setup interaction with DPE

Figure 3 presents how the Job Manager contacts the App Setup on a new app event through the App Setup interface. Additionally, the App Setup will continuously monitor the Aggregator, from which it will obtain real time information on the deployment status of the application. With this information, it is able to emit app topology information into a bus and the Distributed Parallel Execution will consume this app topology in order to reconfigure its resources.

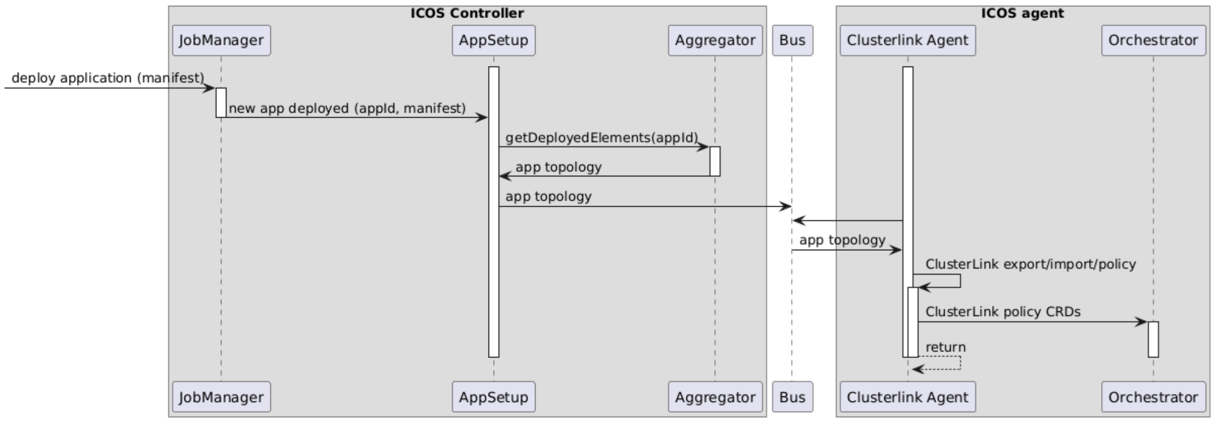


Figure 4: Sequence diagram for the App Setup interaction with ClusterLink

A similar diagram can be seen in Figure 4, which shows how the information given by the App Setup (broadcasted through the bus) is consumed by the ClusterLink agent that, eventually, transmits the required changes through the orchestrator. More information on the ClusterLink operational procedures is provided in section 3.1.1.

2.2.4 Matchmaking

As described in D3.1, the Matchmaker is triggered by the Job Manager, and receives an application descriptor YAML file as well as the ICOS topology retrieved through the Aggregator component. The Matchmaker extracts the resources required for the user application and matches them with the available resources in the ICOS topology to find the best possible solution for the application deployment. It then responds to the job manager for the actual deployment of the user application.

In IT-2, the Matchmaker core process has been redesigned with respect to the previous version in order to provide better solutions and meet the new requirements of this release. The current version is based on a scoring mechanism to select the best nodes to execute the application's components as well as on affinity interrelationship features to prioritize nodes based on locality. Moreover, the Matchmaker considers the technology-dependent compatibility of the application component mapping on node compatibility within the ICOS topology, ensuring that all components within a single application are placed on the same type of node.

The Matchmaker algorithm for IT-2 operates through the following four steps:

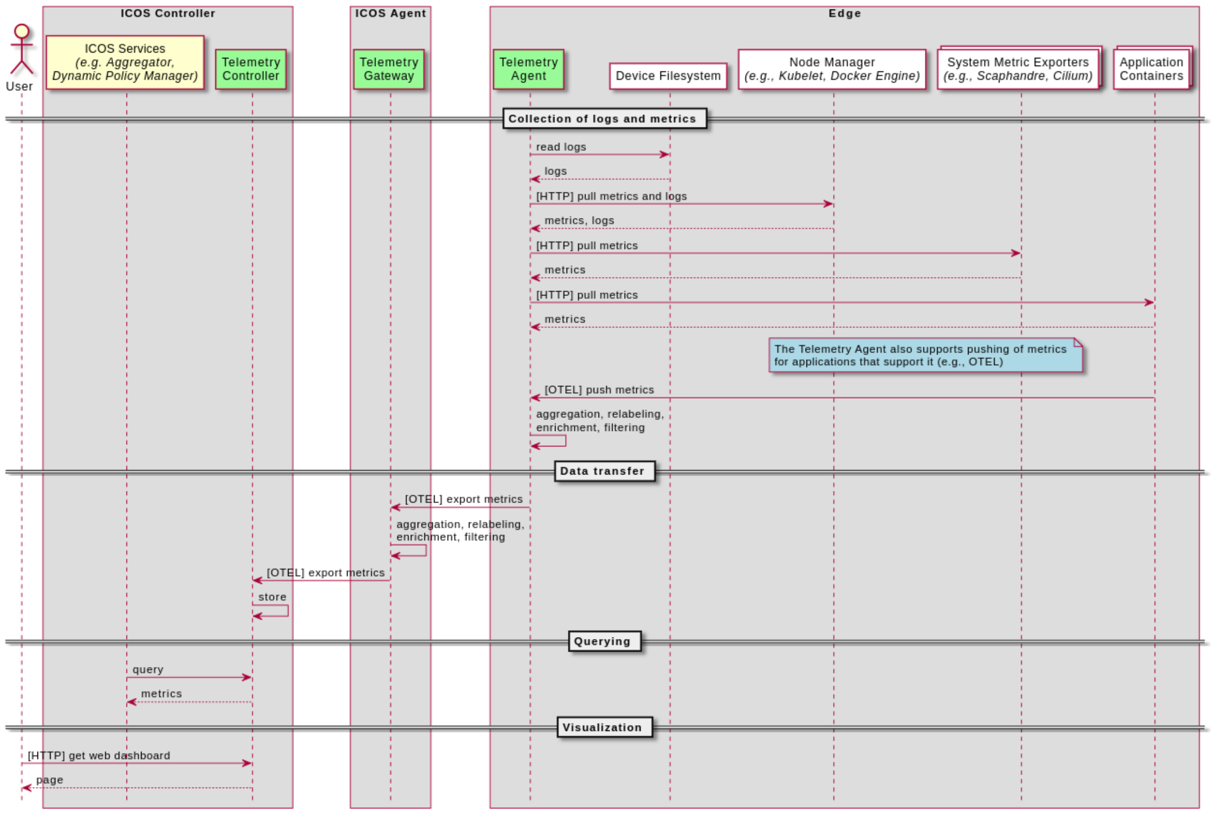
1. **Filtering:** It extracts the resources required for each component by parsing the application header and filters out the candidate nodes from the ICOS topology that fulfil the hard constraints defined for each component and the user-defined policies.
2. **Scoring:** After the filtering stage, the score function is invoked to assign scores to the filtered nodes of each component. The scoring mechanism ranks the nodes based on performance, security, and energy consumption, with scores normalised between 0 and 100. Weights are assigned to prioritise the scoring of nodes based on user demands for performance, security, and energy consumption.
3. **Affinity:** After the scoring stage, an affinity function is invoked to prioritize the interrelation between selected nodes. For instance, it gives higher priority to nodes within the same cluster compared to nodes in different cluster regions. To rank the nodes link, factors such as cost and latency to other nodes can be considered. Evaluating the delays between nodes helps optimise

the application's end-to-end latency and maintain the quality of service by selecting nodes close to the connected application components.

4. **SelectBest:** The matchmaking then selects the combination of nodes with the highest scores by implementing a max-flow algorithm.

2.2.5 Logging and Telemetry

In IT-2, a new architectural component has been added: the Telemetry Gateway. The main role of this component is to decouple the Telemetry Agents from the Telemetry Controllers. In the new architecture, the Telemetry Agents send their data to the Telemetry Gateway. The Gateway then forwards the data to the Telemetry Controller (see diagram in Figure 5).



ICOS Controller
ICOS Agent
Edge
User
ICOS Services (e.g. Aggregator, Dynamic Policy Manager)
Telemetry Controller
Telemetry Gateway
Telemetry Agent
Device Filesystem
Node Manager (e.g., Kubelet, Docker Engine)
System Metric Exporters (e.g., Scaphandre, Cilium)
Application Containers
Collection of logs and metrics
read logs
logs
[HTTP] pull metrics and logs
metrics, logs
[HTTP] pull metrics
metrics
[HTTP] pull metrics
metrics
The Telemetry Agent also supports pushing of metrics for applications that support it (e.g., OTEL)
[OTEL] push metrics
aggregation, relabeling, enrichment, filtering
Data transfer
[OTEL] export metrics
aggregation, relabeling, enrichment, filtering
[OTEL] export metrics
store
Querying
query
metrics
Visualization
[HTTP] get web dashboard
page

Figure 5: Logging and Telemetry components communications

This change brings several benefits:

- It follows more closely the ICOS Continuum architecture.
- It supports new use cases where the ICOS Controllers are not directly reachable from the edge devices.
- It allows to pre-process data at ICOS Agent level (before sending it at the Controller). This is particularly important since it allows to a) filter out data not needed at Controller level and, therefore, reduce the data stored in the Controller, and b) to enrich data with additional metadata (e.g. the “icos-agent-id").
- It allows to store and analyse data at the Agent directly. This makes it possible for components in the ICOS Agent to exploit the data directly without relying on the Controller. This feature will be exploited by the Intelligence Layer.

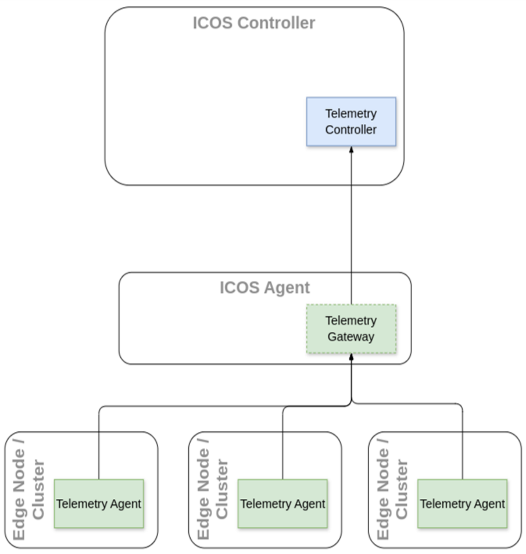


Figure 6: Distribution of the Telemetry components across the Continuum nodes.

As depicted in Figure 6, the Telemetry Gateway has been designed to be deployed in the ICOS Agent nodes. It receives data from the Telemetry Agents and forward it (after a processing step) to the Telemetry Controller.

2.2.6 Enforcement of policies

In order to adapt the orchestration of an application based on the application and/or system conditions and events following pre-defined policies, an important integration has been realized between the Job Manager, the Dynamic Policy Manger and the Logging and Telemetry subsystem.

As depicted in the diagram in Figure 7, the Job Manager invokes the Dynamic Policy Manager component when a new application is deployed. The Dynamic Policy Manager parses the Application Descriptor and instantiates the policies defined in it. Policies are based on data measured and collected at the edge. The Dynamic Policy Manager registers (using a new API described in section 2.1.16) the necessary alerting rules to be notified when the data measured is not compliant with the conditions expressed by the policies.

During the application runtime, the Dynamic Policy Manager will be notified every time the metrics received from the edge and the application are violating the rules defined at deployment time. The Dynamic Policy Manager processes the event and, if this results in a policy violation, notifies the Job Manager. The Job Manager will then take appropriate actions to re-establish a deployment that can satisfy the policies defined (e.g., migrate the application to another node, scale-up the application, etc.).

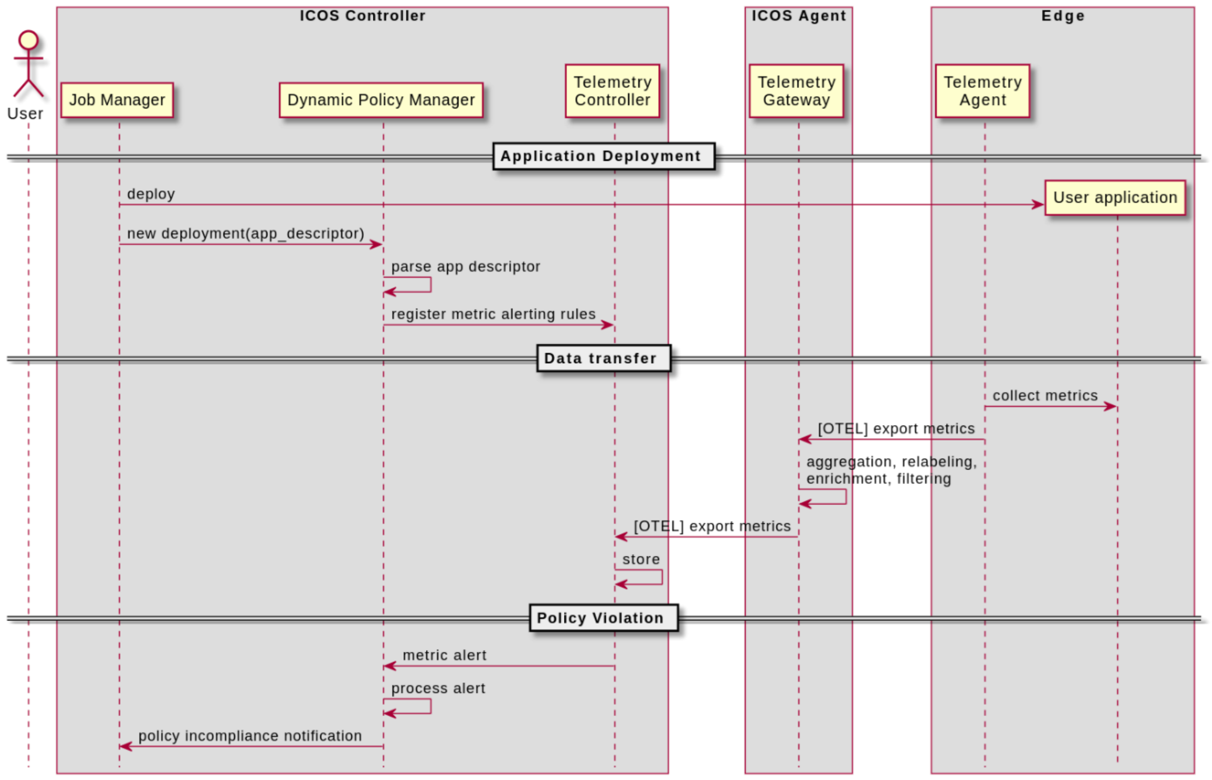


Figure 7: Policies definition and enforcement

3 Technology and Tools Selection for Implementation

This chapter describes new technologies that have been added to ICOS IT-2. All previously discussed technologies and tools (i.e. those discussed in the homonym chapter in D3.1 [1]) remain true and are part of IT-2 implementation.

The following sections will discuss components or functionalities in which a new technology or tool has been adopted.

3.1 Inter-agent communications

During IT-2 development, several new scenarios that need to perform communication across different ICOS agents and/or ICOS controller were detected. We can roughly consider two main scenarios that require this kind of communications:

- Communication between ICOS components from different agents.
- Communication between parts of the application that are deployed in different agents.

In general, ICOS tries to minimize the assumptions regarding connectivity and networking across different parts of the infrastructure. Considering the continuum context of this project, there is the need to provide some basic functionalities to the application in addition to certain internal requirements on communication between ICOS components.

For this goal, we introduce a new technology: ClusterLink. ClusterLink deploys a gateway into each cluster, facilitating the configuration and access to multi-cloud services. It establishes secure, service-level connectivity between multiple clusters.

This, along with its distributed control plane and fine-grained policy control over connection establishment and access policies, is the main advantage of ClusterLink over some of its competitors.

During the development of App Setup, ICOS team has identified a need of a decentralized and distributed data exchange. From the App Setup point of view, a bus is required for sending information to agents (as shown in the sequence diagrams from Figure 3 and Figure 4). The technology chosen to satisfy this requirement is Eclipse Zenoh, a technology already present in WP4 and thus part of ICOS.

3.1.1 ClusterLink

The ClusterLink gateway contains the following internal subcomponents:

- **Control Plane** (CP) is responsible for maintaining the internal state of the gateway, for all the communications with the remote peer gateways by means of the ClusterLink CP Protocol, and for configuring the local data plane to forward user traffic according to policies. Part of the control plane is the policy engine that can also apply network policies (ACL, load-balancing, etc.)
- **Data Plane** (DP) responds to user connection requests, both local and remote, initiates policy resolution in the CP, and maintains the established connections. ClusterLink DP relies upon standard protocols and avoids redundant encapsulations, presenting itself as a Kubernetes service inside the cluster and as a regular HTTP endpoint from outside the cluster, requiring only a single open port (HTTP/443) and leveraging HTTP endpoints for connection multiplexing.

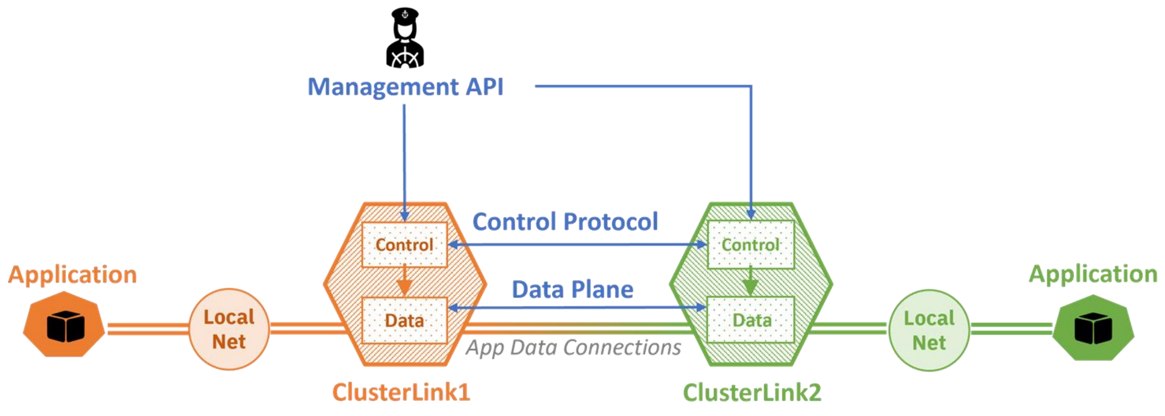


Figure 8: ClusterLink architecture

Figure 8 shows the high-level overview of the ClusterLink architecture. ClusterLink leverages the Kubernetes API using CRDs to configure cross-cluster communication. ClusterLink management is based on the following key concepts:

- **Peers**: Represent remote ClusterLink gateways and contain the metadata necessary for creating protected connections to these remote peers.
- **Exported services**: Represent application services hosted in the local cluster and exposed to remote ClusterLink gateways as Imported Service entities in those peers.
- **Imported services**: Represent remote application services that the gateway makes available locally to clients inside its cluster.
- **Policies**: Represent communication rules that must be enforced for all cross-cluster communications at each ClusterLink gateway.

For further information, please refer to the concepts section on the ClusterLink website: https://clusterlink.net/docs/latest/concepts/.

3.1.2 Zenoh

Eclipse Zenoh is a communication and networking protocol that enables developers to unify the way they interact with the data across its journey: a) capture of data: sensors capture data at the edge, b) data transmission: data is transmitted from the edge to its destination, c) computation and storage: data is stored as is, or after computation, and d) retrieval of data: data is retrieved, often for further processing. Existing communication protocols do not care about computation, storage, and retrieval. Eclipse Zenoh exploits this opportunity and provides Pub/Sub/Query protocol and a set of unified abstractions for data in motion, data at rest, and computations at the Internet Scale.

In the context of Edge-to-cloud continuum, IoT solutions are faced with a set of compute, storage, and communication resources that span from the microcontroller to the cloud, it is in their benefit to leverage this potential. Eclipse Zenoh runs efficiently on server-grade hardware and networks, microcontrollers, and constrained networks. Finally, Zenoh supports peer-to-peer, routed, and brokered communication, thus allowing for an optimal communication model at each system stage.

Key differentiators of Eclipse Zenoh are:

- **Cloud to Microcontroller Communication**. Zenoh can work efficiently and perform from server-grade hardware and networks to the embedded microcontroller and extremely constrained networks. Zenoh allows data to freely flow vertically and horizontally from the microcontroller to the data center, providing developers with a single solution for data distribution, and the ability to integrate technologies to bridge the communication between the enterprise and the embedded world.

- **Data Centricity and Location Transparency for Data in Movement and at Rest**. Location transparency is a consequence of data centricity – in these systems users only need to express interests without any concern about the location of its source. This feature is essential as it makes dealing with scale, failures, and load-balancing easier.
- **It Runs Over Any Topology**. Zenoh can operate under different models, allowing it to run over any topology and anywhere across the continuum. It can run in (i) a peer-to-peer fashion way, allowing the creation of clique or mesh topologies; (ii) a brokered fashion way, where nodes have only a limited set of functionalities and rely on the network to provide the full Zenoh capabilities; (iii) routed fashion way, where nodes act as software routers that forward Zenoh messages between nodes, as shown in Figure 9.

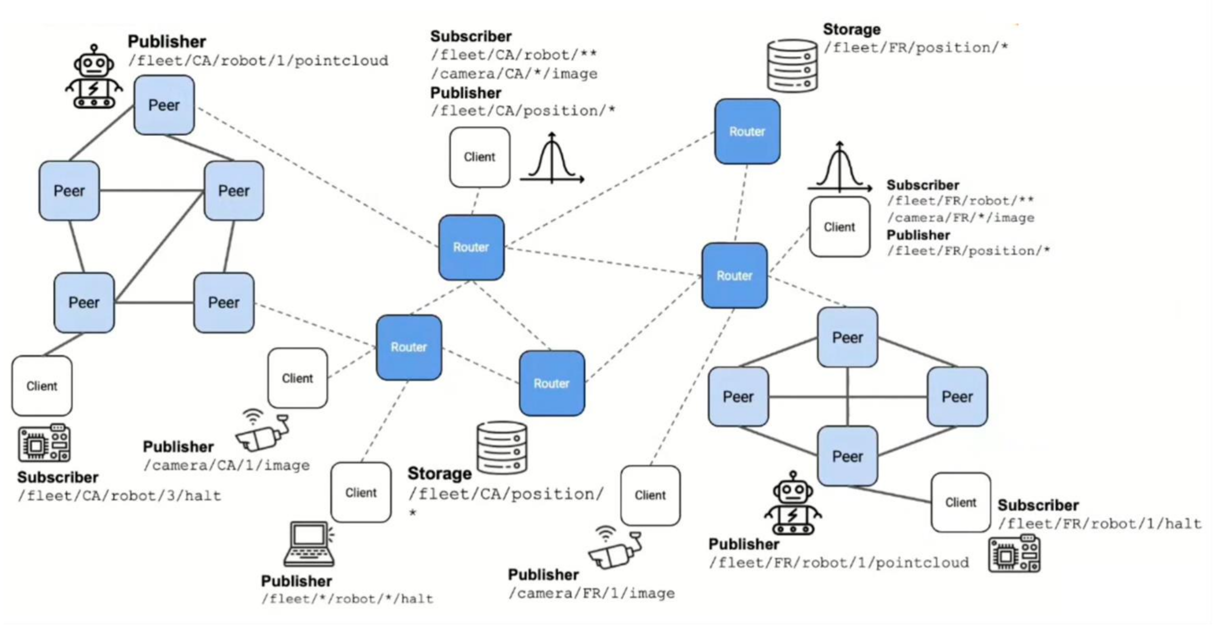


Figure 9: Illustration of a distributed system in an IoT/Edge continuum context

Eclipse Zenoh APIs are available in different programming languages such as: Rust, C, C++, Python, Java/Kotlin. Zenoh core's API has been developed in Rust language to provide security by design and avoid memory leakage.

Subscriptions can leverage wildcards to denote a set of resources of their interest related to a specific vehicle or a set of vehicles, as follows:

```
subscribe fleet/CA/robot/3/halt
```

Query are also supported, an example of a *get* operation is the following:

```
get fleet/FR/robot/1/pointcloud
```

Query operations also support wildcard **, which will bring multiple results (i.e. all results that are present at the system).

At the time of writing this deliverable, Eclipse Zenoh is used as a data bus for inter-process communication (IPC) for multiple Kubernetes clusters. Eclipse Zenoh is also used in the Use Case 1 Agriculture Operational Robotic Platform for Robot-to-Cloud communication developed by the L-PIT and PSNC partners. The latest version of Eclipse Zenoh is v.1.0.0, it was released on 21 October 2024.

3.2 Logging and Telemetry

Figure 10 highlights the main third-party technologies integrated to realize the telemetry components. With respect to the previous version presented in D3.1 [1], the following new third party technologies have been used:

- The Telemetry Gateway component has been implemented using the OpenTelemetry and Prometheus protocols and tools. This ensures a perfect compatibility and uniformity with the other telemetry components.
- The Alerting API has been implemented using two components:
    1. Prometheus AlertManager (Apache 2.0 license) to send alerts. https://prometheus.io/docs/alerting/latest/alertmanager/
    2. Prometheus-API project (MIT license) to manage alerting rules programmatically. https://github.com/hayk96/prometheus-api

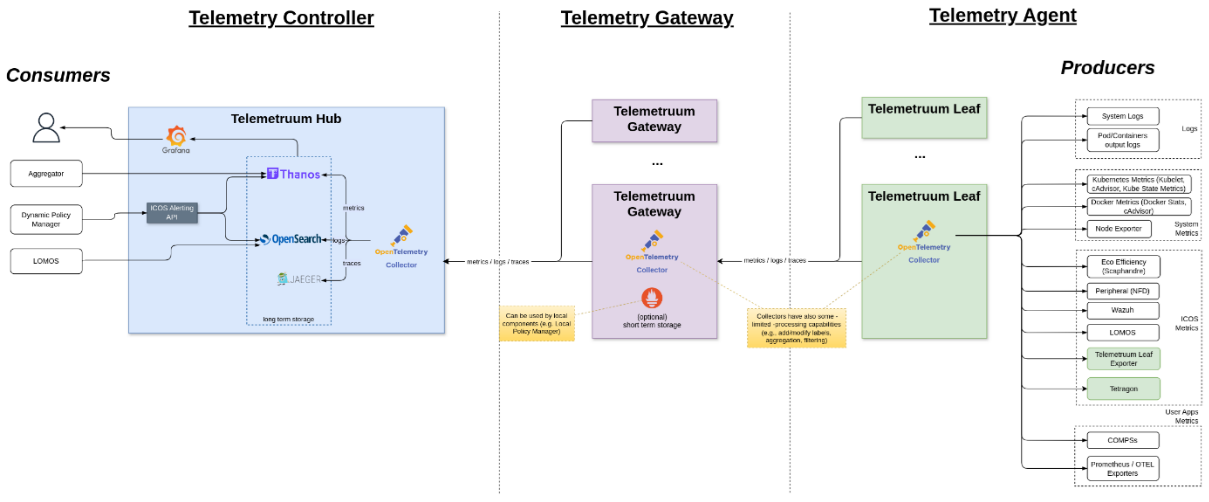


Figure 10: Logging and Telemetry technologies

In order to avoid confusion and clashes with ICOS concepts and terminology (e.g., the word "Agent" is use in both contexts), a renaming of the software module has been done. In particular:

- **Telemetruum Hub** is the software module that implements the Telemetry Controller architectural component.
- **Telemetruum Gateway** is the software module that implements the Telemetry Gateway architectural component.
- **Telemetruum Leaf** is the software module that implements the Telemetry Agent architectural component.

Note that Figure 10 shows the technologies with the renaming to avoid confusion.

3.2.1 Metrics Collected

Concerning the metrics collected at the edge, in addition to the metrics mentioned in deliverable D3.1, some improvements have been introduced:

- A new component named **Telemetruum Leaf Exporter** has been implemented. This component fills a gap generating some metrics there were missing to allow ICOS to manage the edge devices. The component generates the following metrics:
  - *tlum_host_info*: provides generic information about the device (e.g. operating system, hostname, ip address).
  - *tlum_orch_info*: provides information about the ICOS orchestrator that manages the device (e.g. OCM/Nuvla agent id).
  - *tlum_runtime_info*: provides information about the local runtime platform (i.e. docker or kuberentes versions).
  - *tlum_workload_info*: provides information about the ICOS applications (or components of an applications) that are running on the device (e.g., name, id, labels).
  - *node_mounted*: for Nuvla only, provides information about the currently available attached peripherals.
- Logs and metrics from Tetragon (https://tetragon.io) have been integrated. Tetragon is a tool integrated in WP4 for auditing the security of ICOS devices. Results from the security assessment and policies are ingested by the Telemetry components and made available to the other ICOS components (e.g., to define orchestration policies).
- Fine tuning and improvements of the metadata (i.e., labels) attached to the metrics to make them more easily usable in ICOS.

3.3 Policy Manager

As reported in deliverable "D3.1 Meta-Kernel Layer Module Integrated (IT-1)" [1], the Policy Manager is used to support and enable policies for other components of the ICOS project.

The first release of the PM provided a first version about the architecture, design of the component, the GUI and the methodology about the use of the PM; specifically an API REST is implemented for the use of the Policy Manager to read and create a policy (more details available in previous D3.1 [1]).

The second release of the PM includes the following implementation:

- Integration with the other ICOS components, such as Job Manager, App Descriptor, and dataClay as storage backend.
- Policies improvements about:
  - features: activate/deactivate/delete policy actions.
  - models: a new implementation of the model where more details are added about the functionality, description of the policy.
- An user friendly GUI is re-designed and implemented to use policies and its features integrated with the Keycloak tool.
- Creation of the documentation website.

3.3.1 Policies Integration

One of the important achievements for the Policy Manager is to integrate with other ICOS components and tools such as:

- dataClay as a policy storage backend: in the previous release the storage of the information policies used MongoDB as a non-relational database and a simple file text. The current release is able to use dataClay (part of the Data Management Layer) as a backend instead.
- In the new release a new redesign of how to storage is provided in order to make uniform and compliant the integration with the ICOS project.
- Job Manager where:
  - PM manager will forward these to JM as part of an incompliance.
  - JM will enrich the related job and make it executable by the owner.
  - PM will receive the job, extract the subtype(action) and will enforce it with specific strategy as for example .
  - PM will report the status to JM.
- Application descriptor: provides an HTTP REST API to parse an Application Descriptor manifest and set-up the policies for the application based on the content of the descriptor.

3.3.2 Policies Model

A common model for expressing the policies is adopted and used to store the policies internally and to transfer the policies in the ICOS MetaOS. Specifically, a policy is made of three essential parts:

- a subject (subject)
- a specification (spec)
- an action (action)

The JSON format file can be seen in the Annexes.

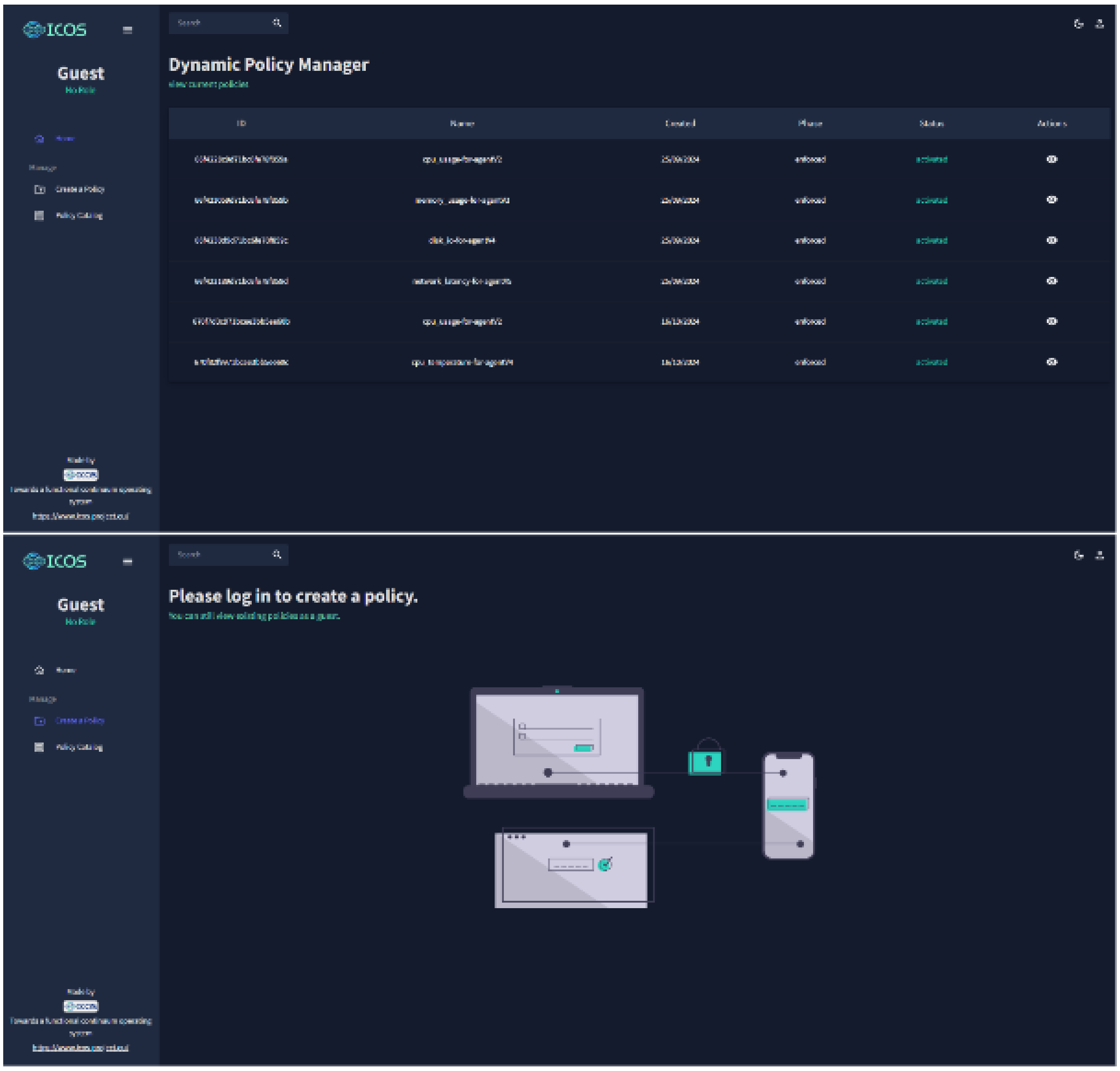
3.3.3 GUI

The Policy Manager component includes a dedicated GUI to help the user. The objective is to easily use the ICOS policies. The main achievements are the following:

- The integration with the Keycloak to manage how to access to the policies: guest user and user mode access.
- Reading and creation of the policies.

And the Keycloak integration, consisting of the following list of features:

  - Guest Mode: Guest users can only view the list of policies but cannot edit them (Figure 11).
- User Mode: Logged-in users can access a private area where they can create, update, and view policies (Figure 12).
- Light/Dark Mode: The application supports a toggle between light and dark themes for all users.



Dynamic Policy Manager

view current policies

ID	Name	Created	Phase	Status	Actions
[illegible]	cpu_usage-for-agent72	25/09/2024	enforced	activated	👁
[illegible]	memory_usage-for-agent93	25/09/2024	enforced	activated	👁
[illegible]	disk_io-for-agent94	25/09/2024	enforced	activated	👁
[illegible]	network_latency-for-agent95	25/09/2024	enforced	activated	👁
[illegible]	cpu_usage-for-agent72	15/10/2024	enforced	activated	👁
[illegible]	cpu_temperature-for-agent74	15/10/2024	enforced	activated	👁

Please log in to create a policy.

You can still view existing policies as a guest.

Figure 11: Guest mode

The other features implemented are about the policy management, specifically:

- Display a list of all policies.
- Create new policies.
- Edit policies through a dedicated edit screen (accessed via the edit icon).
- Delete policies with a confirmation dialog that appears when the delete icon is clicked.
- Not Found Page: a custom error page is shown if the user tries to access a non-existent link.

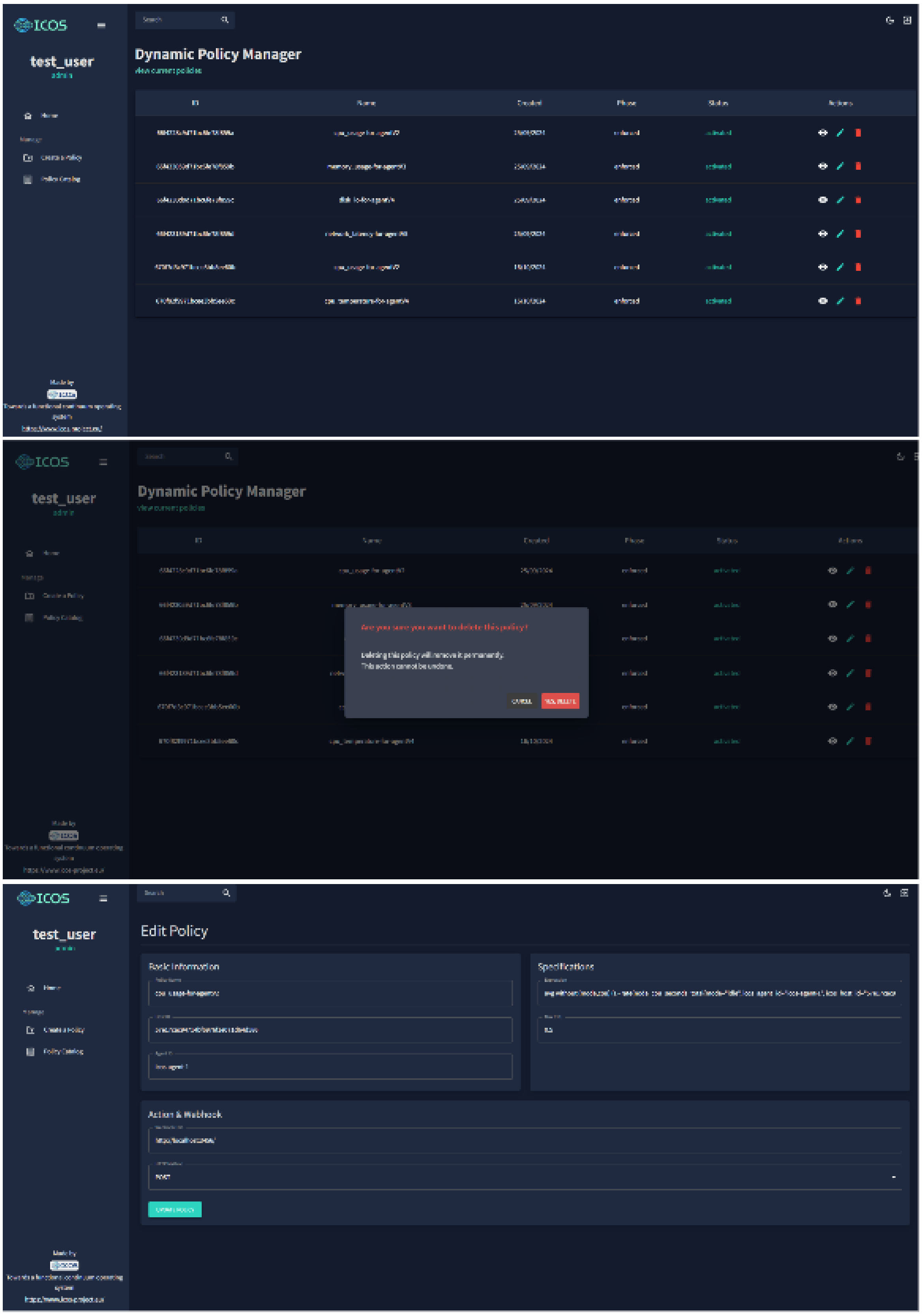

ICOS
test_user
admin
Dynamic Policy Manager
Are you sure you want to delete this policy?
Deleting this policy will remove it permanently.
This action cannot be undone.
Edit Policy
Basic Information
Specifications
Action & Webhook

Figure 12: User mode

At the moment of creating this deliverable we are finalizing the access for the policy manager services, and the basic functionalities as create and delete policy. The planned features for the next releases are:

- Full policy deletion functionality.
- Policy updates.
- Activation/deactivation of policies.
- Security improvements.
- Search component functionality.

3.3.4 Documentation

The official ICOS tehcnical documentation includes a dedicated section that shows and explains the features of the Policy Manager. This documentation will help the technical user and allow him to understand and work with this component. This documentation is available at the following URL: https://www.icos-project.eu/docs/Developer/Components/dynamic-policy-manager/

4 Conclusions

The ICOS project successfully concluded the second iteration of development and integration of the ICOS Meta-Kernel Layer Module. This document describes Meta-Kernel Layer Module IT-2 and manifests an important milestone on the progress this project has experienced since the previous IT-1.

The progress on the components that have undergone modifications during IT-2 development has been validated in the ICOS testbed. On this environment, the components have shown that they are ready to be integrated and they demonstrate their contribution to the overall ICOS MetaOS. This holds true for:

- The components receiving maintenance upgrades during this period.
- The modifications and new features that have been designed and implemented for IT-2.

Following up this IT-2, the ICOS project will be able to focus on finishing the Meta-Kernel Layer for IT-3 by combining all the gathered knowledge regarding the interaction of their components. The ICOS codebase will be able to reach a next step in terms of stability and the ICOS project will produce the artifacts required for validating through Use Cases as well as being ready for general public adoption.

5 References

[1] ICOS. D3.1 – Meta-Kernel Layer Module Integrated (IT-1). Skaburskas, Konstantin. 2023. https://www.icos-project.eu/files/deliverables/D3.1_Meta_Kernel_Module_IT-1_v1.0.pdf, retrieved 31/10/2024.

[2] ICOS. D2.4 – ICOS Architectural Design (IT-2), García, Jordi. 2024. https://www.icos-project.eu/files/deliverables/D2.4-architectural-design-it2.pdf, retrieved 31/10/2024.

Annexes

1.1 Job Manager API

1.1.1 Endpoints

1.1.1.1 GET /jobmanager/groups

Response:

Code	Description	Schema
200	OK	[[models.JobGroup]]
400	Bad Request	string
404	Not Found	string

1.1.1.2 PUT /jobmanager/groups

Parameters:

Name	Located in	Description	Required	Schema
JobGroup	body	JobGroup information	Yes	models.JobGroup

Response:

Code	Description	Schema
200	OK	models.JobGroup
400	Bad Request	string
404	Not Found	string

1.1.1.3 POST /jobmanager/groups

Parameters:

Name	Located in	Description	Required	Schema
application	body	Application manifest YAML	Yes	String

Response:

Code	Description	Schema
201	Created	models.JobGroup
422	Unprocessable Entity	string

1.1.1.4 PUT /jobmanager/groups/start/{group_uuid}

Parameters:

Name	Located in	Description	Required	Schema
group_uuid	path	JobGroup UUID	Yes	string

Responses:

Code	Description	Schema
200	OK	models.JobGroup
400	Bad Request	string
404	Not Found	string

1.1.1.5 PUT /jobmanager/groups/stop/{group_uuid}

Parameters:

Name	Located in	Description	Required	Schema
group_uuid	path	JobGroup UUID	Yes	string

Responses:

Code	Description	Schema
200	OK	models.JobGroup
400	Bad Request	string
404	Not Found	string

1.1.1.6 GET /jobmanager/groups/{group_uuid}

Parameters:

Name	Located in	Description	Required	Schema
group_uuid	path	JobGroup UUID	Yes	string

Responses:

Code	Description	Schema
200	OK	models.JobGroup
400	Bad Request	string
404	Not Found	string

1.1.1.7 DELETE /jobmanager/groups/{group_uuid}

Parameters:

Name	Located in	Description	Required	Schema
group_uuid	path	JobGroup UUID	Yes	string

Responses:

Code	Description	Schema
200	OK	string
400	Bad Request	string
404	Not Found	string

1.1.1.8 GET /jobmanager/jobs

Responses:

Code	Description	Schema
200	OK	[[models.Job]]
404	Can not find Jobs	string

1.1.1.9 PUT /jobmanager/jobs

Parameters:

Name	Located in	Description	Required	Schema
job_uuid	path	Job UUID	Yes	string
Job	body	Job information	Yes	models.Job

Responses:

Code	Description	Schema
200	OK	models.Job
400	Error reading request body	string
404	Could not find job	string

1.1.1.10 GET /jobmanager/jobs/executable/{orchestrator}?{icos_agent_id}

Parameters:

Name	Located in	Description	Required	Schema
orchestrator	path	Orchestrator type [ocm | nuvla]	Yes	string
icos_agent_id	query	ICOS Agent ID	No	string

Responses:

Code	Description	Schema
200	OK	[models.Job]
400	Orchestrator type is required	string
404	Cannot find executable Jobs	string
500	Internal server error	string

1.1.1.11 PATCH /jobmanager/jobs/promote/{job_uuid}

Parameters:

Name	Located in	Description	Required	Schema
job_uuid	path	Job UUID	Yes	string

Responses:

Code	Description	Schema
204	Job Promoted	string
400	Job UUID is required	string
404	Can not find Job to promote	string

1.1.1.12 GET /jobmanager/jobs/{job_uuid}

Parameters:

Name	Located in	Description	Required	Schema
job_uuid	path	Job UUID	Yes	string

Responses:

Code	Description	Schema
200	Ok	models.Job
400	Job UUID is required	string
404	Can not find Job by UUID	string

1.1.1.13 DELETE /jobmanager/jobs/{job_uuid}

Parameters:

Name	Located in	Description	Required	Schema
job_uuid	path	Job UUID	Yes	string

Responses:

Code	Description	Schema
200	Ok	string
400	Job UUID is required	string
404	Can not find Job to delete	string

1.1.1.14 POST /jobmanager/policies/incompliance

Parameters:

Name	Located in	Description	Required	Schema
application	body	Remediation Object	Yes	string

Responses:

Code	Description	Schema
200	OK	models.Remediation
400	Remediation Object is not correct	string
422	Unprocessable Entity	string

1.1.1.15 PUT /jobmanager/resources/status

Parameters:

Name	Located in	Description	Required	Schema
id	path	Resource UUID	Yes	string
resource	body	Resource info	Yes	models.Resource

Responses:

Code	Description	Schema
200	Resource updated	string
400	Resource UUID is required	string
404	Can not find Resource to update	string

1.1.1.16 GET /jobmanager/resources/status/{job_uuid}

Parameters:

Name	Located in	Description	Required	Schema
job_uuid	path	Job UUID	Yes	string

Responses:

Code	Description	Schema
200	OK	models.Resource
400	Job UUID is required	string
404	Can not find Job by UUID	string

1.1.2 Models

1.1.2.1 models.Condition

Name	Type	Description	Required
lastTransitionTime	string		Yes
message	string		Yes
observedGeneration	integer		No
reason	string		Yes
status	models.ConditionStatus		Yes
type	models.ResourceState		Yes

1.1.2.2 models.ConditionStatus

Name	Type	Description	Required
models.ConditionStatus	string		

1.1.2.3 models.Content

Name	Type	Description	Required
name	string		Yes
yaml	string		Yes

1.1.2.4 models.Instruction

Name	Type	Description	Required
componentName	string		No
contents	[models.Content]		No
id	string		No
job_id	string	Excluded from YAML	No
policies	[models.Policy]		No
requirements	models.Requirement		No
type	string		No

1.1.2.5 models.Job

Name	Type	Description	Required
id	string		No
instruction	models.Instruction		No
job_group_id	string		No
namespace	string		No
orchestrator	models.OrchestratorType		No
owner_id	string		No
resource	models.Resource		No
state	models.JobState		No
sub_type	models.RemediationType		No
targets	models.Target		No
type	models.JobType		No

1.1.2.6 models.JobGroup

Name	Type	Description	Required
appDescription	string		No
appName	string		No
id	string		No
jobs	[models.Job]		Yes

1.1.2.7 models.JobState

Name	Type	Description	Required
models.JobState	string		

1.1.2.8 models.JobType

Name	Type	Description	Required
models.JobType	string		

1.1.2.9 models.OrchestratorType

Name	Type	Description	Required
models.OrchestratorType	string		

1.1.2.10 models.Policy

Name	Type	Description	Required
component	string		No
fromTemplate	string		No
name	string		No
remediation	string		No
spec	models.PolicySpec		No
variables	models.PolicyVariables		No

1.1.2.11 models.PolicySpec

Name	Type	Description	Required
expr	string		No
thresholds	models.Thresholds		No

1.1.2.12 models.PolicyVariables

Name	Type	Description	Required
compssTask	string		No
thresholdTimeSeconds	integer		No

1.1.2.13 models.Remediation

Name	Type	Description	Required
id	string		No
remediationStatus	models.RemediationStatus		Yes
remediationTarget	models.RemediationTarget		No
remediationType	models.RemediationType		Yes
resource_id	string		No

1.1.2.14 models.RemediationStatus

Name	Type	Description	Required
models.RemediationStatus	string		

1.1.2.15 models.RemediationTarget

Name	Type	Description	Required
command	string		No
container	string		No
id	string		No
namespace	string		No
node	string		No
pod	string		No
pod_uid	string		No
remediation_id	string		No

1.1.2.16 models.RemediationType

Name	Type	Description	Required
models.RemediationType	string		

1.1.2.17 models.Requirement

Name	Type	Description	Required
architecture	string		No
cpu	string		No
devices	string		No
memory	string		No

1.1.2.18 models.Resource

Name	Type	Description	Required
conditions	[models.Condition]	OriginalResourceName string gorm:"type:text" json:"-" validate:"omitempty"	No
job_id	string		No
remediations	[models.Remediation]		No
resource_name	string		No
resource_uuid	string	we set this field manually	No

1.1.2.19 models.ResourceState

Name	Type	Description	Required
models.ResourceState	string		

1.1.2.20 models.Target

Name	Type	Description	Required
cluster_name	string		Yes
node_name	string		No
orchestrator	models.OrchestratorType		Yes

1.1.2.21 models.Thresholds

Name	Type	Description	Required
critical	integer		No
warning	integer		No

1.2 Policy Model JSON

```
{
    "name": "string",
    "subject": {
        "type": "app",
        "appName": "string",
        "appComponent": "string",
        "appInstance": "string"
    },
    "spec": {
        "description": "",
        "type": "template",
        "templateName": "string"
    },
    "action": {
        "type": "webhook",
        "url": "string",
        "httpMethod": "CONNECT",
        "extraParams": {},
        "includeAccessToken": false
    },
```

An example of an actual valid policy:

```
{
    "name": "cpu_usage-for-agent",
    "subject": {
        "type": "host",
        "hostId": "57e17cac94714bf6976f1e071d64d586",
        "agentId": "icos-agent-1"
    },
    "spec": {
        "description": "",
        "type": "telemetryQuery",
        "expr": "avg without (...) > 0.5",
        "violatedIf": None,
        "thresholds": None
    },
    "action": {
        "type": "webhook",
        "url": "https://localhost:3246/",
        "httpMethod": "POST",
        "extraParams": {},
        "includeAccessToken": False
    },
    "variables": {
```

1.3 Policy Manager API

1.3.1 Endpoints

1.3.1.1 GET/registry/api/v1/policies/{id}

Parameters: an authorization access token is requested as String

Request body is provided from the following schema;

Attribute Name	Type	Description
id	Long	Unique identifier of the deployment

1.3.1.2 GET /registry/api/v1/policies/

Parameters: an authorization access token is requested as String.

No schema is provided.

1.3.1.3 POST/registry/ap1/v1/policies/{policy}

Parameters: an authorization access token is requested as String.

Request body is provided from the following schema;

Attribute Name	Type	Description
subject	Subject[]	Provides a set of information about the type of application, name of the application, component and instance
spec	Spec[]	Set of information about the template used
action	Action[]	Set of information about the action to provided. [DEFAULT] is “webhook”.
variables	Variables[]	[OPTIONAL] addtional properties. It can a string, an integer or a number. [DEFAULT] is {}
properties	Properties[]	[OPTIONAL] additional properties [DEFAULT] is {}

Subject[]

Attribute Name	Type	Description
type	const	Default is “app”
appName	String	Name of the app
appComponent	String	Component of the app
appInstance	String	Instance of the app

Spec[]

Attribute Name	Type	Description
description	String	[DEFAULT] is empty “”
type	Const	[DEFAULT] is “template”
templateName	String	Name of the template

Action[]

Attribute Name	Type	Description
url	String	Link to the alermanager
type	Const	[DEFAULT] is “webhook”
httpMethod	String	[DEFAULT] is “CONNECT”
extraParams	[]	Additional properties. [DEFAULT] is empty {}
includeAccessToken	boolean	Token to acces to the webhook, default is FALSE

Http methods from the following RFCs are all observed:

- RFC 7231: Hypertext Transfer Protocol (HTTP/1.1), obsoletes 2616
- RFC 5789: PATCH Method for HTTP

Allowed values are : "CONNECT", "DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT", "TRACE".

Variables[]

Attribute Name	Type	Description
#0	String	[OPTIONAL]
#1	integer	[OPTIONAL]
#2	number	[OPTIONAL]

Properties[]

Attribute Name	Type	Description
oneoff	boolean	boolean
interval	string	Interval where the policy should be acts
pendingInterval	String	Information about the interval status

1.3.1.4 POST/registry/api/v1/icos/

Parameters: authorization access token

Request body is provided from the following schema;

Attribute Name	Type	Description
app_descriptor	App_descriptor[]	Object that provided a set of information.
app_instance	String	Instance name of the app descriptor.
common_action	Common_action[]	Object that provides a set of information about the service, specifically for the icos-service.
service	String	name of the service

app_descriptor[]

Attribute Name	Type	Description
name	String	Name of the app descripton
description	String	[DEFAULT] is “”.
components	Component[]	Object that providesa a set of information about the compoennts
policies	Policies[]	Object that provided a set of information about the policies

Component[]

Attribute Name	Type	Description
name	Name of the component	Object that provided a set of information.
type	String	Type of the component
policies	policies[]	Array Object that providesa a set of information about the policies for the icos service.. [DEFAULT] is empty : []

Policies[] :

Attribute Name	Type	Description
name	string	Name of the policies
component	string	It can be Null
fromTemplate	string	It can be null
spec	Spec[]	Object provides a set of information about policies template, telemetry and constraints
remediation	string	It can be null
variables	Variables[]	[DEFAULT] is {}
properties	Properties[]	[DEFAULT] is {}

common_action[]

Attribute Name	Type	Description
uri	String	Link to the alermanager
type	Const	[DEFAULT] is “icos-service”
httpMethod	String	[DEFAULT] is “CONNECT”
extraParams	[]	Additional properties. [DEFAULT] is empty {}
includeAccessToken	boolean	Token to acces to the webhook, default is FALSE

1.3.1.5 POST/watcher/api/v1/webhooks/alertmanager

Parameters: access token

Attribute Name	Type	Description
version	String	
groupKey	string	
truncatedAlerts	0	
status	String	
receiver	String	
groupLabels	GroupLabels[]	[DEFAULT] is {}
commonLabels	CommonLabels[]	DEFAULT] is {}
commonAnnotations	CommonAnnotations[]	DEFAULT] is {}
externalURL	string	
alerts	AlertsArray<object>	Array Object provides items as status, label, ...

Grouplabels[]

Attribute Name	Type	Description
Additional properties	String	[DEFAULT] : {}

CommonLabels[]

Attribute Name	Type	Description
Additional properties	String	[DEFAULT] : {}

CommonAnnotations[]

Attribute Name	Type	Description
Additional properties	String	[DEFAULT] : {}

Alerts[]

Attribute Name	Type	Description
Items	Items[]	

Items []

Attribute Name	Type	Description
status	String	It can be null
labels	Labels[]	Additional properties
annotations	Annotations[]	Additional properties
startsAt	String	It provides the date-times when the policies started.
endsAt	String	It provides the date-time when the policies ended.
generatorUrl	String	Url of the generator
fingerprint	String	It can be null.

Labels

Attribute Name	Type	Description
Additional properties	String	[DEFAULT] : {}

Annotations[]

Attribute Name	Type	Description
Additional properties	String	[DEFAULT] : {}

1.3.1.6 GET/status/

Parameters: access token