



## D3.1 Meta-Kernel Layer Module Integrated (IT-1)

Document Identification			
<b>Status</b>	Final	<b>Due Date</b>	30/09/2023
<b>Version</b>	1.0	<b>Submission Date</b>	20/10/2023

<b>Related WP</b>	WP3	<b>Document Reference</b>	D3.1
<b>Related Deliverable(s)</b>	D2.1, D2.2	<b>Dissemination Level (*)</b>	PU
<b>Lead Participant</b>	SIXSQ	<b>Lead Author</b>	Konstantin Skaburskas (SixSq)
<b>Contributors</b>	ATOS, NCSR, PSNC, L-PIT, SUITE5, XLAB, ENG, UPC, BSC, TUBS, NKUA, CRF	<b>Reviewers</b>	Sebastian Cajas Ordoñez (CeADAR) Jaydeep Samanta (CeADAR) Nikos Dimitriou (NCSR)

<b>Keywords:</b>
Cloud, Edge, IoT, MetaOS, development, integration

This document is issued within the frame and for the purpose of the ICOS project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101070177. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

The dissemination of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This deliverable is subject to final acceptance by the European Commission.

This document and its content are the property of the ICOS Consortium. The content of all or parts of this document can be used and distributed provided that the ICOS project and the document are properly referenced.

Each ICOS Partner may use this document in conformity with the ICOS Consortium Grant Agreement provisions.

(\*) Dissemination level: **(PU)** Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page). **(SEN)** Sensitive, limited under the conditions of the Grant Agreement. **(Classified EU-R)** EU RESTRICTED under the Commission Decision No2015/444. **(Classified EU-C)** EU CONFIDENTIAL under the Commission Decision No2015/444. **(Classified EU-S)** EU SECRET under the Commission Decision No2015/444.

## Document Information

---

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

Document History			
Version	Date	Change editors	Changes
0.1	28/06/2023	SIXSQ	First draft version of ToC
0.2	18/10/2023	SIXSQ	Main contributions from all the partners.
1.0	20/10/2023	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Konstantin Skaburskas (SixSq)	18/10/2023
Quality manager	Carmen San Román	20/10/2023
Project Coordinator	Francesco D'Andria	20/10/2023

# Table of Contents

---

Document Information .....	2
Table of Contents .....	4
List of Tables.....	6
List of Figures .....	7
List of Acronyms.....	8
Executive Summary .....	9
1 Introduction .....	10
1.1 Purpose of the document.....	10
1.2 Relation to other project work.....	10
1.3 Structure of the document .....	10
2 Meta-Kernel Layer Module Design.....	11
2.1 Design considerations .....	11
2.2 Components and interfaces .....	14
2.2.1 user.shell .....	16
2.2.2 user.shell-backend .....	16
2.2.3 security.IAM.....	17
2.2.4 discovery.lighthouse .....	17
2.2.5 runtime.job-manager.....	18
2.2.6 runtime.deployment-manager .....	18
2.2.7 runtime.matchmaker .....	19
2.2.8 runtime.execution-manager .....	20
2.2.9 runtime.execution-manager.offloader.....	20
2.2.10 continuum.aggregator .....	21
2.2.11 continuum.orchestrator-edge-cloud .....	22
2.2.12 cloud.infrastructure-service .....	22
2.2.13 edge.node .....	22
2.2.14 telemetry.agent .....	23
2.2.15 telemetry.controller.....	23
2.3 Detailed design.....	24
2.3.1 Node On-Boarding.....	24
2.3.2 Basic Application Deployment.....	25
2.3.3 Collect and visualise Metrics and Logs .....	26
3 Technology and Tools Selection for Implementation .....	27
3.1 Lighthouse.....	27
3.2 Continuum Manager .....	28

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	4 of 79	
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

3.2.1	Resource and Clustering Manager.....	28
3.2.2	Dynamic Policies Manager.....	34
3.3	Runtime Manager.....	37
3.3.1	Job Management.....	37
3.3.2	Matchmaking.....	37
3.3.3	Aggregator.....	37
3.3.4	Distributed & Parallel Execution.....	38
3.3.5	Workload Offloader.....	40
3.4	Logging and Telemetry.....	40
3.4.1	Metrics Collected.....	41
4	Development and Validation.....	46
4.1	Development and Integration Processes and Tools.....	46
4.2	Validation.....	48
4.2.1	Node On-Boarding.....	48
4.2.2	Application Descriptor Definition and Basic Application Deployment.....	54
4.2.3	Distributed and Parallel Execution.....	57
4.2.4	Collect and visualise Metrics and Logs.....	59
5	Results and next steps.....	63
6	Annexes.....	64
6.1	RESTful API resource schemas and examples.....	64
6.1.1	user.shell-backend-i.....	64
6.1.2	runtime.job-manager-i.....	64
6.1.3	runtime.matchmaker-i.....	67
6.1.4	runtime.execution-manager-i.....	70
6.1.5	continuum.aggregator-i.....	72

## List of Tables

---

*Table 1 Summary of the design components and their development and integration state for IT-1. \_\_\_\_\_ 15*  
*Table 2 ICOS Meta-Kernel components, responsible partner, and selected technology. \_\_\_\_\_ 47*

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	6 of 79				
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final

## List of Figures

---

Figure 2-1 ICOS Meta-Kernel conceptual architecture from D2.2	11
Figure 2-2 ICOS Meta-Kernel and related ICOS ecosystem components considered for IT-1.	14
Figure 2-3 ICOS Meta-Kernel Layer component design for IT-1: functional layers, design components, and their interfaces. In blue are the software components and interfaces developed for IT-1. Legend: s. - security; d. - discovery; r. - runtime; c. - continuum; t. - telemetry.	16
Figure 2-4 Sequence diagram of the deployment management used in IT-1.	19
Figure 2-5 Meta-Kernel layer for IT-1: Basic Application Deployment.	25
Figure 2-6 Interaction between ICOS components to collect and visualise telemetry data.	26
Figure 3-1 Controller registration with one domain	27
Figure 3-2 Controller registration with two domains	28
Figure 3-3 Nuvla: Onboarding of Edge device - creation of NuvlaEdge in Nuvla.	29
Figure 3-4 Nuvla: Onboarding of Edge device – helm command for deployment of NuvlaEdge on Edge device.	30
Figure 3-5 Nuvla: Onboarding of Edge device – Edge device is operational.	30
Figure 3-6 NuvlaEdge conceptual architecture: components, relation to Nuvla.io, and management actions.	31
Figure 3-7 Nuvla: Discovered peripherals on edge device.	32
Figure 3-8 Sequence diagram of onboarding of a Kubernetes cluster to OCM.	34
Figure 3-9 Initial architecture of the Dynamic Policies Manager.	35
Figure 3-10 High-level architecture of the Aggregator service.	38
Figure 3-11 Composition of the technologies used for the implementation of the Logging and Telemetry Layer.	40
Figure 3-12 Developed stack of Scaphandre, Prometheus, and Grafana, as implemented at the NKUA lab.	43
Figure 3-13 Recorded energy consumption curve of a node.	44
Figure 3-14 Query parameters to request 'scaph_process_power_consumption_microwatts' metric for specific pods.	44
Figure 3-15 JSON response with 'scaph_process_power_consumption_microwatts' metric.	45
Figure 4-1 Snapshot of the ICOS System Release schedule around IT-1 (source D2.2).	46
Figure 4-2 ICOS Project source code repository using GitLab.	47

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	7 of 79	
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

## List of Acronyms

---

Abbreviation / acronym	Description
COE	Container Orchestration Engine
Dx.y	Deliverable number y belonging to WP x
WP	Work Package
SUC	System Use Cases
SLO	Service Level Objective
SLI	Service Level Indicator
SRE	Site Reliability Engineering
MM	Matchmaking service
JM	Job Manager
OCM	Open Cluster Management
VM	Virtual Machine
OTLP	OpenTelemetry protocol



## Executive Summary

---

This document, titled "D3.1 Meta-Kernel Layer Module Integrated (IT-1)," serves as a comprehensive report on the design, development, and implementation of the ICOS Meta-Kernel Layer Module for the IT-1 delivery phase. 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-1), 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. To accomplish the development of the Meta-Kernel Layer Module, the conducted work leverages the outcomes of D2.2 "ICOS Architecture Design IT-1" and D2.1 "ICOS ecosystem: Technologies, requirements and state of the art". It incorporates the architectural insights from D2.2 and requirements from D2.1, including the definition of the ICOS Meta-Kernel Layer, its core functions, and how it interfaces with other layers within the ICOS Meta Operating System. The development work and the document build upon the foundation laid by D2.2, expanding on the conceptual design with concrete implementations.

This 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-1. It encapsulates the collective efforts of the project team in realising a fundamental component of the ICOS Meta Operating System, bringing the project one step closer to its ultimate objectives.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)			<b>Page:</b>	9 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0
				<b>Status:</b>	Final

# 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 for IT-1 delivery..

## 1.2 Relation to other project work

---

---

This document uses results of D2.2 “ICOS Architecture Design IT-1”. 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.

## 1.3 Structure of the document

---

---

In section 2 “Meta-Kernel Layer Module Design”, we first present the System User Stories from D2.2 as generic targets for ICOS Alpha release and then, discuss and present the actual scope of the implementation for IT-1 delivery of the Meta-Kernel Layer. We then proceed with the detailed design of the components and their interfaces selected for the implementation and integration. Section 3, “Technology and Tools Selection for Implementation”, elaborates on the selection of the tools for the implementation of the components that are part of the IT-1. Then, section 5, “Development and Validation”, first describes the development and integration processes selected for IT-1 and then presents the IT-1 validation Test Cases and their results.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	10 of 79				
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final

## 2 Meta-Kernel Layer Module Design

As shown in Figure 2-1, and detailed in D2.2, the Meta-Kernel layer is part of the ICOS system and serves the following main functional purposes: management of the Cloud-Edge-IoT continuum regarding resource and applications, management of the runtime of the applications, and collection and storage of logging and telemetry from the managed resources and applications.

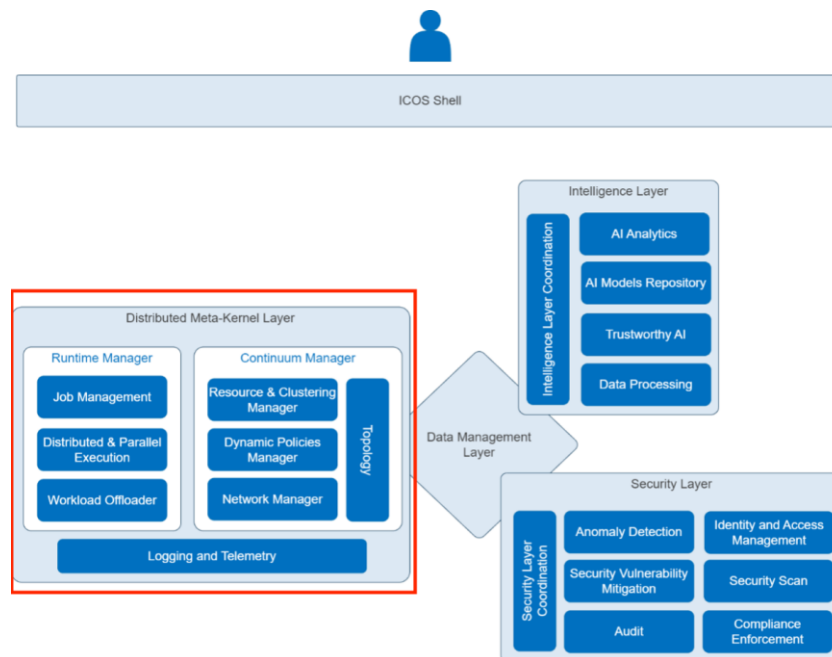


Figure 2-1 ICOS Meta-Kernel conceptual architecture from D2.2

This section:

- ▶ Defines the scope of the functionalities implemented for IT-1 delivery.
- ▶ Provides concrete design considerations and limitations for IT-1 delivery.
- ▶ Lists design components and interfaces.
- ▶ Provides the detailed design of IT-1.

### 2.1 Design considerations

This section presents the design considerations and limitations for IT-1 delivery.

Following D2.2 deliverable section 6.3 Table 9: “Functionalities ICOS Beta and ICOS Final releases”, the target System Use Cases for ICOS Alfa release at M15 (November 2023) are presented with an analysis below. The requirements of the System Use Cases (SUCs) that are considered to be implemented are also provided below to give a better context for understanding the considerations on the design decisions and selection of the components for the IT-1 implementation (these requirements are described in detail in Tables 6-10 of deliverable D2.1).

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	11 of 79	
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

**A. Node On-Boarding (SUC CC 1)**

a. System Use Case SUC CC 1 - Allows to add a new node to an ICOS Cloud Continuum

- i. Requirement: CC\_FR\_04 **Controller Communication:** ICOS SHOULD allow the communication of multiple ICOS controllers to exchange local views, policies and information.

**IT-1 Note 1:** IT-1 is an early stage development and integration work; bringing in cross ICOS Controller communication first requires basic ICOS Controller functionality to be in place; the work on cross ICOS Controller communication will be done after IT-1 for ICOS Beta release.

- ii. Requirement: OP\_FR\_04 **Resource Descriptor Data-model:** ICOS MUST define a data model to be used for describing each device/node resource capability in order to be accessible by the metaOS

**IT-1 Note 2:** It will be implemented as part of the external infrastructure services' resource schema of Cloud and Edge Orchestrators.

**B. Application Descriptor Definition (SUC RT 1)**

a. System Use Case SUC RT 1 - Allows the definition of an application and all the details for its deployment

- i. Requirements: NONE.

**IT-1 Note 3:** Definition of user application description for ICOS Controller is one of the primary goals for IT-1 delivery.

- ii. User Stories: US.4, US.7

**C. Basic Application Deployment (SUC RT 4)**

a. System Use Case SUC RT 4 - Allows to deploy an application in an ICOS Instance

- i. Requirement: CC\_FR\_01 **Resources Catalog:** The ICOS MUST support a resource registry to record (publish) available resources to operate application workloads.

**IT-1 Note 4:** This is implemented as part of the Aggregator service that contains all the resources of the IoT-Edge-Cloud continuum.

- ii. Requirement: CC\_FR\_03 **Topology Awareness:** ICOS should be able to monitor and maintain the topology of the created Cloud Continuum.

**IT-1 Note 5:** This is implemented as part of the Aggregator service.

- iii. Requirement: CM\_FR\_01 **Smart resources first allocation and migration:** ICOS MUST be able to find a near-optimal match (considering different metrics, such as response time, energy footprint, monetary cost) in terms of nodes to run one business application taking into account nodes performance, reliability and availability

**IT-1 Note 6:** A simple application match-making for initial placement (day-0 application deployment) at the cloud or edge is implemented for IT-1. Application migration for optimal re-provisioning will be worked on for the next milestones (ICOS Beta release).

- iv. Requirement: CM\_FR\_03 **Function execution request:** ICOS COULD provide a mechanism to request the execution of a function on the continuum being totally transparent of the device that will host the execution.

**IT-1 Note 7:** This functionality was not selected for the scope of IT-1. It will be considered for the implementation in the next milestones.

- v. Requirement: CM\_FR\_09 **Green Policies Monitoring:** ICOS MUST be able to determine when reserved resources are not used or required for proper application operation at some point in time and provide an alert system and offer Service Level Objective (SLO) modifications accordingly.

**IT-1 Note 8:** Monitoring of the edge and cloud resources (CPU/RAM/Disk/Network) is implemented in IT-1. Alerting and decision making around green policies based on the collected telemetry from the user applications and resources will be done in the scope of the next milestones.

- vi. Requirement: CM\_FR\_13 **Parallelism exploitation**: ICOS MUST provide a mechanism that allows service components to decompose application components into sub-components (or tasks) to enable massive/distributed parallel execution (and achieve lower response times and a better resource exploitation).

**IT-1 Note 9:** This was not considered as part of the IT-1 delivery. This functionality will be designed and implemented in the scope of the next milestones.

**D. Collect and visualise Metrics and Logs (SUC\_RT\_6, SUC\_RT\_7)**

- a. System Use Case SUC\_RT\_6 - Review the status and the logs of all the application components

- i. Requirement: CM\_FR\_06 **Monitoring**: ICOS MUST collect monitoring infrastructure-level and application-level metrics from various sources as well as provide appropriate solutions to preserve and access historical performance data. Collecting metrics from already existing monitoring tools on the infrastructures should be supported.

**IT-1 Note 10:** See IT-1 Note 8 regarding the metrics collection. The initial implementation for the long-term storage of the collected telemetry is provided.

- ii. Requirement: OP\_FR\_05 **Monitoring system performance**: ICOS MUST display resources and application workloads performances in real time as well as historical performance data within a graphical view. This view should allow the service operator to modify the data to display.

**IT-1 Note 11:** See **IT-1 Note 10**.

- b. System Use Case SUC\_RT\_7 - See and query the performance metrics related to the resources that are hosting the application and the metrics generated by the application itself

- i. Requirement: CM\_FR\_06 (see above)
- ii. Requirement: CM\_FR\_07 **SLO monitor to raise a corrective plan**: ICOS MUST monitor application workload SLO to raise remedial plan as well as corrective actions when QoS levels are violated.

**IT-1 Note 12:** SLO monitoring of application workload was not considered for the IT-1 delivery. This topic requires more research and solution-level design for coupling of the SLO definition policies, metrics and successive generation of the remedial plan and effectively, the corrective actions.

The **IT-1 notes** provided along with the SUCs and associated requirements (**IT-1 Note 1-12**), define the scope for the functionalities of IT-1 delivery. Based on that, we define the following constraints:

- ▶ **Constraint A:** The IT-1 delivery supports only a single ICOS Controller (**IT-1 Note 1**).
- ▶ **Constraint B:** The IT-1 delivery doesn't support application migration (**IT-1 Note 6**).
- ▶ **Constraint C:** The IT-1 delivery doesn't contain FaaS (**IT-1 Note 7**).
- ▶ **Constraint D:** The IT-1 delivery doesn't contain alerting and decision making around green policies (**IT-1 Note 8**).
- ▶ **Constraint E:** The IT-1 delivery doesn't contain application parallelism exploitation (**IT-1 Note 9**).
- ▶ **Constraint F:** The IT-1 delivery doesn't contain application SLO monitoring and hence Policy Manager component (**IT-1 Note 12**).

Taking into account the considerations on the system use cases and respectively derived constraints, below we present the ICOS component level architecture implemented for IT-1 as part of Meta-Kernel Layer.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	13 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

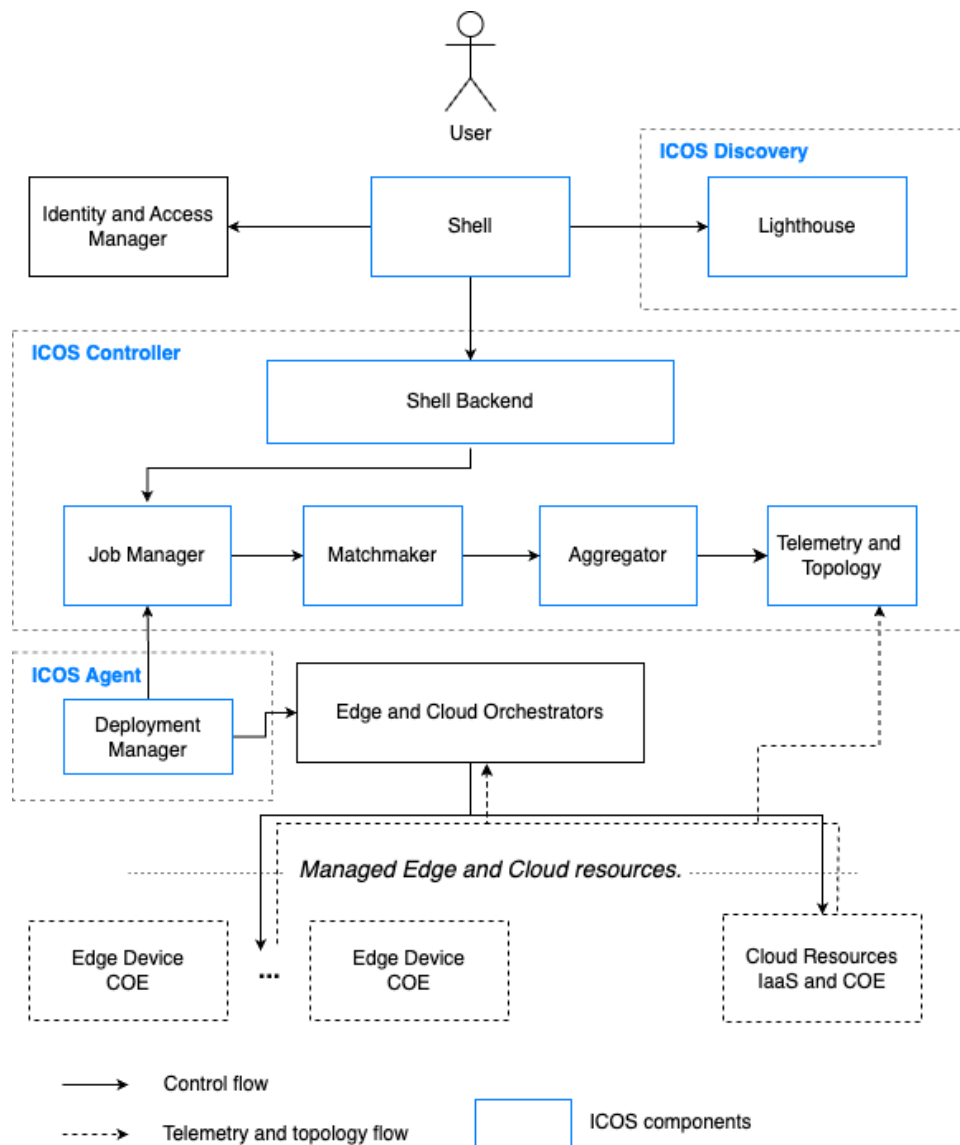


Figure 2-2 ICOS Meta-Kernel and related ICOS ecosystem components considered for IT-1.

In the following section we define the design components and their respective interfaces. Only components that are part of the ICOS IT-1 delivery are considered.

## 2.2 Components and interfaces

This section lists and describes the critical design components and interfaces that were implemented and integrated for IT-1 delivery.

Below is the mapping of the component names from the architecture diagram to the sub-domain specific component design naming:

- ▶ Identity and Access Manager - *security.IAM*
- ▶ Shell - *user.shell*
- ▶ Lighthouse - *discovery.lighthouse*

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	14 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

- ▶ ICOS Controller
  - Shell Backend - `user.shell-backend`
  - Job Manager - `runtime.job-manager`
  - Matchmaker - `runtime.matchmaker`
  - Execution Manager - `runtime.execution-manager`
  - Execution Manager Offloader - `runtime.execution-manager.offloader`
  - Aggregator - `continuum.aggregator`
- ▶ ICOS Agent
  - Deployment Manager - `runtime.deployment-manager`
- ▶ Edge and Cloud Orchestrator - `continuum.orchestrator-edge-cloud`
- ▶ Edge Agent - `continuum.edge-agent`
- ▶ Logging and Telemetry
  - Telemetry Agent - `telemetry.agent`
  - Telemetry Controller - `telemetry.controller`

The table below gives an overview of the new components that were developed, as well as of the existing ones that were employed and of the ones that were integrated as part of the Meta-Kernel Layer. In cases where an existing tool was used for integration a name and the reference are provided.

Table 1 Summary of the design components and their development and integration state for IT-1.

Component	Developed	Integrated
<code>user.shell</code>	+	+
<code>user.shell-backend</code>	+	+
<code>security.IAM</code>	not applicable (KeyCloak <sup>1</sup> )	-
<code>discovery.lighthouse</code>	+	+
<code>runtime.job-manager</code>	+	+
<code>runtime.deployment-manager</code>	+	+
<code>runtime.execution-manager</code>	extended (COMPSs <sup>2</sup> )	<b>(Constraint E)</b>
<code>runtime.execution-manager.offloader</code>	extended (COMPSs)	<b>(Constraint E)</b>
<code>runtime.matchmaker</code>	+	+
<code>continuum.aggregator</code>	+	+
<code>continuum.orchestrator-edge-cloud</code>	not applicable (OCM <sup>3</sup> , Nuvla <sup>4</sup> )	+
<code>continuum.edge-agent</code>	not applicable (OCM klusterlet <sup>5</sup> , NuvlaEdge <sup>6</sup> )	+
<code>telemetry.agent</code>	extended (OpenTelemetry Collector, <sup>7</sup> Prometheus Node Exporter <sup>8</sup> )	+
<code>telemetry.controller</code>	not applicable (Thanos <sup>9</sup> , Grafana <sup>10</sup> )	+

<sup>1</sup> <https://www.keycloak.org>

<sup>2</sup> <https://compss.bsc.es>

<sup>3</sup> <https://open-cluster-management.io>

<sup>4</sup> <https://nuvla.io>

<sup>5</sup> <https://open-cluster-management.io/concepts/architecture/>

<sup>6</sup> <https://github.com/nuvlaedge>

<sup>7</sup> <https://opentelemetry.io/docs/collector/>

<sup>8</sup> [https://github.com/prometheus/node\\_exporter](https://github.com/prometheus/node_exporter)

<sup>9</sup> <https://thanos.io/>

<sup>10</sup> <https://grafana.com/>

Following the table above Table 1, the picture below presents the design components and their interfaces of ICOS Continuum that are part of IT-1 delivery. The design components and their provided and required interfaces are described in the subsections that follow.

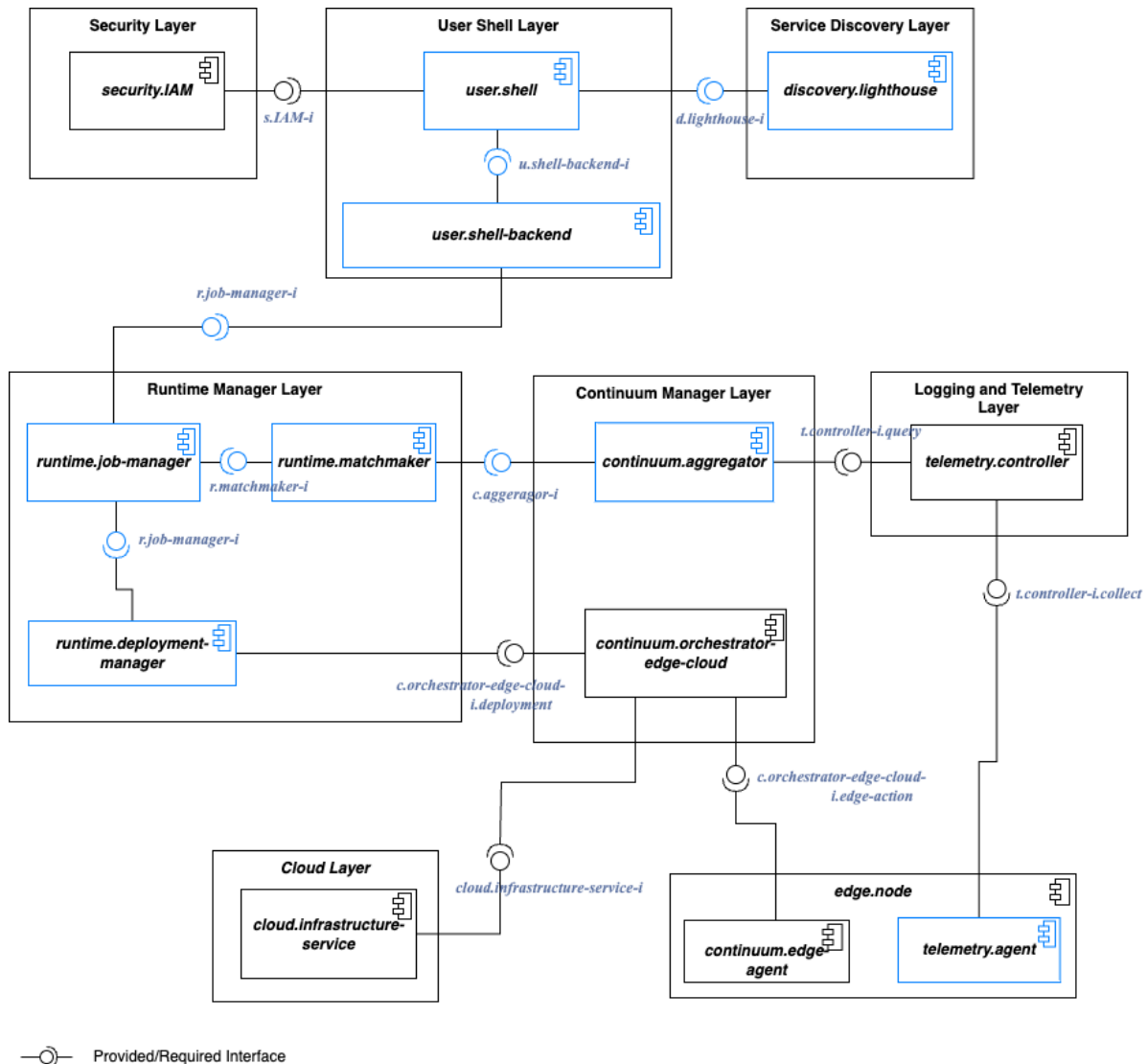


Figure 2-3 ICOS Meta-Kernel Layer component design for IT-1: functional layers, design components, and their interfaces. In blue are the software components and interfaces developed for IT-1. Legend: s - security; d - discovery; r - runtime; c - continuum; t - telemetry.

### 2.2.1 user.shell

Command line interface for end-user and Cloud/Edge operator to access the ICOS ecosystem. Connects to `user.shell-backend [TUBS]` via required interface `user.shell-backend-i`.

### 2.2.2 user.shell-backend

API server providing a single entry point to the ICOS ecosystem. Contacted by `user.shell [TUBS]` on `user.shell-backend-i` interface.



### 2.2.2.1 user.shell-backend-i

Provided RESTful API exposing the resources and actions described below.

#### API Endpoint

All URIs are relative to `http://${CONTROLLER_ADDRESS}/api/v3`

HTTP request	Description
GET /healthcheck	Health check
POST /deployment/	Creates a new deployment
DELETE /deployment/{deploymentId}	Deletes a deployment by ID
GET /deployment/{deploymentId}	Get deployment by ID
GET /deployment/	Returns a list of deployments visible to the concrete user
PUT /deployment/{deploymentId}	Updates a deployment
GET /resource/{resourceId}	Get resource by ID
GET /resource/	Returns a list of resources

The schemas of the resources are provided in the Appendix section **6.1.1**.

### 2.2.3 security.IAM

The Identity and Access Management component is the component in the security layer (WP4) responsible for the authentication and the authorization of users in the ICOS System. The component exposes an API to trigger authentication flows, request and validate authentication tokens. A detailed description of the component and the APIs exposed is provided in deliverable “D4.1 Data Management, Intelligence and Security Layers (IT-1)”. In IT-1, the ICOS Shell and the Lighthouse components have been integrated with this component to authenticate all the requests from the users.

#### 2.2.3.1 security.IAM-i

Provided RESTful API exposing identity and access management functionalities. Implementation specific. For concrete selected technology and tools see section [Technology and Tools Selection for Implementation \[ALL\]](#).

### 2.2.4 discovery.lighthouse

This service maintains a list of all ICOS Controllers currently part of the ICOS continuum. This list can be retrieved by any component that needs to make contact with a controller; namely the ICOS [user.shell \[TUBS\]](#) for configuration through an instance of the ICOS [user.shell-backend \[TUBS\]](#) and the ICOS Agent to find a controller through which to join the ICOS Continuum.

#### 2.2.4.1 discovery.lighthouse-i

Provided RESTful API exposing the resources and actions described below.

#### API Endpoint

All URIs are relative to `http://${LIGHTHOUSE_ADDRESS}/api/v3`

HTTP request	Description
POST /controller/	Adds a new controller
GET /controller/	Returns a list of controllers

The schemas of the resources are provided below.

### /controller/ resource data model

GET returns list of the maps of the following schema

Attribute Name	Type	Description
name	String	Name of the controller
address	String	IP address of the controller

### 2.2.5 runtime.job-manager

This service enables ICOS Controller to manage and offload the workload based on the incoming application description and placement information from the [runtime.matchmaker \[UPC\]](#).

#### 2.2.5.1 runtime.job-manager-i

Provided RESTful API exposing the resources and actions described below.

### API Endpoints

All URIs are relative to [http://\\${CONTROLLER\\_ADDRESS}](http://${CONTROLLER_ADDRESS})

HTTP request	Description
GET /jobmanager/healthcheck	Health check
POST /jobmanager/jobs/create	Creates a new Job with state “Created”
DELETE /deployment/jobs/{deploymentId}	Deletes a Job by ID
GET /deployment/jobs/{deploymentId}	Finds a Job by its ID
PUT /deployment/jobs/{deploymentId}	Updates a Job
GET /deployment/jobs/	Returns a list of Jobs
GET /deployment/jobs/executable	List Jobs that can be executed

### API Schemas

The schemas of the resources are provided in Appendix section **6.1.2**.

### 2.2.6 runtime.deployment-manager

This service enables ICOS Controller to start user applications on the resource managed by *continuum.orchestrator-edge-cloud*. The service doesn't export any HTTP endpoints. The service consumes *runtime.job-manager-i* from *runtime.job-manager* component.

Below we present the sequence diagram that shows the deployment management workflow between the corresponding design components. For any technology and instances of *continuum.orchestrator-edge-cloud*, the pulled jobs are handled as described in the diagram.

## Deployment Management

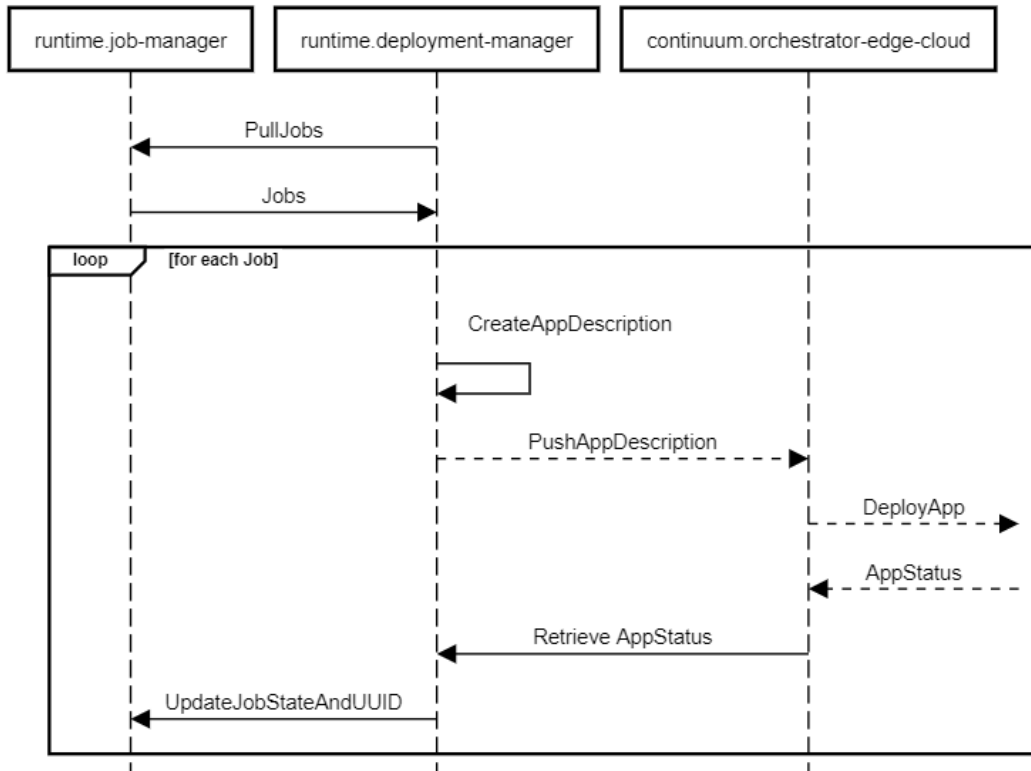


Figure 2-4 Sequence diagram of the deployment management used in IT-1.

Note: for a Job to be pullable/executable by the *runtime.deployment-manager*, this job must comply with the following rule:

- ▶ Job.State is “Created” and Job.Locker is False or Job.State is “Progressing” and Job.Locker is True and Job.UpdatedAt is less than 300 seconds. This rule is validated by the *runtime.job-manager*.

### 2.2.7 runtime.matchmaker

It receives an application descriptor and requests the continuum.aggregator the ICOS topology. The service finds the most appropriate nodes of the edge-to-cloud continuum to execute the requested user application on It receives the user application description and requests the [continuum.aggregator](#) the ICOS topology. The service is invoked from the [runtime.job-manager](#) through the provided interface [runtime.matchmaker-i](#).

#### 2.2.7.1 runtime.matchmaker-i

Provided RESTful API exposing the resources and actions described below.

#### API Endpoints

All URIs are relative to `http://${MATCHMAKER_ADDRESS}`

HTTP request	Description
GET /	Displays a welcome message.
POST /matchmake	Receive a JSON with the description of the application. JSON response with the best matching cluster and node. This resource in turn calls the <a href="#">continuum.aggregator-i</a> to receive the topology.

## API Schemas

The request/response schemas of the resources are provided in the Appendix section **6.1.3**.

### 2.2.8 runtime.execution-manager

The purpose of the component is to parallelize and distribute the workload of an application. Each application container will run a process providing a REST API that allows requesting the execution of a function or python script and managing the resources onto which its workload can be offloaded. The REST API is defined by the [runtime.execution-manager-i](#) interface.

#### 2.2.8.1 runtime.execution-manager-i

Provided RESTful API exposing the resources and actions described below.

### API Endpoints

All URIs are relative to [http://\\${CONTAINER\\_ADDRESS}](http://${CONTAINER_ADDRESS})

HTTP request	Description
GET /COMPSs/test	Health check
GET /COMPSs/resources	Lists the pool of resources where to offload onto
PUT /COMPSs/addResources	Increases the resource pool with some resources
PUT /COMPSs/removeResource	Shrinks the resource pool with some resources
PUT /COMPSs/removeNode	Removes all the resources from one node
PUT /COMPSs/lostNode	Removes all the resources from one node and re-submits the workload currently offloaded onto it
PUT /COMPSs/startApplication	Triggers the asynchronous execution of a function

### API Schemas

The request/response schemas of the resources are provided in the Appendix section **6.1.4**.

### 2.2.9 runtime.execution-manager.offloader

The execution offloader pursues establishing a common interface for other applications (in this case, the execution-manager) to submit tasks and transfer data among nodes in the continuum through multiple protocols. The component is offered as a Java library and several implementations (Adaptors) are included.

### JAVA API

#### es.bsc.compss.comm.CommAdaptor

Class for managing general operations through the adaptor.

Method Name	Description
init	Initialises the Communicator
constructConfiguration	Creates a Configuration object describing how a resource should be contacted
getStarterCommand	Creates a command to start the client of the remote worker through SSH (if needed)

Method Name	Description
initWorker	Initialises a COMPSsWorker instance that represents the node and will be used to interact with the node
getPending	Lists all the data operations that haven't been fully executed
completeMasterURI	Extends a data location with necessary meta-data to contact the specific node
stopSubmittedJobs	Requests the cancellation of all the jobs offloaded onto remote nodes
stop	Stops the Communicator

### **es.bsc.compss.types.COMPSsWorker;**

Each COMPSsWorker represents a remote node.

Method Name	Description
getName	Returns the name of the node
getAdaptor	Returns the Name of the CommAdaptor being used
start	Starts the worker process in the remote node
setInternalURI	Initialises a COMPSsWorker instance that represents the node and will be used to interact with the node
newJob	Creates a new Job description to execute a task on the remote node
sendData	Tells the node to send a data to a third node
obtainData	Tells the node to obtain a data value
stop	Stops the worker process in the remote node

### **es.bsc.compss.types.job.Job;**

Each Job represents the execution of a Task (specific invocation to a method) on a remote resource. The implementation of this interface should manage the necessary communications with the remote node to execute that task.

Method Name	Description
getJobId	Returns the identifier of the Job
getTaskId	Return the identifier of the task
stageIn	Copies all the necessary input data into the remote node
submit	Orders the execution of the Job into the remote node
cancel	Cancels the execution of the job into the remote node

## 2.2.10 continuum.aggregator

This is a service that transforms the infrastructure telemetry data stored on a time series database into the ICOS Infrastructure Taxonomy data structure which is served to the [runtime.matchmaker \[UPC\]](#) via [continuum.aggregator-i](#) interface.

### 2.2.10.1 continuum.aggregator-i

Provided RESTful API exposing the resources and actions described below.

HTTP request	Description
GET /	Returns the infrastructure static and dynamic data.

The request/response schemas of the resources are provided in the Appendix section **6.1.5**.

### 2.2.11 continuum.orchestrator-edge-cloud

This component is external to ICOS. It acts as an aggregator and orchestrator service of the Cloud and Edge resources as defined in [cloud.infrastructure-service](#) and [edge.node](#). It provides cloud and edge device management and application orchestration capabilities on top of the resources it aggregates. The ICOS-compatible Orchestrator for IT-1 is expected to expose the following functionalities via its public interface [continuum.orchestrator-edge-cloud-i](#).

- ▶ Inventory of Edge and IoT Devices
- ▶ Inventory of Cloud resources
- ▶ Application deployment and lifecycle management at the Edge and Cloud
- ▶ Authentication and authorization

#### 2.2.11.1 continuum.orchestrator-edge-cloud-i

Public interface of Cloud and Edge Orchestrator. It is an Orchestrator implementation specific RESTful API. To invoke various actions or query information, the ICOS Controller connects to the provided Orchestrator API via Orchestrator specific connectors (e.g. for deployment management: *runtime.deployment-manager*).

#### 2.2.11.2 continuum.orchestrator-edge-cloud-i.deployment

Deployment instantiation and lifecycle management interface. It is an Orchestrator implementation specific RESTful API. Used by [runtime.deployment-manager.cloud-edge-orch.driver \[ATOS\]](#) for application deployment lifecycle management.

### 2.2.12 cloud.infrastructure-service

Resources available on Cloud

- ▶ IaaS (e.g. VM management, S3, etc); exposed via [cloud.infrastructure-service.iaas-i](#)
- ▶ Container Orchestration Engine (COE) clusters; exposed via [cloud.infrastructure-service.coe-i](#)

#### 2.2.12.1 cloud.infrastructure-service.iaas-i

Public IaaS interface of a cloud service.

#### 2.2.12.2 cloud.infrastructure-service.coe-i

Public COE interface of COE running on cloud.

### 2.2.13 edge.node

Edge node running COE.

#### 2.2.13.1 edge.node-i

COE of types:

- ▶ Docker
- ▶ Docker Swarm
- ▶ Kubernetes

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	22 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

### 2.2.14 *telemetry.agent*

ICOS implements a full observability framework to effectively and efficiently acquire data on the status and the performance of the infrastructures and the applications and to take decisions to improve the usage of resources and the performance of the system. In IT-1, the components responsible for managing the telemetry data are the “Telemetry Agent” and the “Telemetry Controller”. The Agent is deployed in the computational nodes and is responsible to collect data (i.e. metrics and logs) from the system, to appropriately label it and to transmit it to the Telemetry Controller. The Controller is responsible for receiving the data, storing it and providing interfaces to query and visualise it. In the following development iterations these components will be enriched with support for acquiring more types of data, alerting and analysis tools, and authentication and authorization features.

The *telemetry.agent* has the goal of implementing a generic mechanism to collect metrics and logs from the systems, from other ICOS components and from user applications, and to transmit them where they can be stored, analysed and accessed from other components.

The Telemetry Agent is deployed in the cloud and edge nodes during the on-boarding process. Depending on the characteristics and capabilities of the node, it is properly configured to acquire the telemetry data. The Agent acquires the logs of the system and the applications running in the node, by reading the corresponding files in the node's filesystem and watching for changes in these files. For collecting metrics, the Agent supports both push and pull paradigms. In the pull paradigm the Agent is configured via endpoints that expose metrics. On a regular basis, the Agent reads metrics from these endpoints. The Telemetry Agent also implements mechanisms to automatically discover metrics endpoints available on the system. In push mode, the components of the infrastructures that generate or collect metrics are responsible for sending metrics to the Telemetry Agent.

When data is received, the Telemetry Agent applies simple processing of the received data, like adding ICOS-related metadata (e.g. the ICOS Node ID), aggregating (reducing the amount of data) or filtering (e.g., removing unnecessary metrics).

Finally, the Agent transmits the data to the configured Telemetry Controller. The Agent will also be able to send data to another Agent. This is an important feature in deployment scenarios where the Telemetry Agent in the node has no network connectivity with the Controller. In these cases, data can be transmitted from the node to the controller through a chain of intermediate Telemetry Agents that act as bridges. Other reasons for having intermediate Telemetry Agents can be data efficiency (the agents in the middle can further aggregate and filter data), caching of data (e.g. in case of loss of network connections), or utilising data for taking local decisions.

Concerning data models, the Agent supports collection of metrics both in the Prometheus text format<sup>11</sup> and in the OpenTelemetry Protocol (OTLP) format<sup>12</sup>, while transmitting data using the OTLP format only. For the transmissions both HTTP and gPRC<sup>13</sup> are supported.

### 2.2.15 *telemetry.controller*

The *telemetry.controller* runs in the ICOS controllers and offers an interface to acquire data transmitted by the Telemetry Agents running in the nodes registered to the ICOS Controller where it is running. The acquired data is stored in long-term storage. Given the different nature of data (e.g., metrics and logs) different specialised storage technologies are adopted to store the different types of data. This guarantees high efficiency in the storage of large amounts of data for a long retention period. The controller offers an API for querying the stored data. This will be used by the ICOS components to access the data. In the future development iterations, the Controller will also offer an API to subscribe to changes in metrics and definition of alerting rules.

<sup>11</sup> [https://prometheus.io/docs/concepts/data\\_model/#notation](https://prometheus.io/docs/concepts/data_model/#notation)

<sup>12</sup> <https://opentelemetry.io/docs/specs/otel/protocol/>

<sup>13</sup> <https://grpc.io/>

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)			<b>Page:</b>	23 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0
				<b>Status:</b>	Final

Finally, the *telemetry.controller* provides an interface to visualise the data via a web application. At the moment this is intended to be used by administrators for assessing the status of the system and debugging issues. In the future, the access will be allowed to users that will be able to see data related to their applications running in the system.

### 2.2.15.1 *telemetry.controller-i.query*

Provided RESTful API exposing the resources and actions described below.

#### API Endpoints

All URIs are relative to */api/v1*

HTTP request	Description
GET <i>/query</i>	Perform a query to access metrics. The query must be sent in the “query” string parameter and must be in PromQL <sup>14</sup> language.
POST <i>/query</i>	Using the POST method when specifying a large query that may breach server-side URL character limits.

The schemas of the resources are provided below.

#### */api/v1/query*

Attribute Name	Type	Description
<i>icos_agent_id</i>	Long	Unique identifier of the deployment
<i>name</i>	String	Name of metrics

## 2.3 Detailed design

Based on the above listed targets, design considerations and constraints, below, we present the component and sequence diagram that define the ICOS and external components and their interactions for the Meta-Kernel layer implementation at IT-1.

### 2.3.1 Node On-Boarding

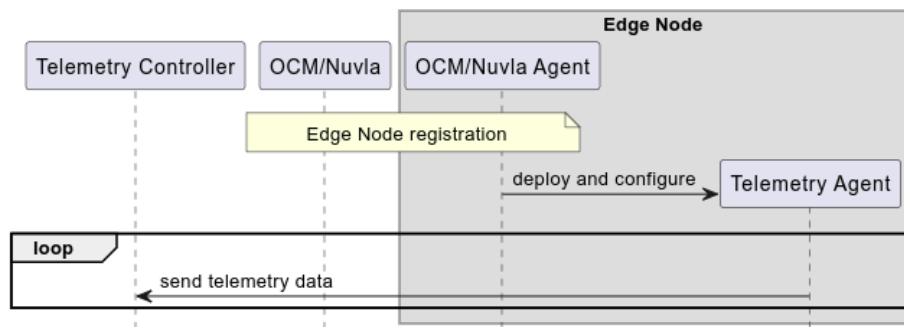
The onboarding of ICOS Node in the scope of IT-1 results in adding *continuum.orchestrator-edge-cloud* instances to the ICOS Controller via the ICOS Agent. In this case it is assumed that the onboarding of the actual compute and storage resources of *edge.node* and *cloud.infrastructure-service* happens internally in *continuum.orchestrator-edge-cloud* via an implementation specific mechanism of the concrete *continuum.orchestrator-edge-cloud* instance. Following the design component Figure 2 3 for IT-1, the binding element between *continuum.orchestrator-edge-cloud* and ICOS Controller is *runtime.deployment-manager* component that is part of the ICOS Agent.

The processes of the onboarding of *edge.node* and *cloud.infrastructure-service* to the concrete selected implementations of *continuum.orchestrator-edge-cloud* can be found in section 3.

The enrichment of the Edge Cluster to become ICOS-compliant is described in the following diagram:

<sup>14</sup> <https://prometheus.io/docs/prometheus/latest/querying/basics/>





The enrichment of Cloud cluster is not yet applicable, unless this cluster is intended to be an ICOS Controller. If so, the following components need to be provided during the on-boarding as well:

- ▶ Shell Backend - user.shell-backend
- ▶ Job Manager - runtime.job-manager
- ▶ Matchmaker - runtime.matchmaker
- ▶ Execution Manager - runtime.execution-manager
- ▶ Execution Manager Offloader - runtime.execution-manager.offloader
- ▶ Aggregator - continuum.aggregator

### 2.3.2 Basic Application Deployment

Note: The application deployment process operates using the following Application Models

- ▶ ICOS Application Model - describes user application and provided by user to the *user.shell*
- ▶ ICOS Application Deployment Model - ICOS Controller internal representation of the Application Model for deployment built based on the ICOS Application Model plus actual deployment target and other bookkeeping information.

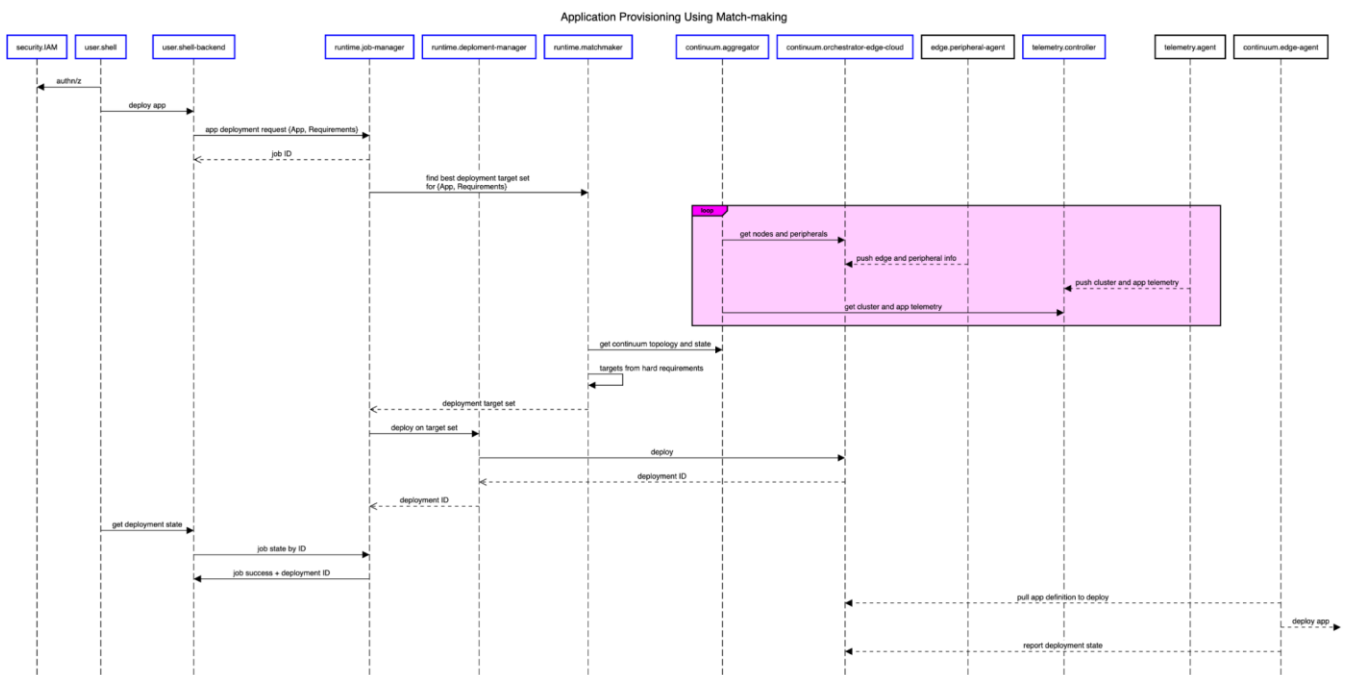


Figure 2-5 Meta-Kernel layer for IT-1: Basic Application Deployment.

### 2.3.3 Collect and visualise Metrics and Logs

The sequence diagram in Figure 2-6 presents how the ICOS components interact to collect and visualise telemetry data. In the section, some examples of metrics endpoints are shown, but potentially additional endpoints will be used in ICOS.

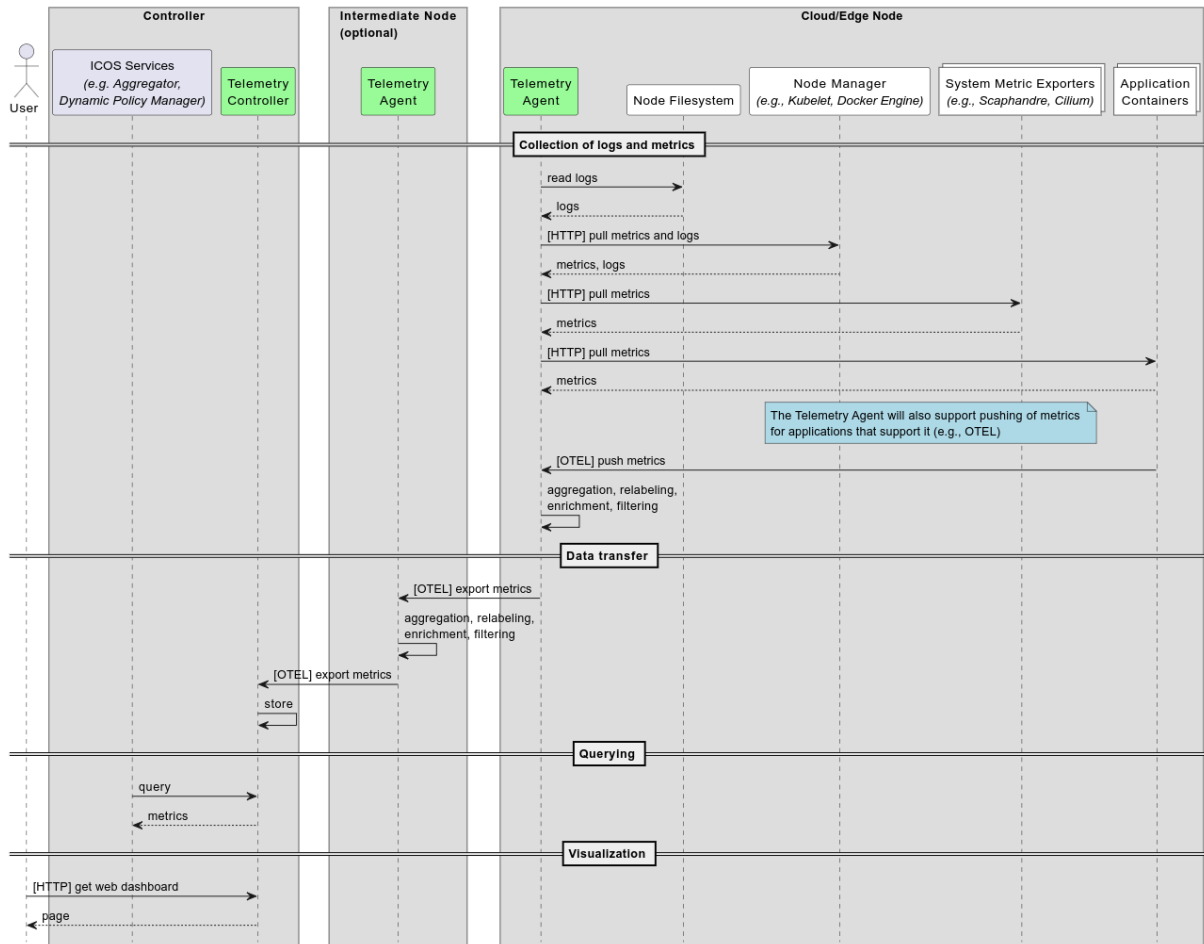


Figure 2-6 Interaction between ICOS components to collect and visualise telemetry data.

## 3 Technology and Tools Selection for Implementation

This section lists the tools that were used to implement the design components and interfaces described in the previous section.

### 3.1 Lighthouse

For IT-1, the lighthouse is a web service that contains a list of ICOS controllers. The lighthouse follows the OpenAPI specification to add new controllers and to retrieve the list of existing ones. The ICOS controllers are responsible for adding themselves to the lighthouse and to update their entries every 60 seconds. Each entry in the list has a timer associated which is reset when an update from the specific ICOS controller is received. If this update is not received from the ICOS controller and a timeout occurs, the entry is deleted from the list assuming the ICOS controller is not available anymore.

The lighthouse is implemented in Go and it is publicly accessible in <http://lighthouse.icos-project.eu:8080/api/v3/controller>.

Assuming the lighthouse is deployed in <http://lighthouse.icos-project.eu>, we now show an example where ICOS controllers and agents join the system. Figure 3-1 shows a node in Domain 1 that is registered into the lighthouse with reachable domain “dom1.icos-project.eu”. When new nodes (node 11 and 12) from the same Domain 1 attempt to join the system, they first query the lighthouse to retrieve the list of available controllers. Since there is already an existing ICOS controller within the same domain, nodes 11 and 12 become ICOS agents from the already existing controller.

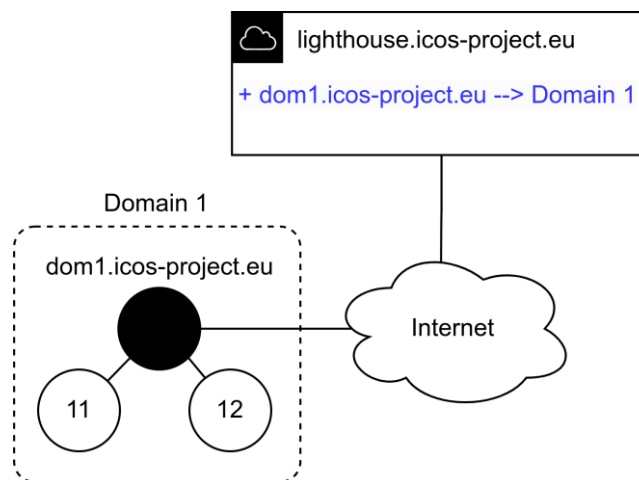


Figure 3-1 Controller registration with one domain

Following the same procedure, when a new node from Domain 2 attempts to join the system, the node first retrieves the list of available controllers from the lighthouse. In this case, since the existing ICOS controller is in another domain, the new node from Domain 2 becomes an ICOS controller itself, as shown in Figure 3-2.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	27 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

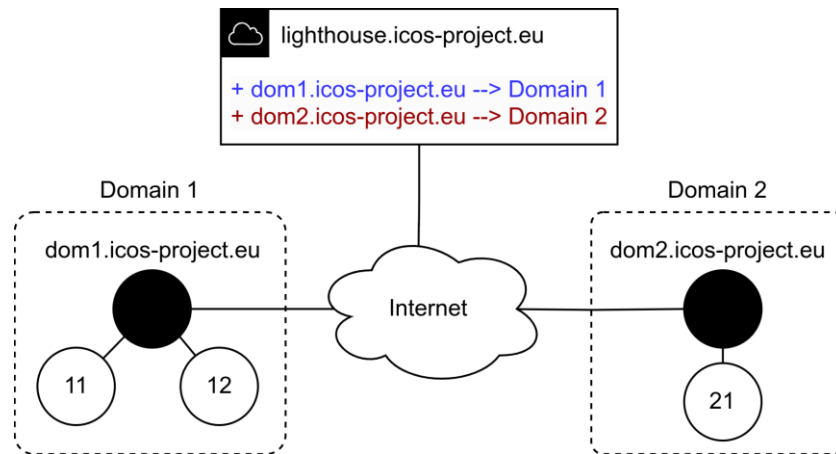


Figure 3-2 Controller registration with two domains

In this case the new node registers into the lighthouse as “dom2.icos-project.eu”. When a new node (node 21) from Domain 2 attempts to join the system, it first retrieves from the lighthouse the list of registered controllers and, in this case, the node joins to the controller in Domain 2.

In the same manner, the shell client and GUI retrieve the controller endpoints from the lighthouse to enable ICOS management for the user.

## 3.2 Continuum Manager

### 3.2.1 Resource and Clustering Manager

The following functionalities of the ICOS Resource and Clustering Manager (*continuum.orchestrator-edge-cloud*) were realised in IT-1

- ▶ Edge device and cloud resource onboarding
  - addition of Kubernetes and Docker based Edge devices to the ICOS continuum.
  - addition of Cloud resource endpoints to the ICOS continuum.
- ▶ Deployment of user applications on the onboarded edge resources.

This was achieved by integration with the following technological tools that were used as ICOS Resource and Clustering Managers

- ▶ Nuvla <https://nuvla.io>
- ▶ OCM <https://open-cluster-management.io/>

#### 3.2.1.1 Nuvla as Resource and Clustering Manager

For IT-1, Nuvla service was provided to the project for integration on <https://nuvla.io> endpoint (Nuvla.io). Nuvla.io is a B2B Platform for cloud and edge management and applications orchestration. The service is fully managed by SixSq. As this is described in deliverable D2.1, Nuvla.io service consists of two main components - Nuvla and NuvlaEdge, where Nuvla is the B2B Platform providing the management API and NuvlaEdge is the agent that is deployed on edge devices.

The source code of Nuvla is available at <https://github.com/nuvla>. The source code for NuvlaEdge is available at <https://github.com/nuvlaedge>.

The Nuvla API documentation is available at <https://docs.nuvla.io/>.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	28 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

## Onboarding

The process of onboarding of Edge devices running Kubernetes or Docker COE to Nuvla is described in the Nuvla online documentation here <https://docs.nuvla.io/nuvlaedge/installation/>. In essence, (on the example of Kubernetes COE based edge device) the process consists of first creating the Edge device reference in Nuvla.io platform (see Figure 3-3)

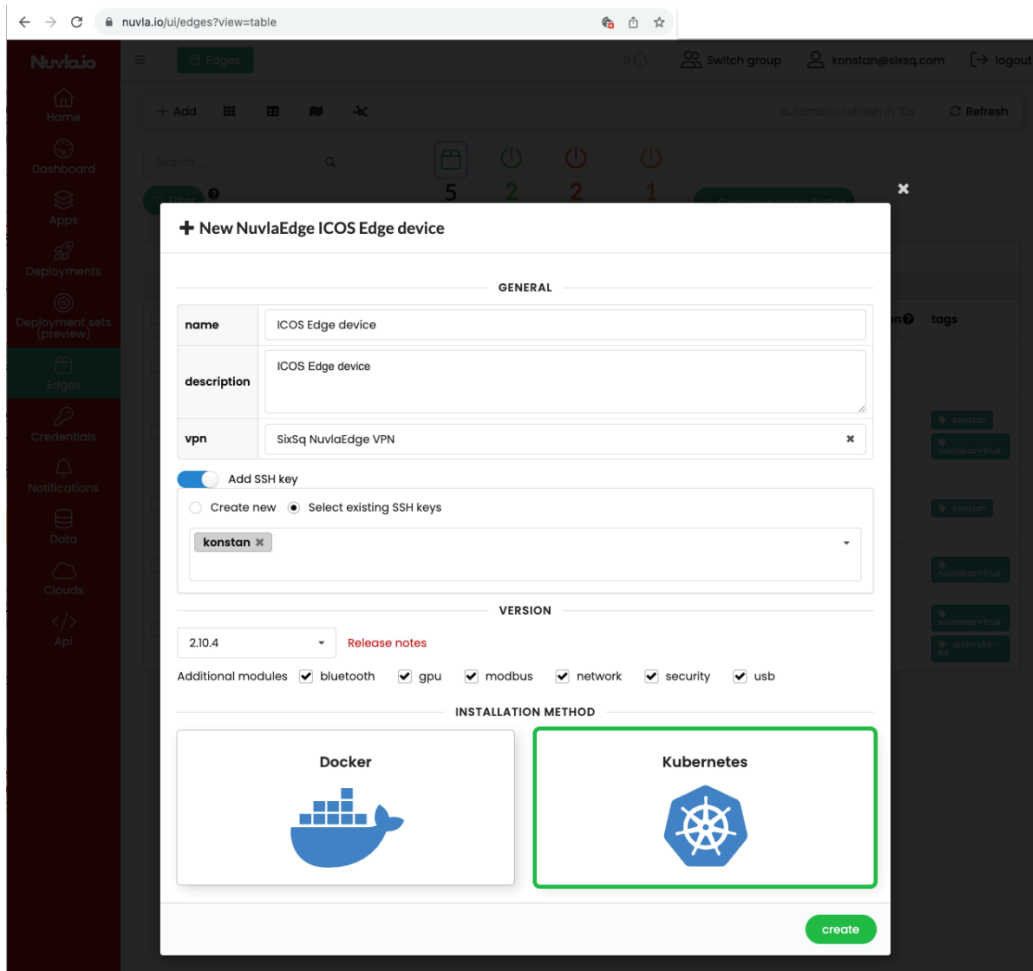


Figure 3-3 Nuvla: Onboarding of Edge device - creation of NuvlaEdge in Nuvla.

and then, running the provided Helm command to provision the NuvlaEdge on the device (see Figure 3-4)

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	29 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

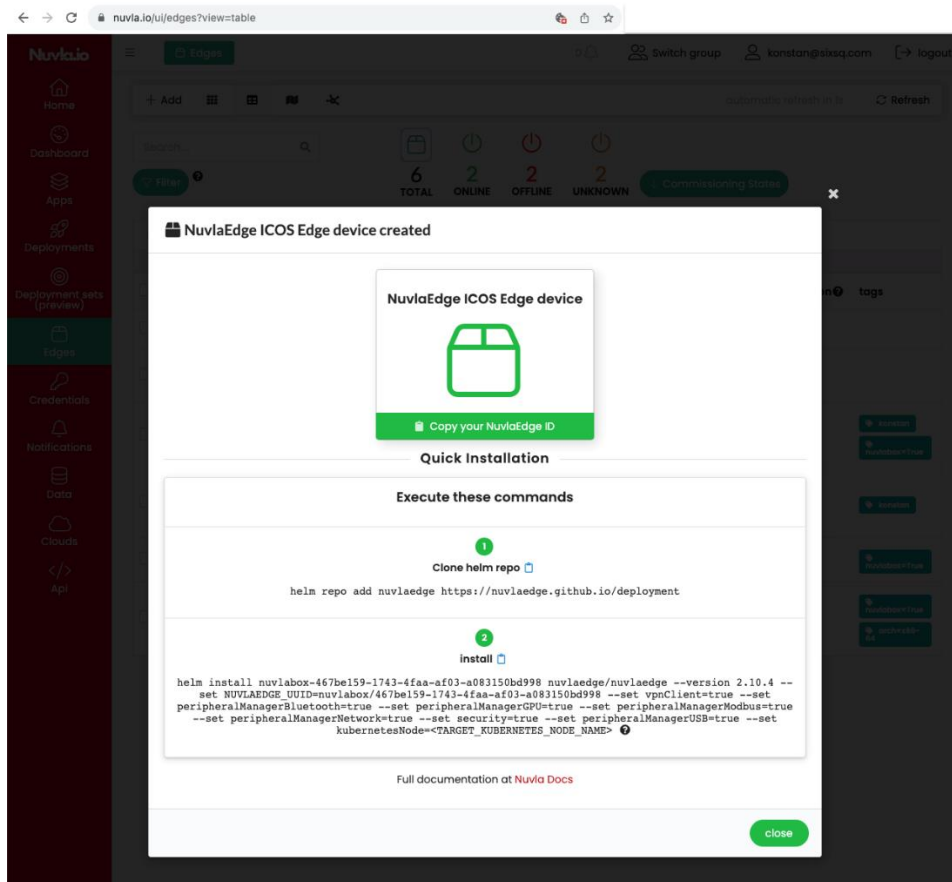


Figure 3-4 Nuvla: Onboarding of Edge device – helm command for deployment of NuvlaEdge on Edge device.

After NuvlaEdge is deployed, it connects to Nuvla.io and commissions itself into an operational state (see Figure 3-5).

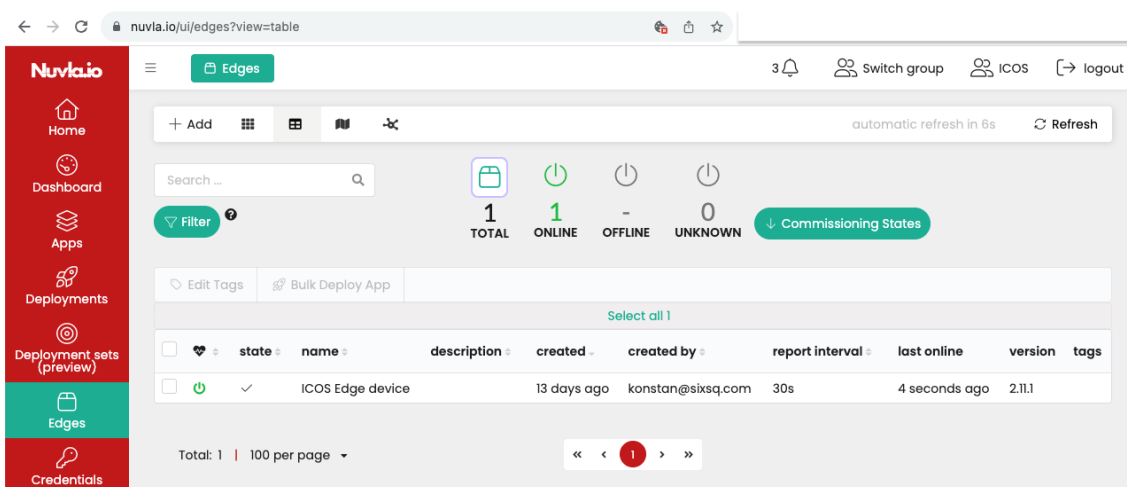


Figure 3-5 Nuvla: Onboarding of Edge device – Edge device is operational.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	30 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

The Figure 3-6 below shows the required and optional NuvlaEdge components deployed as part of the onboarding.

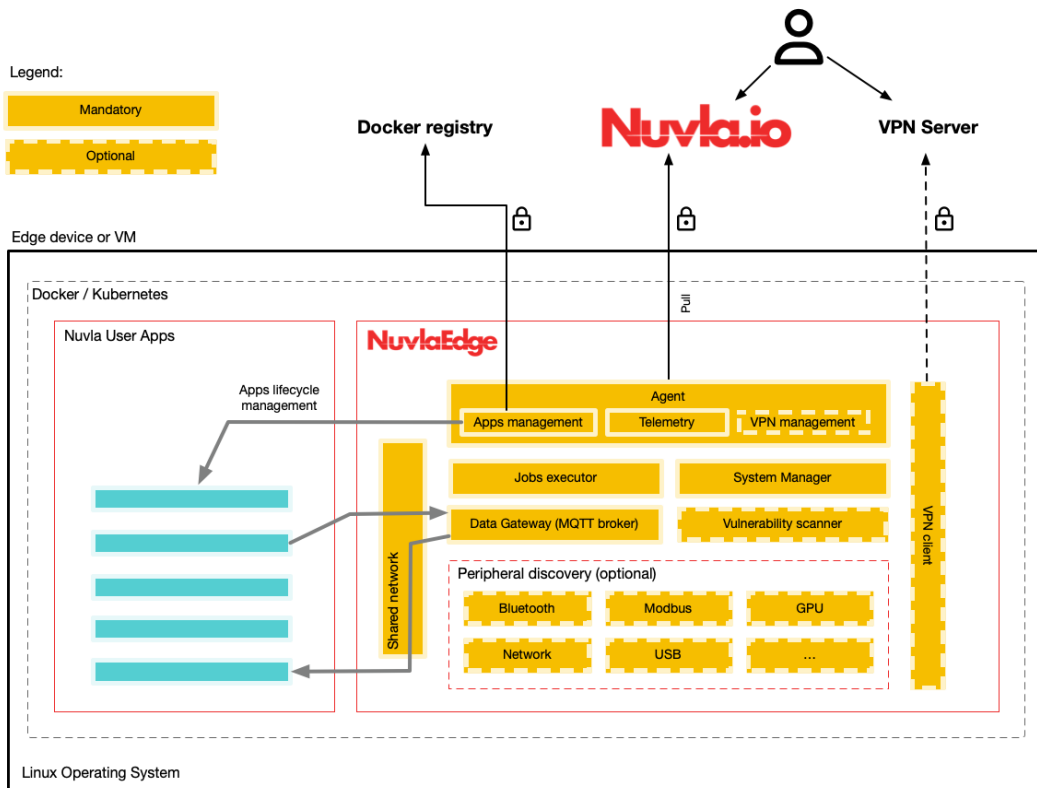


Figure 3-6 NuvlaEdge conceptual architecture: components, relation to Nuvla.io, and management actions.

### Peripheral discovery

Nuvla Platform provides the functionality of the discovery of the peripherals. NuvlaEdge peripheral discovery component can discover the following peripheral types

- ▶ bluetooth
- ▶ gpu
- ▶ modbus
- ▶ network
- ▶ usb

The peripheral discovery process on the edge devices is continuous. The discovered peripherals and information about them are published to Nuvla and available via API and can be seen on the web UI (see Figure 3-7).

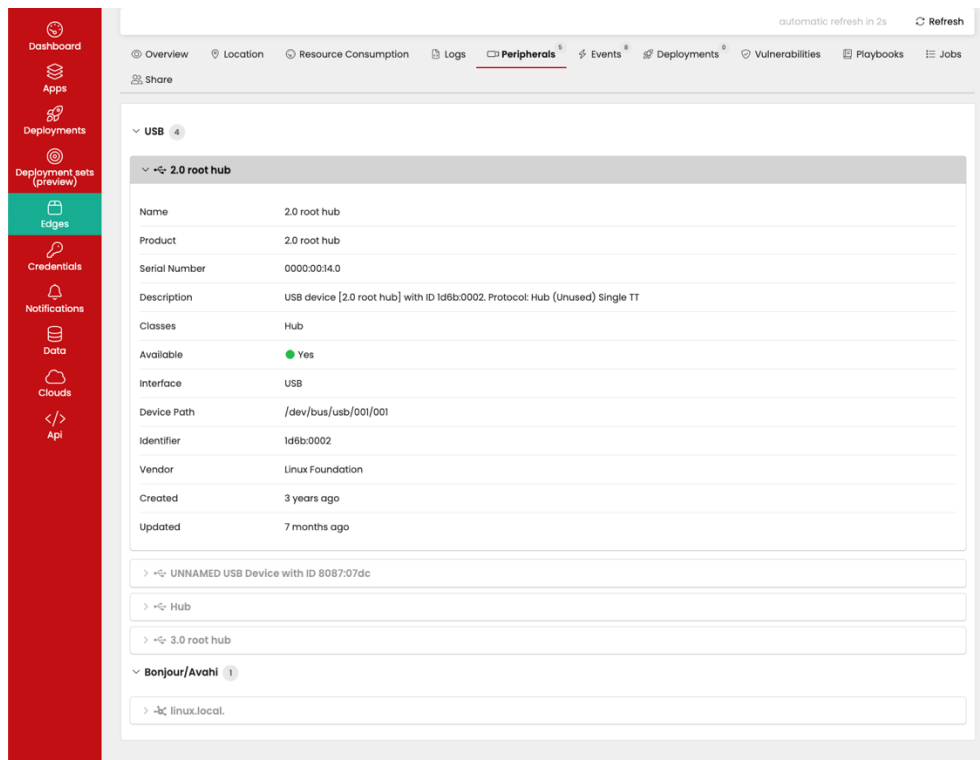


Figure 3-7 Nuvla: Discovered peripherals on edge device.

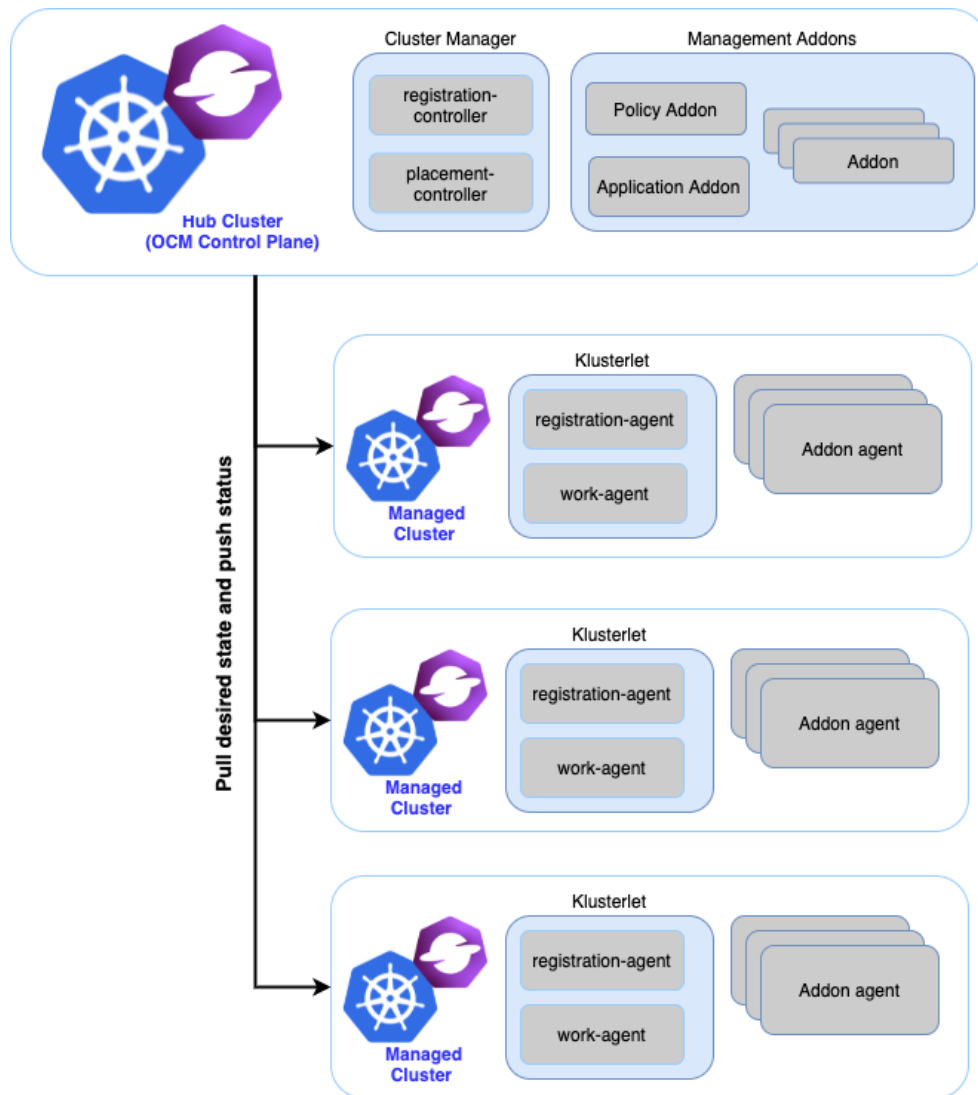
### 3.2.1.2 OCM as Resource and Clustering Manager

Open Cluster Management is a Kubernetes multi-cluster management tool. It allows the Kubernetes operator to orchestrate and manage a set of clusters. The OCM architecture is based on originally defined by Kubernetes pattern “**hub-kubelet**” where two main components are defined:

- ▶ **Hub Cluster:** denotes the cluster containing the multi-cluster control plane for OCM, traditionally, this cluster must only execute management workload and contain a limited number of Kubernetes resources (services, statefulSets, etc...).
- ▶ **Klusterlet:** indicates that cluster is being managed by the Hub Cluster. Klusterlet executes a “work-agent” responsible for pulling workload from Hub Cluster and ensures to achieve the desired state of resources specified by “placement-controller” within the Hub Cluster.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	32 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final





- ▶ Addons & Addon Components are not part of the scope of this document and won't be highlighted.
- ▶ Registration Components provide a way to onboard new managed clusters to the existing setup, as described below.

Open Cluster Management Orchestrator offers a registration process that allows clusters and nodes to join the orchestration ecosystem. This ecosystem is only part of the ICOS Ecosystem. For this reason, the process provided by OCM is enriched with ICOS-specific components to be installed during the onboarding. The following sequence diagram describes the basic OCM-onboarding process, that is valid for any kind of cluster or node to onboard:

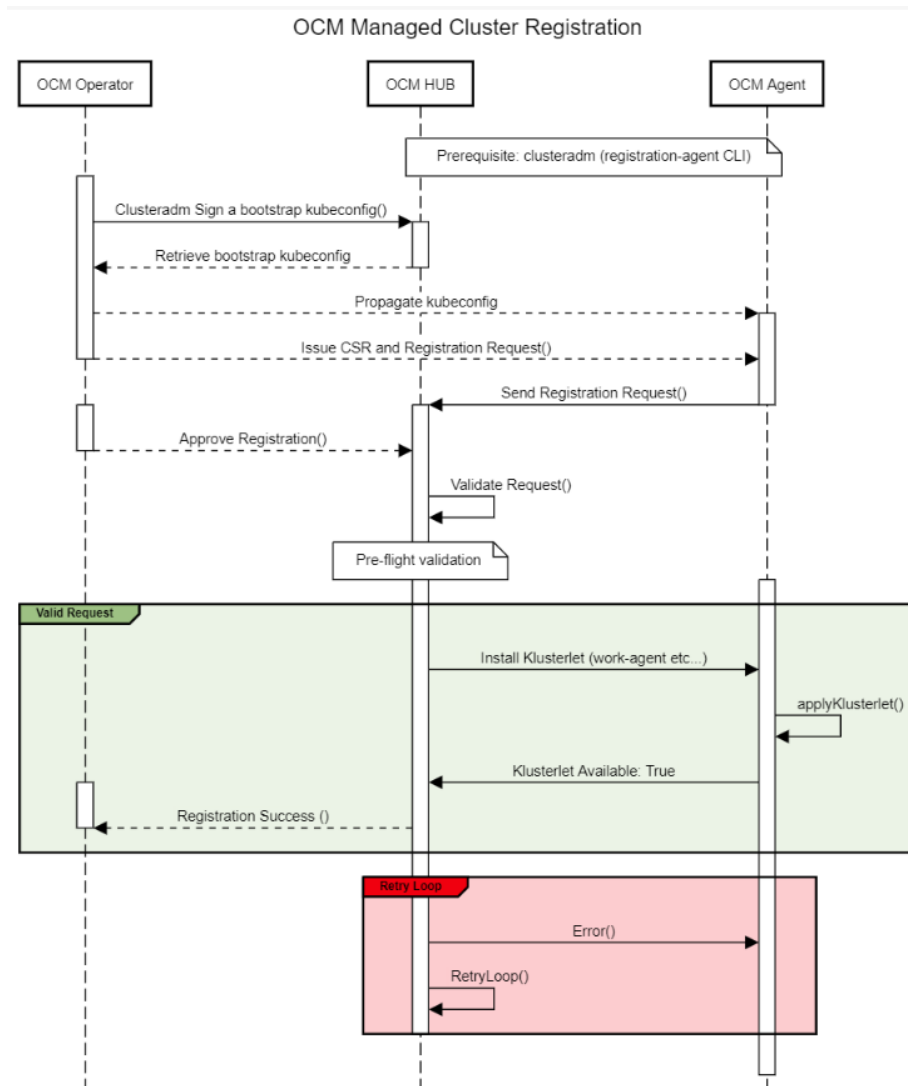


Figure 3-8 Sequence diagram of onboarding of a Kubernetes cluster to OCM.

### 3.2.2 Dynamic Policies Manager

As reported in the deliverable D2.2 ICOS Architecture Design, DPM is responsible for the management of the policies (technical and business performance ) and the detection and prediction of violations of such policies in the running application. Based on this, we identify three sub-components of the DPM (as shown in the DPM architecture Figure 3-9):

- ▶ **Policies Configurator:** responsible for configuring the policy metrics specifically the alter metrics and sending themes to the Monitoring and Telemetry component.
- ▶ **Policies Evaluator:** responsible for checking out the policy alert and values received from the Monitoring and Telemetry components and sending them to the Policy Violation Manager if a violation is identified.
- ▶ **Policies Violation Manager:** is responsible for storing the violation sent by the Policy Evaluator and sharing it with the other ICOS services (for example Job Manager).

The Figure 3-9 provides the initial architecture of the Dynamic Policies Manager.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	34 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

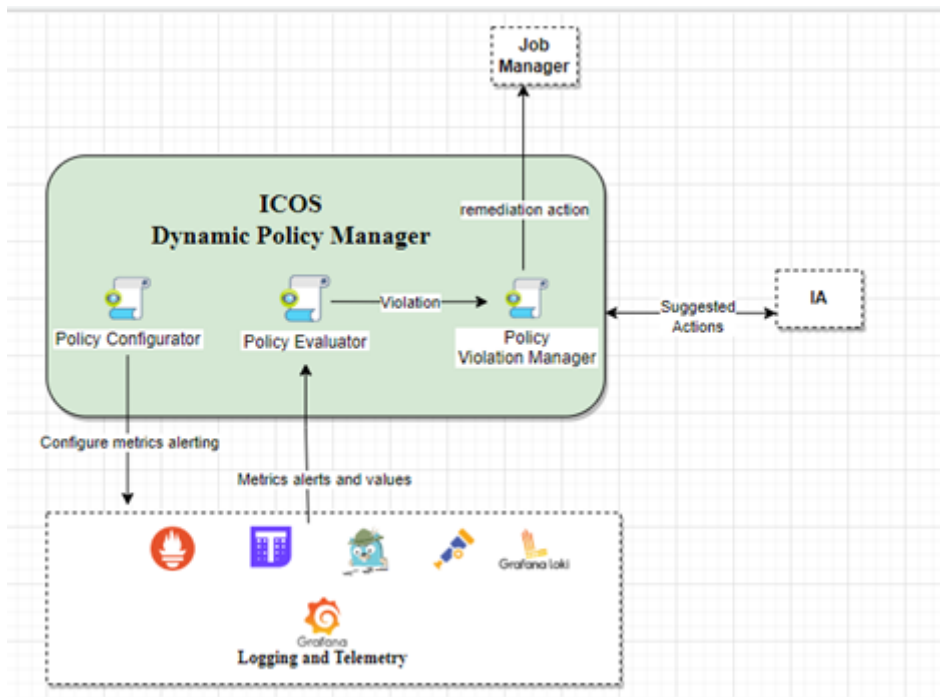


Figure 3-9 Initial architecture of the Dynamic Policies Manager.

#### *How it works*

The Dynamic Policy Manager starts to define and send the policy configured to the Monitoring and Logging component. The DPM creates and sends the policy information to the Monitoring and Telemetry component. The monitoring component sends an alert and the values of the violation -if needed- to the Policy Evaluator. The Policy Evaluator sends the alert violation to the Policy Violation Manager.

The Policy Violation Manager stores and analyses the violation and if the remediation action is provided by itself; it then sends the remediation action to the Job Manager otherwise the Policy Violation Manager sends a request of the suggested action to the intelligence block.

An API HTTP REST will be implemented to manage the communication and the work of the Dynamic Policy Manager: work and implementation of that will be reported in deliverable “D3.2 Meta-kernel Layer Module Developed (IT-2)”.

#### *How to implement the policies*

The complexity of ICOS infrastructures makes it difficult to guarantee optimizations and high-performance levels without resorting to specific control tools. In particular, to evaluate the performance of applications in the ICOS ecosystem it is necessary to choose reference metrics, on the basis of which it will determine the level of functioning and then deduce any necessary improvements. It is important that these parameters allow us to observe both how data moves from one device to another within ICOS systems, and how the information is processed and retrieved.

To perform this, we decided to define and analyse the first version of the Dynamic Policy Manager, including the performance policies.

In general, the performance policies are a set of performance objectives used to define the method with which the data is stored on the ICOS platform in order to achieve optimal performance for the specific application. Observability and performance monitoring data tools can be a valuable aid and support for designing and implementing these policies.

In general observability data tools allow operators to monitor their clusters or their systems. There are three pillars of observability: metrics, logs, and traces.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	35 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

Some tools were investigated in order to determine whether they can be considered as a possible solution for the ICOS ecosystem by plugging them in the ICOS architecture. The first considered tool was OPNI (<https://opni.io/>). Although this tool allows defining, monitoring, and sending alerts for SLO; its SLO/SLI (Service Level Indicator) follows the Site Reliability Engineering (SRE) approach that is focused on finding issues in the infrastructure operation based on Good Events and Bad Events; but this is not fully in line with the objective of the ICOS ecosystem, as it is more focused on the performance of applications.

In addition, the SLO monitoring is not “cross-cluster” and is similar to what can be achieved using Prometheus Alerting Rules. Other reasons for not using OPNI are the following:

- ▶ The ICOS Performance Policies model requires more features than those that OPNI provides. For example, ICOS Performance Policies should include the possible remediation actions.
- ▶ It is not a unified UI.
- ▶ The development phase is still very active.
- ▶ The documentation is fragmented, poor, and not always up-to-date.

For this reason, OPNI was not selected and it was decided to define the ICOS policies based on other open source tools such as Prometheus (<https://prometheus.io/>) and Thanos (<https://thanos.io/>).

The final ICOS policies model will be provided in the deliverable D3.2 about the policies language model used and its implementation. Below in the code snippet we provide an example of the policies model that acts as an initial implementation of the model.

The rule should be created as an XML file that will include all necessary information (rule statement) and will be loaded through a specific field (rule\_files) in the DPM configuration. This configuration includes:

- ▶ **policies** groups identify the collection of the information for the performance policies
- ▶ **name**: name of the policy, for example, policy1.
- ▶ **type**: type of the policy in this first version we have only Performance policy
- ▶ **apply**: it is a record that contains information about the component that the policies is apply to as for example name of the component
- ▶ **expression**: it provides a mathematical expression aimed at defining a very specific limit beyond or below which a violation occurs, for example `avg_execution_time_predict < 5m` : a violation occurs when the average application performance time lasts for more than five minutes; the application duration time is violated if it exceeds ten minutes.
- ▶ **for** time duration
- ▶ **remediations**: list of remediation actions to adopt. It is based on the type of violation.

```

policies:
  - name: String,
    type: String,
    applyTo:
      - component1: nameofcomponent1 : String,
        expression: mathematical expression about the time range limit about the application performance,
        for time duration
        remediations:
          - list of actions
  
```

**Example**

```

policies:
- name: policy1
  type: performance
  apply to:
  - component-1
  expression: avg_execution_time_predict < 5min
  for: 10m
  remediations:
  - scale_pod
  
```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	36 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

## 3.3 Runtime Manager

---

### 3.3.1 Job Management

Runtime.job-manager was implemented using Go language and following the good practices regarding the microservices. The source code can be found in the following repository: <https://production.eng.it/gitlab/icos/meta-kernel/job-manager>

This microservice implements the RESTful API described in the section [runtime.job-manager \[ATOS\]](#) and integrated in the testbed environment. Whenever it is rolled out it exposes the RESTful API defined by this interface [runtime.job-manager-i](#).

### 3.3.2 Matchmaking

The Matchmaking service (MM) is responsible for providing the best topology for the execution of an ICOS application. It is part of the Job Manager (JM) component, and is invoked by the JM upon reception of an application execution request from an ICOS user.

The MM receives from the JM an application description model and requests to the Aggregator service (see next subsection) the current ICOS resources infrastructure. The MM will then find an optimal infrastructure topology (best matching between the app description and the ICOS infrastructure) supported by the Intelligence Layer, and consider any application constraints in terms of required hardware and/or required IoT devices and data sources.

For IT-1 the MM service has been implemented as a web service exposing an API to be called by the JM and using the API exposed by the Aggregator service. At this implementation stage, the matching process considers all application constraints and provides a basic topology based on a first-fit rule. In future releases the selection of the best topology will be based on machine learning technology obtained from historical executions and given some enhanced application description from the ICOS users.

The code has been implemented in Python and it is publicly accessible through [http://\\${MATCHMAKER\\_ADDRESS}](http://${MATCHMAKER_ADDRESS}). At this stage, the URL is <http://147.83.159.195:24780>.

### 3.3.3 Aggregator

In a multi-cluster architecture, the aggregator service provides knowledge of the static and dynamic properties of a resource infrastructure associated with a particular ICOS ecosystem deployment in terms of resource performance, resource availability, eco-efficiency characteristics and peripheral hardware (USB ports, CPU/GPU, connected IoT devices such as cameras, sensors, actuators, etc.).

For IT-1, the aggregator component is a web service that exposes a web API that is called by the matchmaking service. The API provides the infrastructure model or ICOS Infrastructure Taxonomy that describes the characteristics of the ICOS infrastructure. The aggregator is implemented in Go and it is accessible from the matchmaking service in [http://\\${AGGREGATOR\\_ADDRESS}/api/v1](http://${AGGREGATOR_ADDRESS}/api/v1).

The aggregator service queries Thanos (as the central point to which telemetry data is sent) to retrieve stored metrics (more details can be found in the Logging and Telemetry subsection of this chapter). The Figure 3-10 below shows a high-level architecture of the service.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	37 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

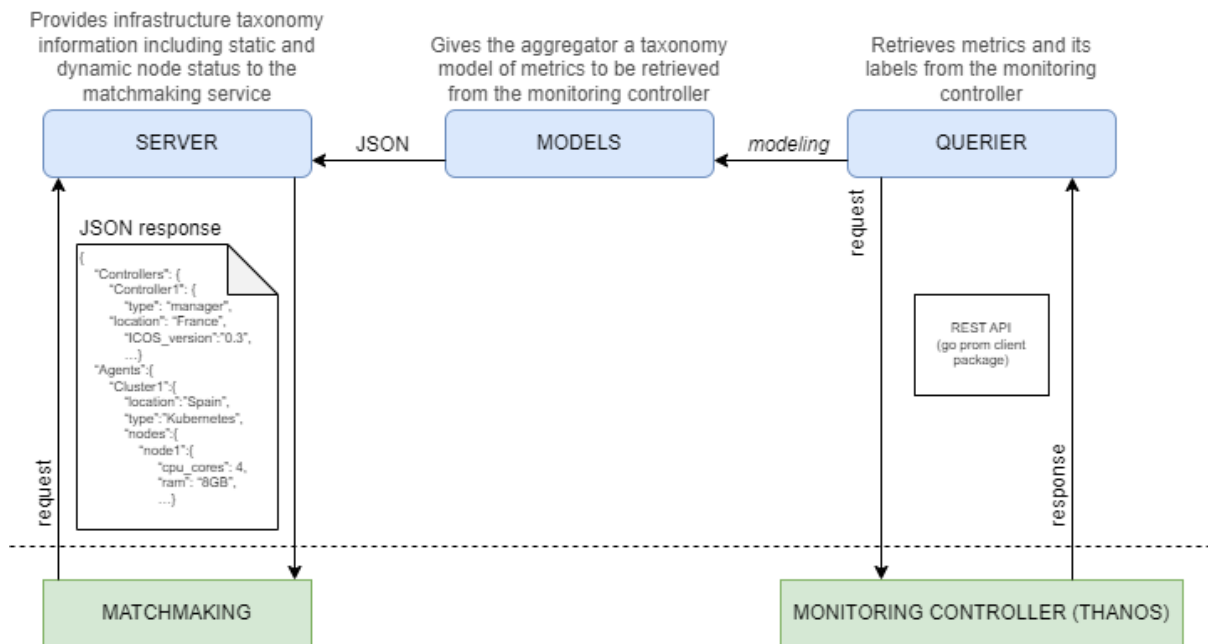


Figure 3-10 High-level architecture of the Aggregator service.

The Models component of the Aggregator architecture modifies the data structure received from Thanos server according to the IT-1 ICOS Infrastructure Taxonomy model. This taxonomy is used to describe and categorise the ICOS infrastructure that spans throughout the cloud, edge and IoT. Unlike traditional cloud infrastructures, edge computing devices are located closer to the end user in a distributed geographical location. However, these devices have certain weaknesses compared to the cloud, in terms of compute, storage and connectivity capabilities. The ICOS taxonomy recognises the hierarchy of continuous computing, its complexity and multiple characteristics, and allows large cloud infrastructures and datacenters to be categorised as edge clusters and edge nodes to reflect the fact that application execution can be organised at different levels.

The taxonomy used in this IT-1 can be found at the following URL located in the project source code repository:

<https://production.eng.it/gitlab/icos/meta-kernel/aggregator/-/blob/develop/Infra-taxonomy.md>

### 3.3.4 Distributed & Parallel Execution

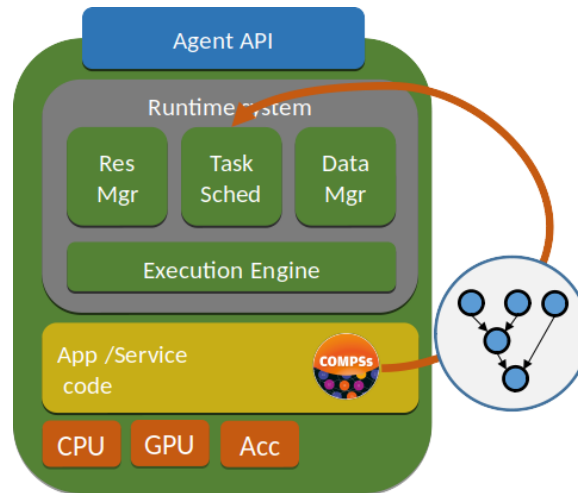
COMPSs/PyCOMPSs is a programming framework aiming at easing the development of general-purpose applications targeting the Cloud-Edge-IoT Continuum. COMPSs is a task-based programming model that orchestrates the execution of such tasks in a serverless manner on top of any distributed platform. PyCOMPSs is an enhanced version of the programming model exploiting the benefits of the Python programming language.

Service developers code their application logic following the COMPSs programming model. The COMPSs runtime receives an external request to compute something coming either from a manual petition by an end-user or automatically triggered as a response to a change on the data or the infrastructure related to the service. These requests to execute code functions arrive at the COMPSs runtime via HTTP requests.

Upon the reception of such a request, the runtime engine divides the application into several tasks, detects the data dependencies among them, and orchestrates the execution of such tasks across all the

resources within the infrastructure aiming to achieve a shorter execution time and an efficient usage of the nodes belonging to the underlying infrastructure.

The following image depicts the architecture of the programming model runtime. Each node belonging to the infrastructure runs a daemon process, known as Agent, that handles the different computation requests that arrive through the Agent API.



Upon the reception of this request, the API submits to the Runtime engine a task that encapsulates the execution of the main code of the computation. Such an engine is composed by four main components:

- ▶ Resource Manager: keeps track of the computing resources currently available (embedded on the device or available as agents on remote nodes).
- ▶ Task Scheduler: picks the resources and time lapse to host the execution of each task while meeting dependencies among them and guaranteeing the exclusivity of the assigned resources
- ▶ Data Manager: stores locally data values and establishes a data sharing mechanism across the whole infrastructure leveraging on the Workload Offloader component.
- ▶ Executor Engine: handles the execution of tasks on the resources embedded on the local device (CPU, GPU, FPGA or any other accelerator). If tasks are Java methods, they are executed within the same progress. To execute Python tasks the runtime system raises several slave processes that host tasks execution while the agent is up. The runtime system can also run other types of methods, such as binaries or containers by launching new processes. When the local computing devices execute a code programmed following the COMPSs/PyCOMPSs programming model, new tasks are spawned and submitted back to the runtime engine so that the runtime handles their execution in the same way as it was done for the main task.

When the Task Scheduler decides to execute a task (function) in the resource embedded on the local device, the task is directly submitted to the Executor Engine. Otherwise, if it decides to offload a task onto a remote node, the task is forwarded to the Workload Offloader component to submit the execution through the most appropriate protocol.

### 3.3.5 Workload Offloader

The runtime engine supporting the COMPSs programming model already includes a component that handles the offloading of workload from one device in the Cloud-Edge-IoT continuum onto another device. The current implementation of the engine supports the offloading – including the necessary stage-in and stage-out data operations – of a function’s execution, either sequential or multi-threaded, through SSH/SCP and through a proprietary protocol building on TCP sockets to another dedicated worker process (not able to detect nested tasks).

The current engine also offers offloading mechanism through the COMPSs API which is publicly offered as a REST service or it is also accessible through as a TCP-based proprietary protocol. Function executions offloaded through the Agent API are able to detect nested tasks and distribute the workload of that function across more devices in the Cloud-Edge-IoT Continuum.

## 3.4 Logging and Telemetry

The Logging and Telemetry architectural functionality is composed of two components:

- ▶ the Telemetry Controller that runs in the ICOS Controllers and is responsible for storing data and provide interfaces to query and visualise it;
- ▶ the Telemetry Agent that runs in the infrastructure nodes and is responsible to collect data from the system and the applications and to transmit it to the Telemetry Controller.

From a technological point of view, these two components heavily rely on third-party open-source solutions for data collection and storage. Figure 3-11 shows the main technologies that will be used.

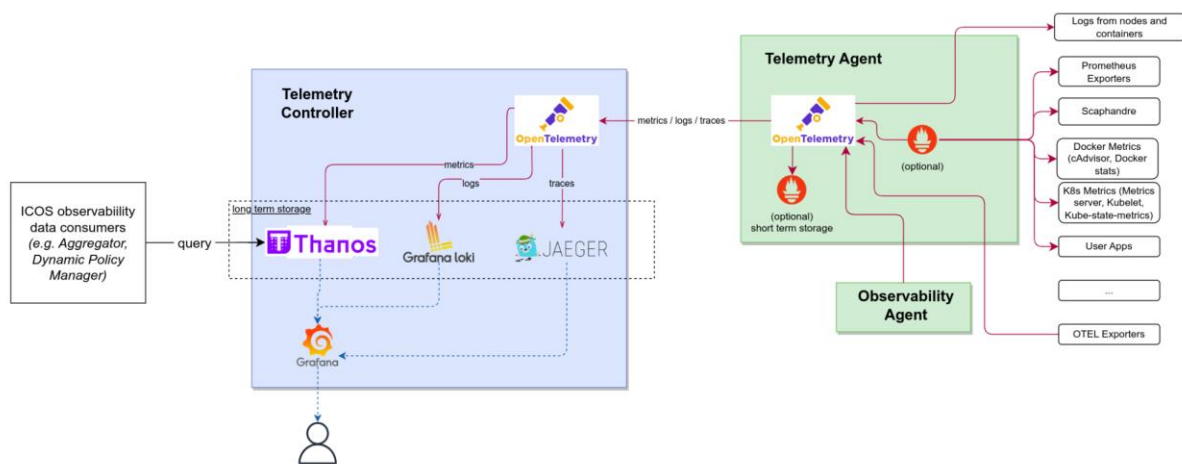


Figure 3-11 Composition of the technologies used for the implementation of the Logging and Telemetry Layer.

The main reference for the design and implementation of the telemetry components in ICOS is the OpenTelemetry project<sup>15</sup>. OpenTelemetry is a Cloud Native Computing Foundation (CNCF) project that was born from the fusion of previous telemetry projects with the aim of unifying and standardising how telemetry data (i.e., metrics, logs and traces) is modelled, collected and transferred in a vendor- and tool-agnostic way. The ICOS telemetry components use the **OpenTelemetry Collector**<sup>16</sup> which is a proxy that can receive, process, and export telemetry data. It supports multiple receiving and sending formats

<sup>15</sup> <https://opentelemetry.io/>

<sup>16</sup> <https://opentelemetry.io/docs/collector/>



(for example, OTLP, Jaeger, Prometheus, as well as many commercial/proprietary tools). It also supports processing and filtering telemetry data before it is exported. The support for multiple data and backend formats has been the main reason for choosing this technology, since it allows ICOS to be very flexible in the choice of other technologies to generate, manage and consume telemetry data. The OpenTelemetry Collector is at the core of the Telemetry Agent and is used to collect data from metrics endpoints as well as other Telemetry Agents, to process the data (i.e., aggregation, filtering and labelling) and to send the data to the Telemetry Controller.

The Telemetry Agent also support **Prometheus**<sup>17</sup> to store metrics locally (for caching or analysis reasons) and to collect metrics from Prometheus servers. The latter case can be useful (although not required) in deployment scenarios where metrics are already collected using Prometheus. In this case ICOS can obtain metrics from Prometheus directly without collecting them again.

For the long-term storage of telemetry data in the Controller, three main technologies have been selected:

- ▶ **Thanos**<sup>18</sup> for storing metrics. It is based on and fully compatible with Prometheus and offers interesting features like unlimited retention of metrics and compaction of historical data;
- ▶ **Loki**<sup>19</sup> for storing logs. It has been chosen for its compatibility with the other tools and its ease of usage;
- ▶ **Jaeger**<sup>20</sup> for storing traces (although it will not be used and deployed in the scope of IT-1).

Finally, **Grafana**<sup>21</sup> is used for visualising the data from long-term storage. All the tools selected are open-source and provide functionalities at the state of the art. In addition, they are very popular and with a good support from the respective communities.

### 3.4.1 Metrics Collected

The Telemetry Agent collects metrics from multiple sources specialised to monitor different aspects of the system. These sources are implemented from different components (third-party and developed by ICOS) that are deployed and configured by the telemetry agents. In the following subsections, the different types of metrics that were collected and the components used to generate them are presented.

#### 3.4.1.1 Topology metrics

The Aggregator component uses the telemetry data to keep an updated structure and status of the infrastructures connected to an ICOS Controller. The main components, at node site, that generates metrics used for this purpose are the following:

- ▶ **Kube State Metrics** and OpenTelemetry's **Kubernetes Receiver** for Kubernetes clusters;
- ▶ The **Node Feature Discovery Plugin** which is a Kubernetes add-on for detecting hardware features and system configuration by providing custom labels for detected devices attached to edge nodes controlled by OCM. The plugin allows for integration with Prometheus and other telemetry frameworks;
- ▶ The Telemetry Agent itself adds metadata about the topology of the infrastructures (e.g., node IDs, cluster names);
- ▶ Metrics for the device discovery processes on the node are exported in the telemetry data. The information about peripheral devices attached to the Edge node, in the Nuvla case, is contained in the Nuvla Orchestrator and will be exported in a future iteration.

<sup>17</sup> <https://prometheus.io/>

<sup>18</sup> <https://thanos.io/>

<sup>19</sup> <https://grafana.com/oss/loki/>

<sup>20</sup> <https://www.jaegertracing.io/>

<sup>21</sup> <https://grafana.com/>

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	41 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

### 3.4.1.2 Resource Usage Metrics

Metrics on the usage of resources and performance of the system and the application running in the nodes are essential to ICOS to understand the behaviour of the system and optimise it. While in IT-1 these metrics are not consumed by any component, in IT-2 they will be used by the Dynamic Policy Manager to evaluate performance policies (and trigger remediation actions). The main components deployed in the Telemetry Agent that will collect this type of metrics are:

- ▶ The Prometheus' **Node Exporter** that monitor the node and provides detailed data about resources usage (i.e., cpu, memory, disk, network, processes);
- ▶ The **cAdvisor** tool that provides resource usages for each container running in a node (both for Kubernetes and Docker nodes);

### 3.4.1.3 Eco Efficiency Metrics

As part of Telemetry capabilities, Scaphandre was employed to allow energy consumption monitoring across multiple ICOS nodes. Scaphandre comprises an open-source project designed for monitoring and optimising the consumption of resources in Kubernetes clusters. It provides insights and collections of metrics related to resource utilisation, thus facilitating the management and the optimisation of the infrastructure.

Key features and capabilities of Scaphandre for resource consumption monitoring in a Kubernetes environment may include:

- ▶ **Resource Metrics:** Scaphandre collects and provides detailed metrics on CPU, memory, and other resource usage across nodes, pods, and containers within a Kubernetes cluster.
- ▶ **Real-time Monitoring:** It offers real-time monitoring and reporting capabilities, allowing parties directly concerned to stay informed about the current state of resource consumption.
- ▶ **Historical Data:** Scaphandre often retains historical data, enabling trend analysis and the identification of resource usage patterns over time.
- ▶ **Alerting:** It may include alerting mechanisms to notify administrators when resource consumption exceeds predefined thresholds, helping prevent performance issues or outages.
- ▶ **Efficiency Recommendations:** Some implementations of Scaphandre may offer recommendations for optimising resource allocation, which can help reduce costs and improve cluster efficiency.
- ▶ **Integration:** It can integrate with other monitoring and observability tools, making it easier to incorporate Kubernetes resource consumption data into existing infrastructure management workflows.

In summary, Scaphandre is a tool that assists Kubernetes users in gaining better visibility into how their clusters are using resources, making it easier to allocate resources effectively, troubleshoot performance issues, and optimise infrastructure costs.

When combined with Prometheus and Grafana, Scaphandre provides a powerful solution for real-time cluster observability, performance optimization, and cost reduction. By developing the triplet of Prometheus, Grafana and Scaphandre, the following interaction loop was enabled: first, Scaphandre records the energy values consumed by an application-specific pod and, then, Prometheus is able to grasp data from Scaphandre's endpoints (via Prometheus node exporters). The collected energy consumption values are stored in a time-series database, making it readily accessible for querying and analysis. Finally, Grafana was used as a data visualisation and alerting tool, so as to complement Prometheus functionalities. The goal of Grafana was to offer a rich and customizable interface for visualising data and creating dashboards. In later stages, Grafana can be also used for alerting purposes, thus enabling timely notifications of resource consumption anomalies or predefined thresholds.

The above-mentioned setup was implemented at NKUA mini-lab premises, which are composed of 3 servers (2 servers act as ICOS nodes/agents and 1 act as ICOS controller). The Figure 3-12 depicts the developed stack of Scaphandre, Prometheus, and Grafana, as implemented at the NKUA lab. Evidently,

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	42 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

to ensure accurate data collection, Scaphandre and Prometheus node exporters have been deployed at all servers, whereas Prometheus Server has been deployed only on ICOS node 1 to gather the metrics from the Prometheus Node Exporters and serve them at Grafana.

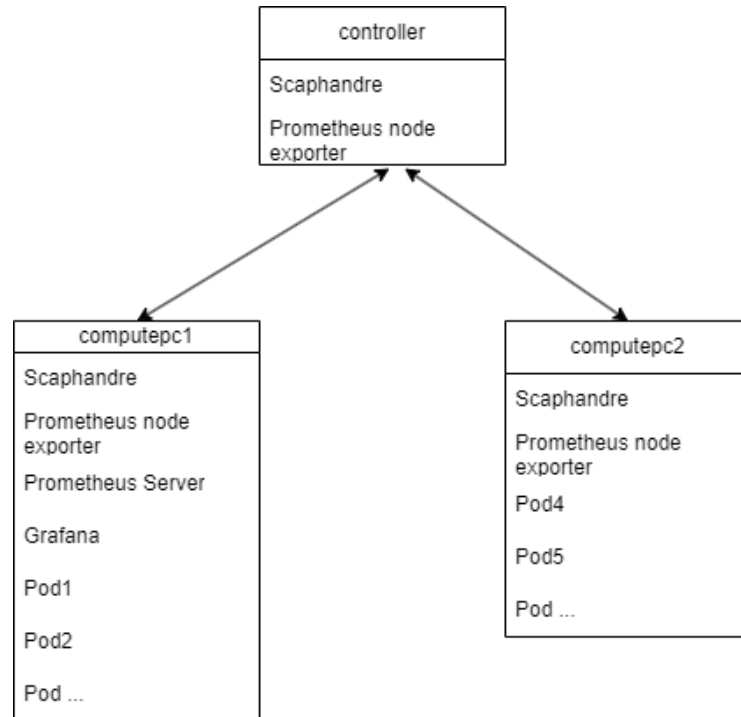


Figure 3-12 Developed stack of Scaphandre, Prometheus, and Grafana, as implemented at the NKUA lab.

The Figure 3-13 shows an example of the recorded energy consumption curve, with the consumption values being gathered via Scaphandre/Prometheus and visualised by Grafana. Overall, by integrating Scaphandre, Prometheus, and Grafana, some key monitoring abilities can be identified as:

- ▶ **Real-time Visibility:** Kubernetes operators gain real-time visibility into resource consumption, proactively addressing performance bottlenecks and optimising resource allocation.
- ▶ **Historical Data Analysis:** The combination of Prometheus's time-series database (InfluxDB will also be used) and Grafana's visualisation capabilities allows for historical data analysis, aiding in the identification of resource consumption patterns and trends (DeCOFFEE input).
- ▶ **Efficient Resource Management:** By continuously monitoring resource consumption, informed decisions can be made to optimise resource allocation, reducing infrastructure costs and minimising resource wastage.
- ▶ **Scalability:** This monitoring stack is highly scalable, making it suitable for monitoring both small-scale and large-scale Kubernetes clusters.
- ▶ **Customised Dashboards:** Grafana's flexibility enables the creation of customised dashboards tailored to specific use cases or requirements, ensuring that relevant metrics are easily accessible, like the image below:

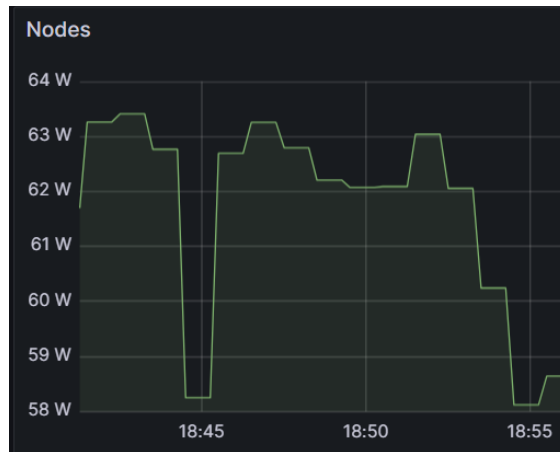


Figure 3-13 Recorded energy consumption curve of a node.

In addition to utilising Grafana's graphical user interface (UI) for accessing metrics, an alternative approach involves querying the Prometheus Server endpoint, which is exposed for data retrieval. These queries necessitate specific parameters, including the name of the Scaphandre metric, the start and end times for monitoring, expressed in absolute Unix epoch time (i.e., seconds since January 1, 1970), and the temporal step size for data points, also specified in seconds. Depending on the metric in question, additional parameters may be either mandatory or optional.

To illustrate this process, an experimental investigation was conducted at the NKUA mini-lab. This investigation aimed to monitor the power consumption associated with a model training operation. The model training operation was containerized and subsequently deployed within NKUA's Kubernetes cluster. Subsequently, queries were made to the Prometheus Server, which was exposed at the following URL: <http://192.168.1.33:30091/>. These queries were defined with specific query parameters, as illustrated in the forthcoming figures (referred to as Figure 3-14 below).

```

X Headers Payload Preview Response Initiator Timing Cookies
▼Query String Parameters view parsed
query=scaph_process_power_consumption_microwatts%7Bkubernetes_pod_name%3D%22lstm-setup-comparison-deploy-66d79dd954-snzpx%22%7D&start=1697560454.91&end=1697564054.91&step=14
X Headers Payload Preview Response Initiator Timing Cookies
▼Query String Parameters view source view decoded
query: scaph_process_power_consumption_microwatts%7Bkubernetes_pod_name%3D%22lstm-setup-comparison-deploy-66d79dd954-snzpx%22%7D
start: 1697560454.91
end: 1697564054.91
step: 14
X Headers Payload Preview Response Initiator Timing Cookies
▼Query String Parameters view source view URL-encoded
query: scaph_process_power_consumption_microwatts{kubernetes_pod_name="lstm-setup-comparison-deploy-66d79dd954-snzpx"}
start: 1697560454.91
end: 1697564054.91
step: 14

```

Figure 3-14 Query parameters to request 'scaph\_process\_power\_consumption\_microwatts' metric for specific pods.

Specifically, the objective was to retrieve the 'scaph\_process\_power\_consumption\_microwatts' metric pertaining to the particular pod that hosted the model training operation. As part of the query parameters, it was essential to specify the commencement and conclusion times for data collection, as well as the time intervals between successive data points, all in seconds. Additionally, the 'kubernetes\_pod\_name,' which uniquely identified the relevant Pod within our application, had to be provided as a query

Document name:	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	Page:	44 of 79
Reference:	D3.1	Dissemination:	PU
	Version:	1.0	Status:
			Final

parameter. The response received from this query was presented in JSON format and is visualised in Figure 3-15.

```

X Headers Payload Preview Response Initiator Timing Cookies
1 {
-   "status": "success",
-   "data": {
-     "resultType": "matrix",
-     "result": [
-       {
-         "metric": {
-           "__name__": "scaph_process_power_consumption_microwatts",
-           "app_kubernetes_io_managed_by": "Helm",
-           "app_kubernetes_io_name": "scaphandre",
-           "cmdline": "pythonLSTM_Setup_Comparison.py",
-           "container_id": "5774d5c4cb474994fae673ed61d394e7490dfd6e48ed157231310c5133202d16",
-           "container_scheduler": "kubernetes",
-           "exe": "python",
-           "instance": "10.1.235.115:8080",
-           "job": "kubernetes-service-endpoints",
-           "kubernetes_node_name": "compute2",
-           "kubernetes_pod_name": "lstm-setup-comparison-deploy-66d79dd954-snzpx",
-           "kubernetes_pod_namespace": "default",
-           "namespace": "default",
-           "node": "compute2",
-           "pid": "4100682",
-           "service": "scaphandre"
-         },
-         "values": [
-           [
-             1697561550.966,
-             "51683106"
-           ],
-           [
-             1697561564.966,
-             "51683106"
-           ],
-           [
-             1697561578.966,
-             "51683106"
-           ],
-           [
-             1697561592.966,
-             "51683106"
-           ],
-           [
-             1697561606.966,
-             "51805822"
-           ]
-         ]
-       }
-     ]
-   }
- }

```

Figure 3-15 JSON response with 'scaph\_process\_power\_consumption\_microwatts' metric.

### System and Application performance Metrics

The Telemetry Agent implements a mechanism to automatically discover system services and user applications running in the node that expose metrics and collect those metrics on a regular basis. This generic mechanism allows an easy integration of the ICOS Telemetry system with any source of metric. Metrics collected in this way can be used, in the same way as system metrics, to define performance policies and constraints that will influence the behaviour of ICOS. In IT-1, the discovery mechanism is delivered and in IT-2 it will be exploited by system and user services.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	45 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

## 4 Development and Validation

### 4.1 Development and Integration Processes and Tools

The development and integration strategy and tools adopted to realise the ICOS platform, is provided in the deliverable “D2.2 ICOS Architecture and Design (IT-1)” and in accordance with the scope of the IT-1.

The Table 2 below is an updating of the initial mapping of the architectural components provided in the deliverable “D2.2 – Architectural and Design (IT-1)” (p75 , Table 8: Initial mapping of all architectural components).

The table provides only components that will be implemented for the ICOS Alfa release at M15 (November 2023) with their respective owner, the list of (possible) sub-components for each component and the partners responsible for the implementation and list of technologies (if necessary) adopted to support module development.

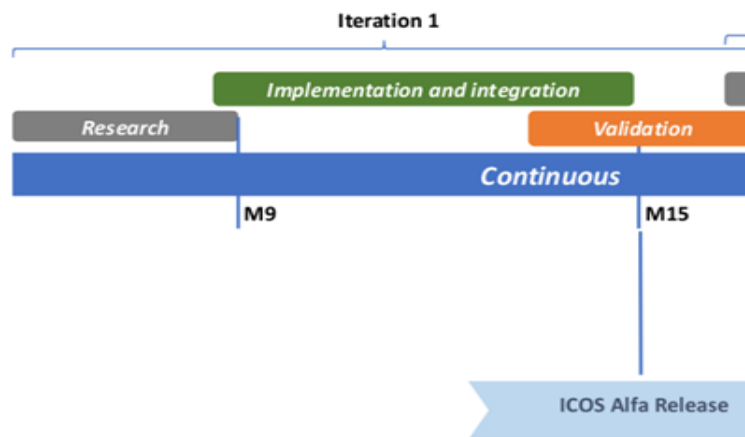


Figure 4-1 Snapshot of the ICOS System Release schedule around IT-1 (source D2.2).

For the first release of the ICOS not all functionalities were implemented for the Meta-Kernel component but only a subset of them and related to the components implemented (and shown in the Table 2):

- ▶ Node On-Boarding
- ▶ Application Descriptor Definition
- ▶ Basic Application Deployment
- ▶ Collect and visualise Metrics and Logs

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	46 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

Table 2 ICOS Meta-Kernel components, responsible partner, and selected technology.

Component (Owner)	Sub-Component	Responsible Partners	Technology/Assets mapping
LightHouse (TUBS)		TUBS	Lighthouse; HTTP server in Golang
Continuum Manager (SixSq)	Resource and Clustering Manager	SixSq	Nuvla <a href="https://nuvla.io/">https://nuvla.io/</a>
		ATOS	OCM
Runtime Manager (BSC)	Aggregator	ATOS	HTTP server in Golang
	Matchmaking	UPC	FastAPI
	Job Manager	ATOS	HTTP server in Golang
	Distributed and Parallel execution	BSC	COMPS
	Workload offloader	BSC	COMPS
Logging and Telemetry (ENG)	Telemetry Agent		OpenTelemetry, Thanos, Jaeger, Loki, Grafana

The tools used for the Continuous Integration and Continuous Delivery is GitLab provided from Enginenering (ENG): [ICOS / Support · GitLab \(eng.it\)](https://production.eng.it).

In GitLab a dedicated project called MetaKernel (<https://production.eng.it/gitlab/icos/meta-kernel>) is created. It consists of a set of sub-projects (shown in Figure 4-2): the ICOS Metakernel ICOS components.

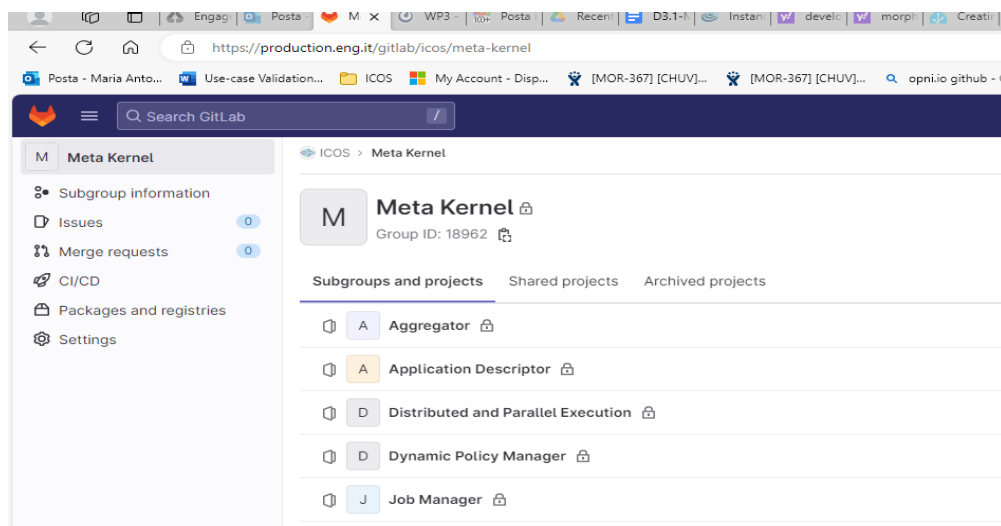


Figure 4-2 ICOS Project source code repository using GitLab.

The developers upload their code every time a new implementation of their components is realised.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	47 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

More details about the use of GitLab will be provided in the deliverable “D5.1 First ICOS Release: ICOS Alfa version”.

For the integration of these tools it has been decided to use Docker for the containerization and Kubernetes for the clustering as runtime platforms to run the ICOS MetaKernel module, because the technologies used to implement this module adopt docker and Kubernetes as baseline technologies.

For the Beta Release two additional tools (SonarQube, <https://www.sonarsource.com/products/sonarqube> and Harbor, <https://goharbor.io/>) will be activated for the protection of artefacts, bug detection, vulnerabilities and quality of the code.

## 4.2 Validation

The following Test Cases were designed for the validation of the functional capabilities of the ICOS Meta-Kernel layer, in order to support the implementation of the System Use Cases (that are part of the ICOS Alfa Release) with the scope of IT-1 delivery. The Test Cases defined below are of multiple types - some are validating components in isolation however, some are defined on a system and integration levels.

### 4.2.1 Node On-Boarding

The following validation Test Cases demonstrate the edge and cloud resources onboarding capabilities of the selected Cloud and Edge Orchestrators (*continuum.orchestrator-edge-cloud*) OCM and Nuvla.

#### Test Case - Onboard edge resource in Nuvla

Prerequisites:

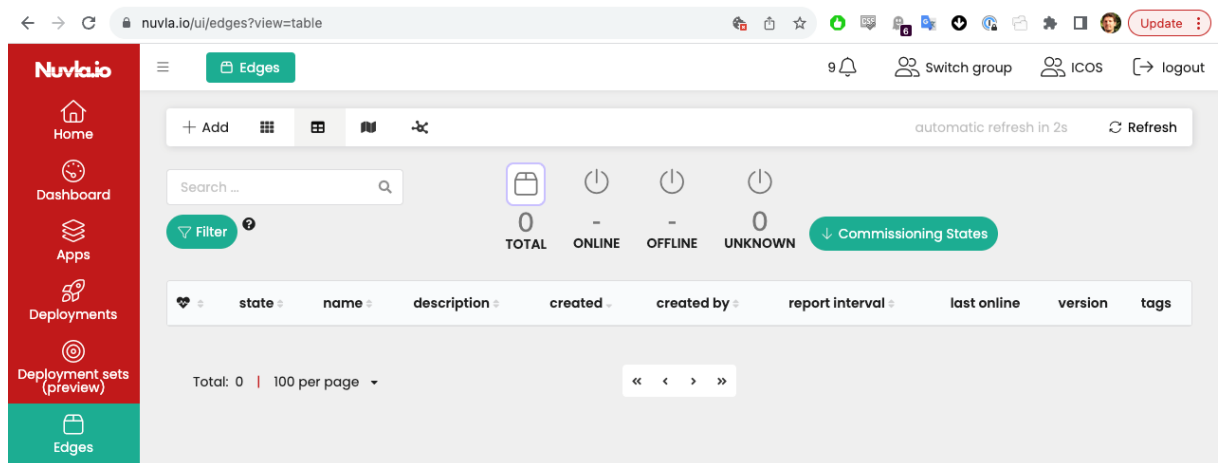
- ▶ Edge: device with Kubernetes COE (e.g. k3s) installed; edge operator with (remote) access to the console of the edge devices.
- ▶ Nuvla: edge operator with account in <https://nuvla.io>

Expected result:

- ▶ Edge device visible on <https://nuvla.io> as commissioned and in online state.

#### Test Record – [PASSED]

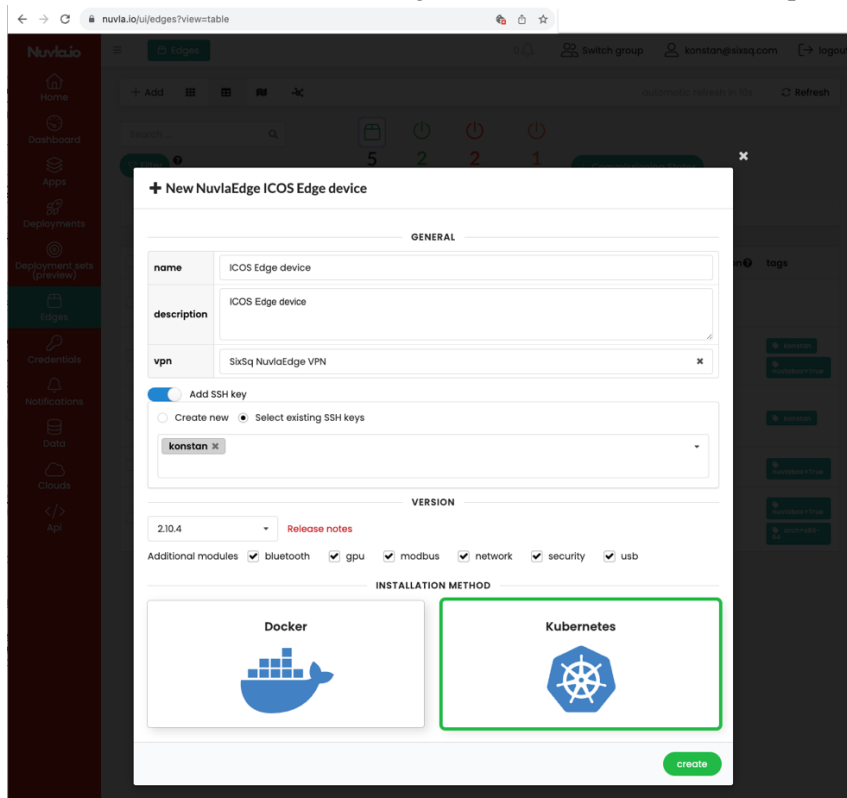
- ▶ Operator logs in to Nuvla.io Platform <https://nuvla.io> and navigates to Edge tab.



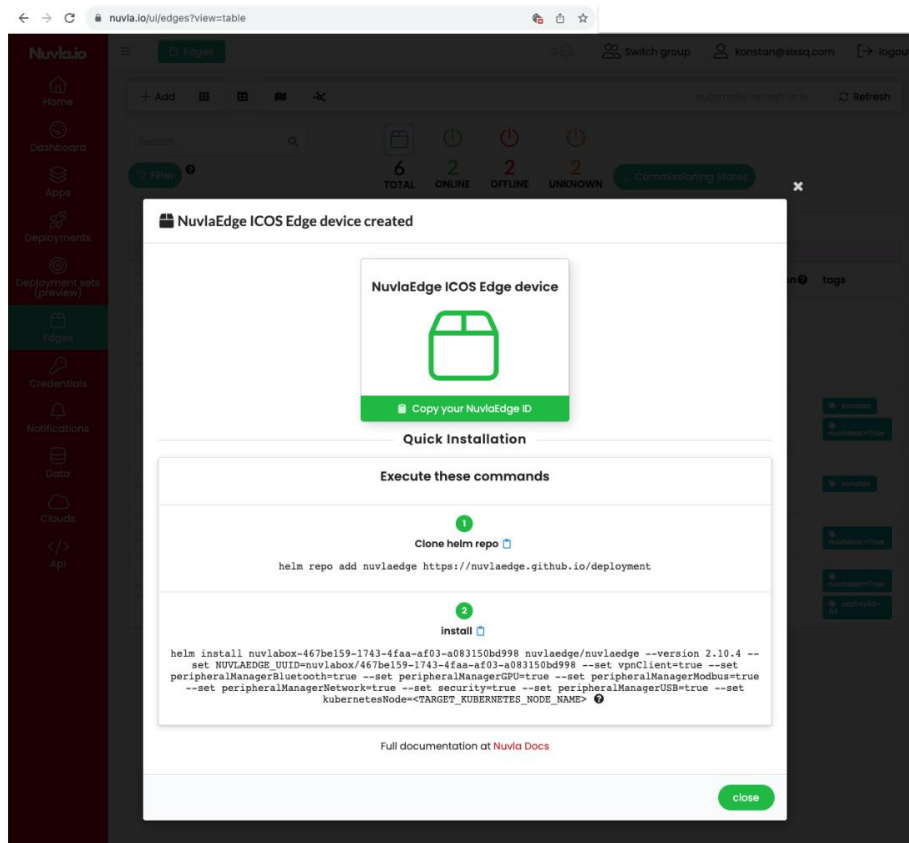
<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	48 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



► Operator creates the Kubernetes COE based Edge device reference in Nuvla.io platform

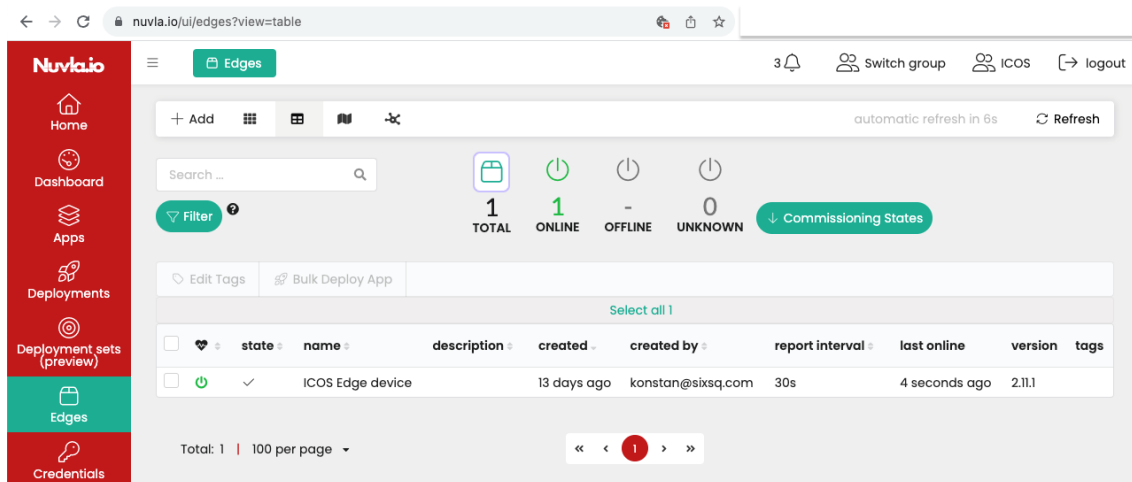


► Operator is presented with Helm command to provision the NuvlaEdge on the Edge device



Document name:	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	Page:	49 of 79
Reference:	D3.1	Dissemination:	PU
	Version:	1.0	Status:
			Final

- ▶ After NuvlaEdge is deployed on the Edge device, it connects to Nuvla.io and commissions itself into an operational state.



### Test Case - Onboard edge or cloud resource in OCM

#### Prerequisites:

- ▶ Edge/Cloud: device with Kubernetes COE (e.g., k3s) installed; edge operator with (remote) access to the console of the edge devices.
- ▶ OCM: operator with account in OCM

#### Expected result:

- ▶ Cluster visible in OCM as commissioned and in online state.
- ▶ Clusteradm tool installed within the Edge and the ICOS Controller.

#### Test Record – [PASSED]

- ▶ Operator logs into the Edge and triggers the on-boarding process by executing Join request:

```
clusteradm join \
  --hub-token <your token data> \
  --hub-apiserver <your hub cluster endpoint> \
  --wait \
  --cluster-name "cluster1" \ # Or other arbitrary unique name
  --context ${CTX_MANAGED_CLUSTER}
```

Once the request is accepted by the ICOS Controller the Resource will appear as follows:

```
NAME          HUB ACCEPTED  MANAGED CLUSTER URLS  JOINED  AVAILABLE
k3s-worker2   true         true                   True    True
k3s-worker1   true         true                   True    True
```

For more information: <https://open-cluster-management.io/getting-started/installation/register-a-cluster/#bootstrap-a-kuberneteslet>

Document name:	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	Page:	50 of 79
Reference:	D3.1	Dissemination:	PU
	Version:	1.0	Status:
			Final

**Test Case - Onboard cloud Kubernetes COE resource in Nuvla**

**Prerequisites:**

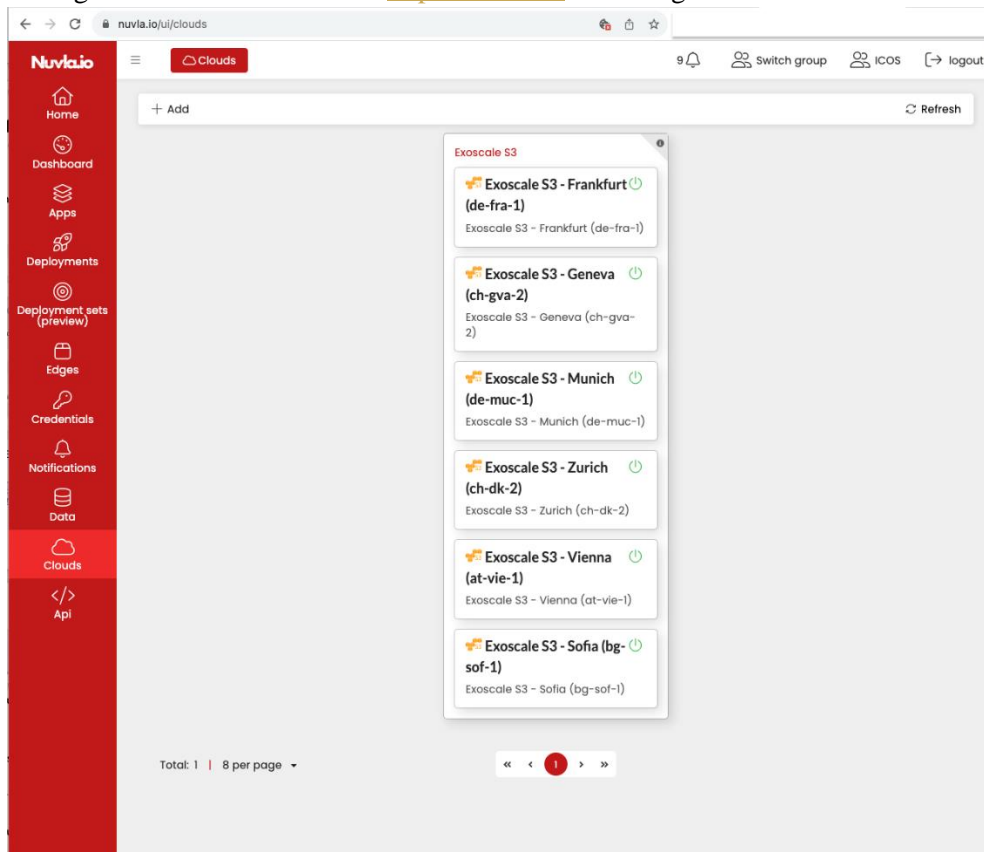
- ▶ Cloud: VM with Kubernetes COE cluster installed; cloud operator with (remote) access to the console of the VM with Kubernetes master node.
- ▶ Nuvla: cloud operator with account in <https://nuvla.io>

**Expected result:**

- ▶ Kubernetes COE cluster is visible in <https://nuvla.io> in online state.

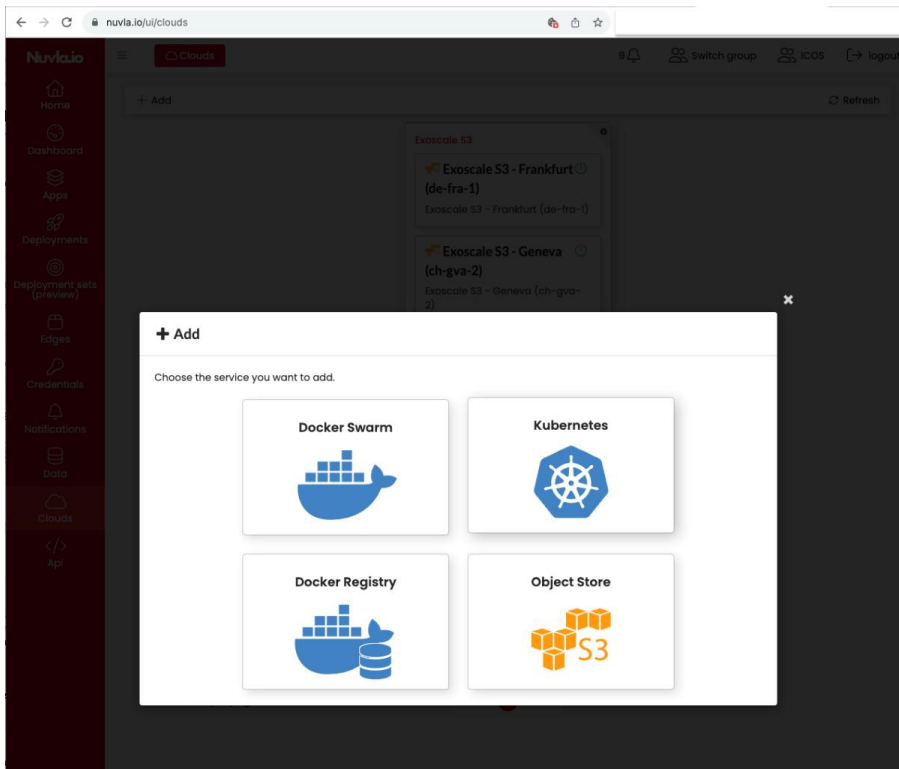
**Test Record – [PASSED]**

- ▶ Operator logs in to Nuvla.io Platform <https://nuvla.io> and navigates to Cloud tab.

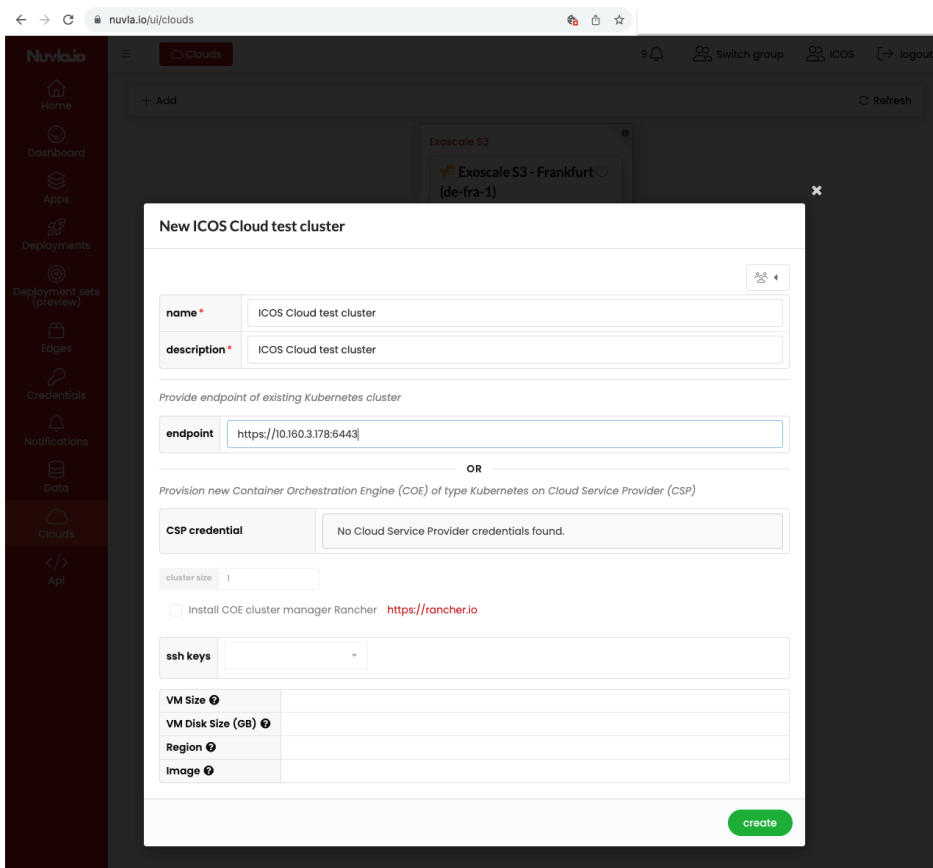


- Operator clicks on +Add button and selects Kubernetes in the presented modal window

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	51 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

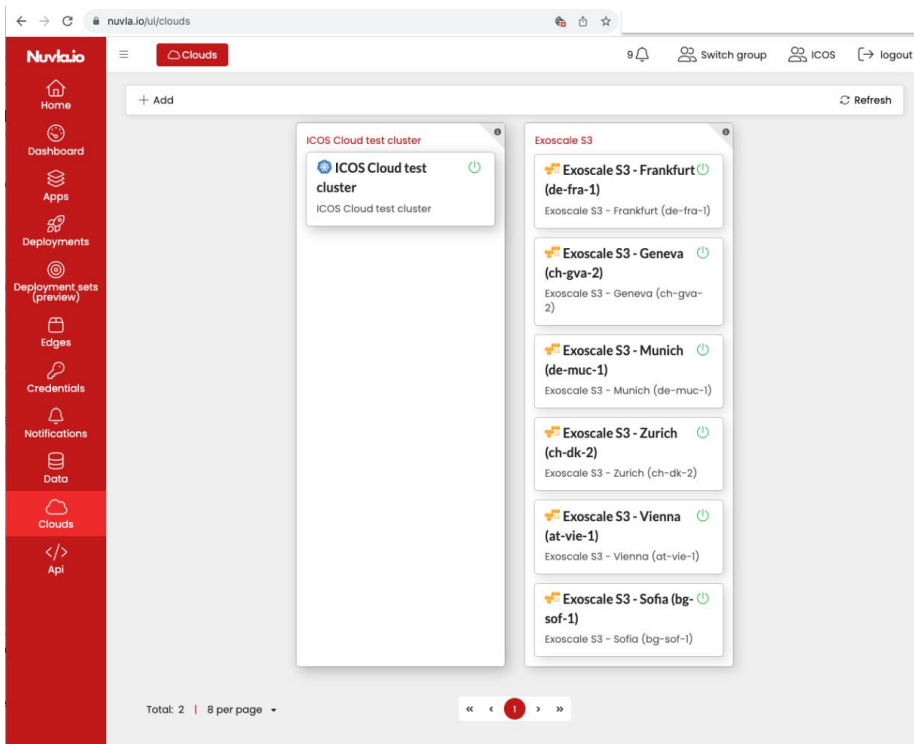


► Operator provides name, description and the endpoint of the Kubernetes cluster. Then, clicks green *create* button.

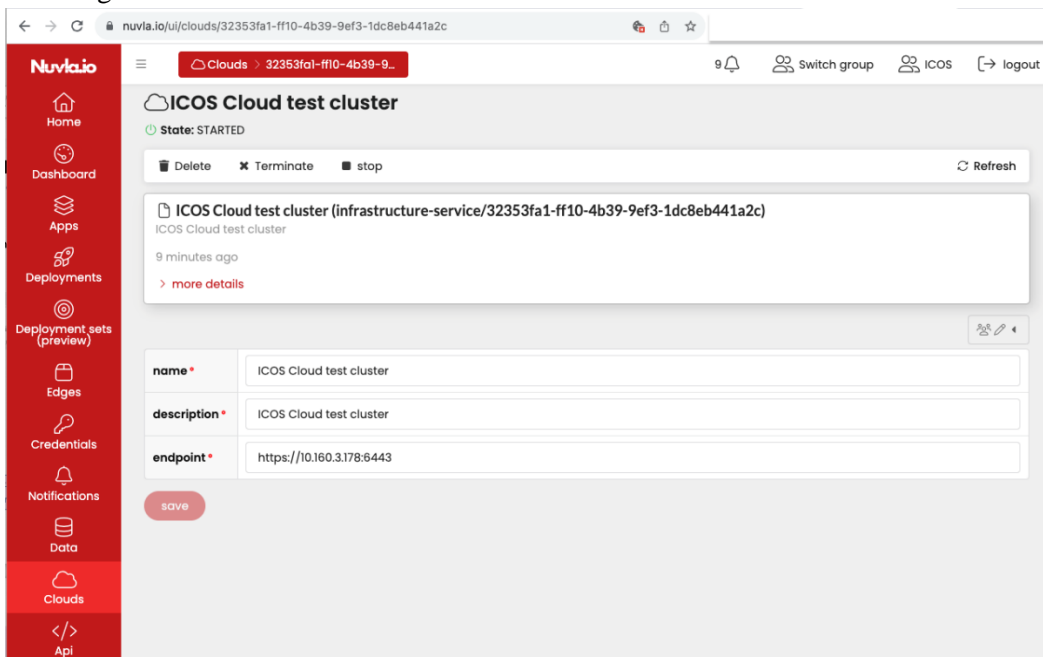


► Operator is presented with the new cluster reference on the Cloud tab.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	52 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



► Operator can get detailed information about the created Kubernetes cluster.



Document name:	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	Page:	53 of 79
Reference:	D3.1	Dissemination:	PU
	Version:	1.0	Status:
			Final

#### 4.2.2 Application Descriptor Definition and Basic Application Deployment

Deployment of synthetic applications with specific resource requirements on Cloud and Edge resources. Application Descriptor Definition will be part of each application deployment, hence, is part of the Application Deployment validations.

##### **Test Case - App deployment at the edge with Open-Cluster-Management Application Descriptor.**

Prerequisites:

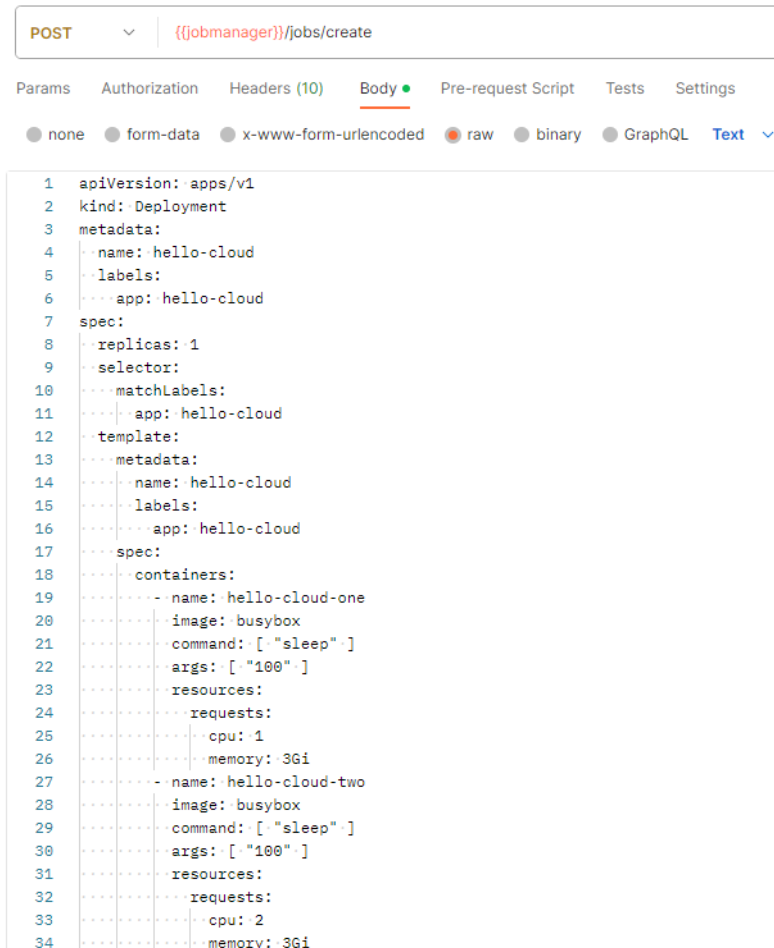
- ▶ ICOS Controller: the ICOS Controller endpoint is available, Cloud and Edge resources are present and available.
- ▶ User Application: simple Hello World application where the only requirement is it must be deployed to an Edge resource.

Expected result:

- ▶ The Application is deployed to an available Edge resource manager by Open-Cluster-Management Orchestrator.

##### Test Record - [PASSED]

- ▶ Create a Deployment Job: POST /jobmanager/jobs/create



```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: hello-cloud
5    labels:
6      app: hello-cloud
7  spec:
8    replicas: 1
9    selector:
10     matchLabels:
11       app: hello-cloud
12   template:
13     metadata:
14       name: hello-cloud
15       labels:
16         app: hello-cloud
17     spec:
18       containers:
19         - name: hello-cloud-one
20           image: busybox
21           command: [ "sleep" ]
22           args: [ "100" ]
23           resources:
24             requests:
25               cpu: 1
26               memory: 3Gi
27         - name: hello-cloud-two
28           image: busybox
29           command: [ "sleep" ]
30           args: [ "100" ]
31           resources:
32             requests:
33               cpu: 2
34               memory: 3Gi

```

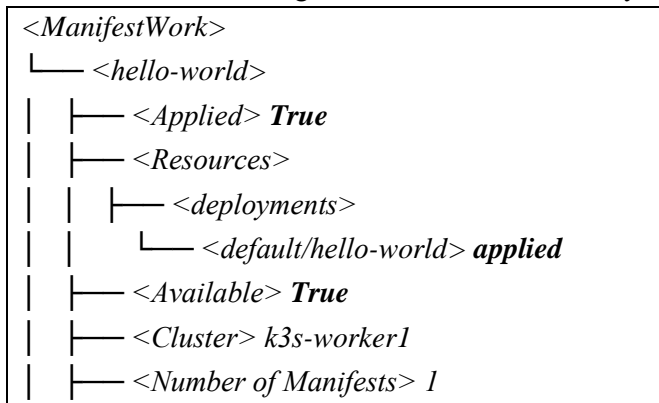
<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	54 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

► The created Job:

```
{
  "ID": "5f9bff8c-f334-46d7-9780-47b40a7c96bd",
  "type": 5,
  "state": 1,
  "manifest": "apiVersion: apps/v1\r\nkind: Deployment\r\nmetadata:\r\n  name: hello-cloud\r\n  labels:\r\n    app: hello-cloud\r\nspec:\r\n  replicas: 1\r\n  selector:\r\n    matchLabels:\r\n      app: hello-cloud\r\n  template:\r\n    metadata:\r\n      name: hello-cloud\r\n      labels:\r\n        app: hello-cloud\r\n    spec:\r\n      containers:\r\n        - name: hello-cloud-one\r\n          image: busybox\r\n          command: [ \"sleep\" ]\r\n          args: [ \"100\" ]\r\n          resources:\r\n            requests:\r\n              cpu: 1\r\n              memory: 3Gi\r\n        - name: hello-cloud-two\r\n          image: busybox\r\n          command: [ \"sleep\" ]\r\n          args: [ \"100\" ]\r\n          resources:\r\n            requests:\r\n              cpu: 2\r\n              memory: 3Gi",
  "targets": [
    {
      "id": 4,
      "JobID": "5f9bff8c-f334-46d7-9780-47b40a7c96bd",
      "cluster_name": "10.42.0.85:8080",
      "node_name": "k3s-worker1"
    }
  ],
  "locker": false,
  "updatedAt": "2023-10-17T08:50:06.846Z"
}
```

Note: it should be noticed that the targets are properly populated by the matchmaking service during the deployment job creation.

- [runtime.deployment-manager \[ATOS\]](#) works in pull mode to retrieve the previously created job from the [runtime.job-manager \[ATOS\]](#). Once the job is pulled and validated it can be executed by Open Cluster Management Orchestrator as “ManifestWork”.
- Since Open Cluster Management Agent within the Edge Node also works in pull mode, it is able to notice mentioned ManifestWork and apply it properly on the local cluster.
- As a result, the application is deployed within the Edge cluster called “10.42.0.85:8080” and node “k3s-worker1”. The following chart shows the availability of the deployment:



**Test Case - App deployment degenerate case**

Prerequisites:

- ICOS Controller: the ICOS Controller endpoint is available but doesn't have any Edge or Cloud resources available.
- User Application: simple Hello World application without any requirements.

Expected result:

- ICOS Controller is not able to deploy the application since no target is found and after some time sets the state of the Job as Degraded(4).

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	55 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

Test Record - [PASSED]

► Create a Deployment Job: POST /jobmanager/jobs/create

```

POST  ({{jobmanager}})/jobs/create

Params  Authorization  Headers (10)  Body  Pre-request Script  Tests  Settings
● none  ● form-data  ● x-www-form-urlencoded  ● raw  ● binary  ● GraphQL  Text  ▾

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4  .. name: hello-cloud
5  .. labels:
6  ... app: hello-cloud
7  spec:
8  .. replicas: 1
9  .. selector:
10  ... matchLabels:
11  .... app: hello-cloud
12  .. template:
13  ... metadata:
14  .... name: hello-cloud
15  .... labels:
16  ..... app: hello-cloud
17  ... spec:
18  .... containers:
19  ..... - name: hello-cloud-one
20  ..... image: busybox
21  ..... command: [ "sleep" ]
22  ..... args: [ "100" ]
23  ..... resources:
24  ..... requests:
25  ..... cpu: 1
26  ..... memory: 3Gi
27  ..... - name: hello-cloud-two
28  ..... image: busybox
29  ..... command: [ "sleep" ]
30  ..... args: [ "100" ]
31  ..... resources:
32  ..... requests:
33  ..... cpu: 2
34  ..... memory: 3Gi

```

► The created Job:

```

{
  "ID": "5f9bff8c-f334-46d7-9780-47b40a7c96bd",
  "type": 5,
  "state": 1,
  "manifest": "apiVersion: apps/v1\r\nkind: Deployment\r\nmetadata:\r\n name: hello-cloud\r\n labels:\r\n  app: hello-cloud\r\nspec:\r\n replicas: 1\r\n selector:\r\n  matchLabels:\r\n    app: hello-cloud\r\n template:\r\n  metadata:\r\n    name: hello-cloud\r\n    labels:\r\n      app: hello-cloud\r\n spec:\r\n   containers:\r\n    - name: hello-cloud-one\r\n      image: busybox\r\n      command: [ \"sleep\" ]\r\n      args: [ \"100\" ]\r\n      resources:\r\n        requests:\r\n          cpu: 1\r\n          memory: 3Gi\r\n    - name: hello-cloud-two\r\n      image: busybox\r\n      command: [ \"sleep\" ]\r\n      args: [ \"100\" ]\r\n      resources:\r\n        requests:\r\n          cpu: 2\r\n          memory: 3Gi",
  "targets": [],
  "locker": false,
  "updatedAt": "2023-10-17T08:50:06.846Z"
}

```

Since no targets are returned by the Match making service, the state is updated to state=Degraded (4):

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	56 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



```
{
  "ID": "5f9bff8c-f334-46d7-9780-47b40a7c96bd",
  "type": 5,
  "state": 4,
  "manifest": "apiVersion: apps/v1\r\nkind: Deployment\r\nmetadata:\r\n  name: hello-cloud\r\n  labels:\r\n    app: hello-cloud\r\nspec:\r\n  replicas: 1\r\n  selector:\r\n    matchLabels:\r\n      app: hello-cloud\r\n  template:\r\n    metadata:\r\n      name: hello-cloud\r\n    labels:\r\n      app: hello-cloud\r\n    spec:\r\n      containers:\r\n        - name: hello-cloud-one\r\n          image: busybox\r\n          command: [ \"sleep\" ]\r\n          args: [ \"100\" ]\r\n          resources:\r\n            requests:\r\n              cpu: 1\r\n              memory: 3Gi\r\n            - name: hello-cloud-two\r\n              image: busybox\r\n              command: [ \"sleep\" ]\r\n              args: [ \"100\" ]\r\n              resources:\r\n                requests:\r\n                  cpu: 2\r\n                  memory: 3Gi",
  "targets": [],
  "locker": false,
  "updatedAt": "2023-10-17T08:50:06.846Z"
}
```

### 4.2.3 Distributed and Parallel Execution

The following test case validates the distributed execution of an application (means model training) using the ICOS' Distributed and Parallel execution manager. Besides this test, the component undergoes more than 250 unit tests that validate the proper running of specific functionalities of the component.

**Test Case** - Execution of a Kmeans training using a input distributed dataset

Prerequisites:

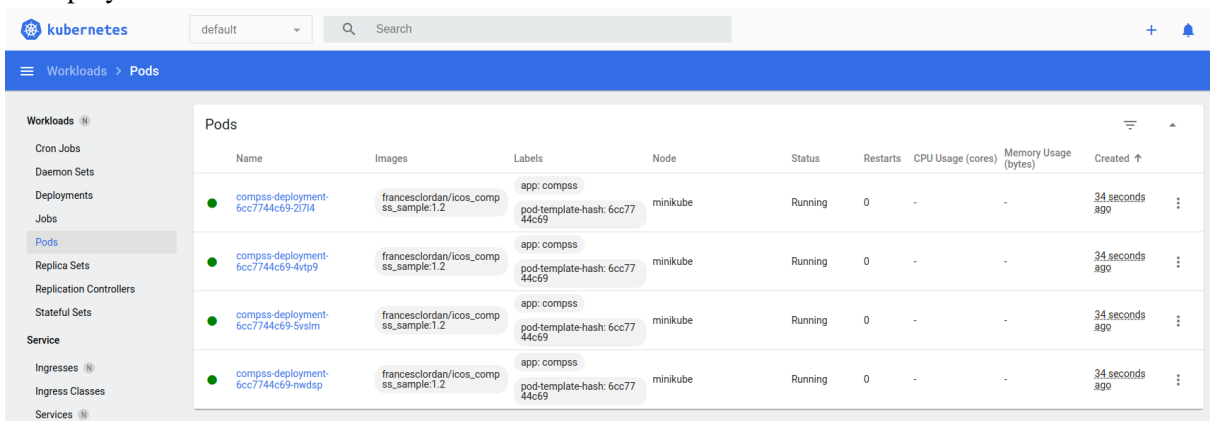
- ▶ Kubernetes cluster where to deploy a service
- ▶ Kmeans application published: docker image containing the Distributed and Parallel execution component and the user application (Kmeans)

Expected result:

- ▶ The service is able to detect the tasks composing the nested workflow of the required processing.
- ▶ Tasks are being submitted to the multiple containers belonging to the deployment

#### Test Record - [PASSED]

- ▶ Deploy the service containers



<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	57 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
<b>Version:</b>	1.0	<b>Status:</b>	Final

### ► Check initial log in the master

```

flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$ kubectl logs compss-deployment-6cc7744c69-21714
[ INFO ] Using default location for project file: /opt/COMPSS/Runtime/configuration/xml/projects/examples/local/project.xml
[ INFO ] Using default location for resources file: /opt/COMPSS/Runtime/configuration/xml/resources/examples/local/resources.xml
[ INFO ] Using default execution type: compss
[ INFO ] Options setup:
AGENT NAME: 10.244.1.176
AGENT REST PORT: 46101
AGENT COMM PORT: 46102

RESOURCES FILE: /opt/COMPSS/Runtime/configuration/xml/resources/examples/local/resources.xml
PROJECT FILE: /opt/COMPSS/Runtime/configuration/xml/projects/examples/local/project.xml
COMM ADAPTOR: es.bsc.compss.agent.comm.CommAgentAdaptor

DEBUG: off
PYTHON INTERPRETER: python3
MPI PYTHON: false
PYTHON CACHE: false
PYTHON PROFILER:
[ INFO ] -----
AGENT 10.244.1.176
-----
Launching COMPSS agent on device 10.244.1.176 and ports (rest: 46101) (comm: 46102) with debug level
WARNING: COMPSS Properties file is null. Setting default values
WARNING: COMPSS Properties file is null. Setting default values
log4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[[838] API] - Starting COMPSS Runtime v3.2.rc2309 (build 20230907-1354.rd3c1c7600352b586ffc85b940c76ff5a9611c94e)
Constructing configuration
Crating Config with port 46102
Initializing Worker
Constructing configuration
Crating Config with port 46102
Initializing Worker
Constructing configuration
Crating Config with port 46102
Initializing Worker
flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$

```

### ► Launch

application

```

flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$ kubectl exec compss-deployment-6cc7744c69-21714 -- train "/data/dataset_800000_100/0_0.npy,/data/dataset_800000_100/0_1.npy,/data/dataset_800000_100/0_2.npy,/data/dataset_800000_100/1_0.npy,/data/dataset_800000_100/1_1.npy,/data/dataset_800000_100/1_2.npy,/data/dataset_800000_100/2_0.npy,/data/dataset_800000_100/2_1.npy,/data/dataset_800000_100/2_2.npy,/data/dataset_800000_100/3_0.npy,/data/dataset_800000_100/3_1.npy,/data/dataset_800000_100/3_2.npy"
Calling service train with @/data/dataset_800000_100/0_0.npy,@/data/dataset_800000_100/0_1.npy,/data/dataset_800000_100/0_2.npy,/data/dataset_800000_100/1_0.npy,/data/dataset_800000_100/1_1.npy,/data/dataset_800000_100/1_2.npy,/data/dataset_800000_100/2_0.npy,/data/dataset_800000_100/2_1.npy,/data/dataset_800000_100/2_2.npy,/data/dataset_800000_100/3_0.npy,/data/dataset_800000_100/3_1.npy,/data/dataset_800000_100/3_2.npy
[ INFO ] Invocation details:
Host: 127.0.0.1
Port: 46101
Lang: PYTHON
Class name: kmeans
Method name: train
Parameters:
@/data/dataset_800000_100/0_0.npy,/data/dataset_800000_100/0_1.npy,/data/dataset_800000_100/0_2.npy,/data/dataset_800000_100/1_0.npy,/data/dataset_800000_100/1_1.npy,/data/dataset_800000_100/1_2.npy,/data/dataset_800000_100/2_0.npy,/data/dataset_800000_100/2_1.npy,/data/dataset_800000_100/2_2.npy,/data/dataset_800000_100/3_0.npy,/data/dataset_800000_100/3_1.npy,/data/dataset_800000_100/3_2.npy
518368043908154552
flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$

```

### ► Check final log in the master and verify the Job completion

```

flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$ kubectl logs compss-deployment-6cc7744c69-21714
[ INFO ] Using default location for project file: /opt/COMPSS/Runtime/configuration/xml/projects/examples/local/project.xml
[ INFO ] Using default location for resources file: /opt/COMPSS/Runtime/configuration/xml/resources/examples/local/resources.xml
[ INFO ] Using default execution type: compss
[ INFO ] Options setup:
AGENT NAME: 10.244.1.176
AGENT REST PORT: 46101
AGENT COMM PORT: 46102

RESOURCES FILE: /opt/COMPSS/Runtime/configuration/xml/resources/examples/local/resources.xml
PROJECT FILE: /opt/COMPSS/Runtime/configuration/xml/projects/examples/local/project.xml
COMM ADAPTOR: es.bsc.compss.agent.comm.CommAgentAdaptor

DEBUG: off
PYTHON INTERPRETER: python3
MPI PYTHON: false
PYTHON CACHE: false
PYTHON PROFILER:
[ INFO ] -----
AGENT 10.244.1.176
-----
Launching COMPSS agent on device 10.244.1.176 and ports (rest: 46101) (comm: 46102) with debug level
WARNING: COMPSS Properties file is null. Setting default values
WARNING: COMPSS Properties file is null. Setting default values
log4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[[838] API] - Starting COMPSS Runtime v3.2.rc2309 (build 20230907-1354.rd3c1c7600352b586ffc85b940c76ff5a9611c94e)
Constructing configuration
Crating Config with port 46102
Initializing Worker
Constructing configuration
Crating Config with port 46102
Initializing Worker
Constructing configuration
Crating Config with port 46102
Initializing Worker
Received REST call to run a PYTHON method
Job completed after 8731
flordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$

```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	58 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
<b>Version:</b>	1.0	<b>Status:</b>	Final

► Verify task executed in each container

```

fLordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$ for pod in $(kubectl describe rs $(rs) | grep "Created pod:" | awk '{print $7}'); do echo $pod; kubectl exec $pod -- "ls
~/tmp/test/jobs/" ; done
comps-deployment-6cc7744c69-2l7l4
job10_NEW_err
job10_NEW_out
job11_NEW_err
job11_NEW_out
job12_NEW_err
job12_NEW_out
job13_NEW_err
job13_NEW_out
job17_NEW_err
job17_NEW_out
job18_NEW_err
job18_NEW_out
job1_NEW_err
job1_NEW_out
job21_NEW_err
job21_NEW_out
job23_NEW_err
job23_NEW_out
job3_NEW_err
job3_NEW_out
job6_NEW_err
job6_NEW_out
job7_NEW_err
job7_NEW_out
job8_NEW_err
job8_NEW_out
job9_NEW_err
job9_NEW_out
comps-deployment-6cc7744c69-4vtp9
job1_NEW_err
job1_NEW_out
job2_NEW_err
job2_NEW_out
job3_NEW_err
job3_NEW_out
comps-deployment-6cc7744c69-nwdsp
job1_NEW_err
job1_NEW_out
job2_NEW_err
job2_NEW_out
job3_NEW_err
job3_NEW_out
comps-deployment-6cc7744c69-5vslm
job1_NEW_err
job1_NEW_out
job2_NEW_err
job2_NEW_out
job3_NEW_err
job3_NEW_out
fLordan@bsc-84885147:~/projects/ICOS/WP2/sample_app_matchmaking/k8s$
  
```

#### 4.2.4 Collect and visualise Metrics and Logs

The Test Cases for this System Use Case are revolving around the two following topics:

- Metrics and logs from monitored resources and applications can be visualised.
- Metrics and logs from monitored resources and applications are collected.

The test cases are executed on Kubernetes clusters. 3 nodes are tested : Nuvla k8s cluster, node-edge cluster and node-cloud cluster.

For each 3 nodes the metrics are collected from the OpenTelemetry Collector (ICOS Agent Collector) and sent to Thanos (ICOS Controller); the metrics are visualised from Grafana.

**Test Case** - Metrics and logs from monitored resources and applications can be visualised

Prerequisites:

- ICOS Controller and ICOS Agent up and running
- ICOS Controller deploys the default datasource Thanos for the resources and application

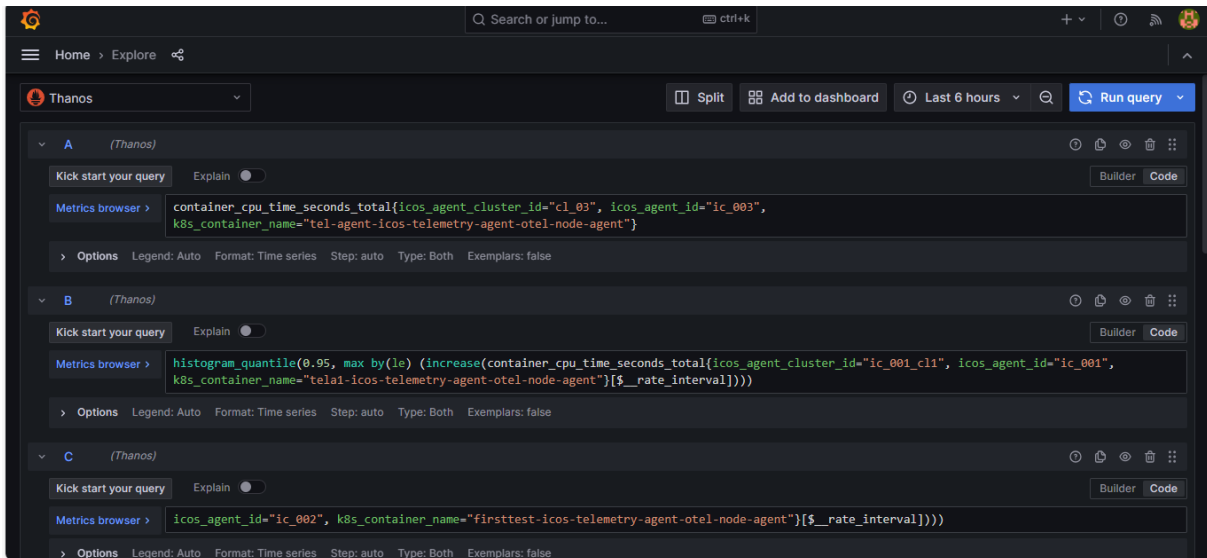
Expected result:

- All metrics are visualised in the Grafana dashboards.

#### Test Record - [PASSED]

1. All metrics are visualised in the Grafana dashboards : <https://10.160.3.177:32100/explore> and the default datasource in the Grafana Explorer is Thanos , as shown in this following picture:

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	59 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



### Test Case - Serve infrastructure information from monitoring aggregator

#### Prerequisites:

- ▶ Monitoring servers: telemetry collected from ICOS nodes is stored in a time-series database with up to date data.
- ▶ Aggregator: service up and running with connectivity to the monitoring server.

#### Expected result:

- ▶ Aggregator serves infrastructure static and dynamic data with the infrastructure data model.

### Test Record - [PASSED]



A representation of how the results of the collection of the metrics are illustrated for the 3 nodes deployed (Nuvla, nod-edge and node-cloud) is provided in the following figure:

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	60 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final



### Test Case - Metrics from monitored resources and applications are collected

#### Prerequisites:

- ▶ Nuvla agent, edge-node and one cloud agent are deployed in ICOS Agent (a kubernetes cluster)
- ▶ ICOS Controller receive the metrics from the OTLP

#### Expected result:

- ▶ The metrics are collected, exposed, and can be accessed on an endpoint.

#### Test Record - [PASSED]

- ▶ The metrics exposed and running can be seen at: [10.160.3.177:32101/metrics](http://10.160.3.177:32101/metrics)

```
# HELP go_cgo_go_to_c_calls_calls_total Count of calls made from Go to C by the current process.
# TYPE go_cgo_go_to_c_calls_calls_total counter
go_cgo_go_to_c_calls_calls_total 0
# HELP go_cpu_classes_gc_mark_assist_cpu_seconds_total Estimated total CPU time goroutines spent performing GC tasks to assist the GC and prevent it from falling behind the application.
This metric is an overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_gc_mark_assist_cpu_seconds_total counter
go_cpu_classes_gc_mark_assist_cpu_seconds_total 0.354102891
# HELP go_cpu_classes_gc_mark_dedicated_cpu_seconds_total Estimated total CPU time spent performing GC tasks on processors (as defined by GOMAXPROCS) dedicated to those tasks. This
includes time spent with the world stopped due to the GC. This metric is an overestimate, and not directly comparable to system CPU time measurements. Compare only with other
/cpu/classes metrics.
# TYPE go_cpu_classes_gc_mark_dedicated_cpu_seconds_total counter
go_cpu_classes_gc_mark_dedicated_cpu_seconds_total 53.27020078
# HELP go_cpu_classes_gc_mark_idle_cpu_seconds_total Estimated total CPU time spent performing GC tasks on spare CPU resources that the Go scheduler could not otherwise find a use for.
This should be subtracted from the total GC CPU time to obtain a measure of compulsory GC CPU time. This metric is an overestimate, and not directly comparable to system CPU time
measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_gc_mark_idle_cpu_seconds_total counter
go_cpu_classes_gc_mark_idle_cpu_seconds_total 2.961480372
# HELP go_cpu_classes_gc_pause_cpu_seconds_total Estimated total CPU time spent with the application paused by the GC. Even if only one thread is running during the pause, this is
computed as GOMAXPROCS times the pause latency because nothing else can be executing. This is the exact sum of samples in /gc/pause:seconds if each sample is multiplied by GOMAXPROCS at
the time it is taken. This metric is an overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_gc_pause_cpu_seconds_total counter
go_cpu_classes_gc_pause_cpu_seconds_total 1.74812612
# HELP go_cpu_classes_gc_total_cpu_seconds_total Estimated total CPU time spent performing GC tasks. This metric is an overestimate, and not directly comparable to system CPU time
measurements. Compare only with other /cpu/classes metrics. Sum of all metrics in /cpu/classes/gc.
# TYPE go_cpu_classes_gc_total_cpu_seconds_total counter
go_cpu_classes_gc_total_cpu_seconds_total 58.333910163
# HELP go_cpu_classes_idle_cpu_seconds_total Estimated total available CPU time not spent executing any Go or Go runtime code. In other words, the part of /cpu/classes/total:cpu-seconds
that was unused. This metric is an overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_idle_cpu_seconds_total counter
go_cpu_classes_idle_cpu_seconds_total 267085.814071008
# HELP go_cpu_classes_scavenge_assist_cpu_seconds_total Estimated total CPU time spent returning unused memory to the underlying platform in response eagerly in response to memory
pressure. This metric is an overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_scavenge_assist_cpu_seconds_total counter
go_cpu_classes_scavenge_assist_cpu_seconds_total 7.47e-07
# HELP go_cpu_classes_scavenge_background_cpu_seconds_total Estimated total CPU time spent performing background tasks to return unused memory to the underlying platform. This metric is
an overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes metrics.
# TYPE go_cpu_classes_scavenge_background_cpu_seconds_total counter
go_cpu_classes_scavenge_background_cpu_seconds_total 0.452758401
# HELP go_cpu_classes_scavenge_total_cpu_seconds_total Estimated total CPU time spent performing tasks that return unused memory to the underlying platform. This metric is an
overestimate, and not directly comparable to system CPU time measurements. Compare only with other /cpu/classes/scavenge.
# TYPE go_cpu_classes_scavenge_total_cpu_seconds_total counter
```

- ▶ The default datasource Thanos is provided at the following link: <http://10.160.3.177:32105/stores>.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	61 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

Browser tabs: Accedi, Posta, Gabriele, WP3, Troub, Posta, Recenti, D3.1-l, Teleme, docker, Docker, signo, Th x, Explor, +

Address bar: Non sicuro | 10.160.3.177:32105/stores

Browser extensions: Posta - Maria Anto..., Use-case Validation..., ICOS, My Account - Disp..., [MOR-367] [CHUV]..., [MOR-367] [CHUV]..., opnio github - Cer..., OpenPolicyAgent

Page title: Thanos - Query | Graph | Stores | Status | Help

### Receive [show less](#)

Endpoint	Status	Announced LabelSets	Min Time (UTC)	Max Time (UTC)	Last Successful Health Check	Last Message
10.233.31.191:10901	UP	<a href="#">resolve="true"</a> <a href="#">replica="tele1-thenos-receive-0"</a> <a href="#">tenant_id="default-tenant"</a>	2023-10-12 09:03:04	—	3.388s ago	

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	62 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

## 5 Results and next steps

---

The ICOS project successfully concluded the initial development and integration of the ICOS Meta-Kernel Layer Module for IT-1 delivery. The IT-1 delivery was primarily used to validate the main architectural and design ideas the project delivered in D2.1 and D2.2 and was working on up this point.

### Implementation and validation results

- ▶ Set up the development and integration platform with CI/CD tools.
- ▶ Set up testbed for validating the results of the Meta-Kernel development and integration activities.
- ▶ Provided initial implementation and integration of the discovery service – Lighthouse.
- ▶ ICOS Controller can be registered with ICOS discovery service Lighthouse.
- ▶ Designed the Application Description and Application Deployment Models used by user and by the Meta-Kernel internally in the process of the application deployment.
- ▶ Provided initial implementation of the ICOS Shell for the users to authenticate with ICOS and interact with the ICOS Controller.
- ▶ User can use ICOS Shell to submit the application deployment request to ICOS Shell Backend.
- ▶ Deployment works for OCM via ICOS Job Manager using a simple Matchmaking mechanism.
- ▶ Telemetry is collected by Prometheus Node Exporters and pushed to Thanos for storage and topology reconciliation.

Below we list immediate next steps that the project will be working on for ICOS Alfa Release

- ▶ Improving topology collection and representation by including IoT peripherals.
- ▶ Improving the request/response schemas of documents used in the cross-component communications of the Meta-Kernel Layer.
- ▶ Extend integration of Cloud and Edge Orchestrator to Nuvla.io Platform by completing the development of the Nuvla.io specific Deployment Manager connector.
- ▶ Implement first elements of the security and authentication/authorization across all the components of the Meta-Kernel with the aim of extending the authorization policies down to the elements of the Cloud-Edge-IoT continuum (e.g., down to the user accounts on the Edge COE clusters) that will already follow in ICOS Beta Release.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)			<b>Page:</b>	63 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0
				<b>Status:</b>	Final

## 6 Annexes

### 6.1 RESTful API resource schemas and examples

This section provides the schemas and examples (if applicable) of the RESTful API interfaces of the design components defined in section 2.2.

#### 6.1.1 user.shell-backend-i

##### GET /deployment/{deploymentId} resource data model

Attribute Name	Type	Description
id	Long	Unique identifier of the deployment
name	String	[optional] Name of the deployment
status	String	[optional] Status of the deployment

##### /resource/{resourceId} resource data model

Attribute Name	Type	Description
id	Long	Unique identifier of the resource
name	String	[optional] Name of the resource
type	String	Type of resource
parentId	Long	ID of the parent resource
status	String	Status of the resource

#### 6.1.2 runtime.job-manager-i

##### GET /deployment/job resource data model

Attribute Name	Type	Description
ID	UUID	Unique identifier of the Job
UUID	UUID	Mapped Unique Identifier of the Deployment executed by OCM
Type	DataType	Name of the deployment
State	DataType	Status of the deployment
Manifest	String	Application Description
Targets	Target[]	Target Nodes
Locker	Bool	True if the Job is locked, False if the job is unlocked
UpdatedAt	DateTime	The timestamp when the Job was last time updated (write operation was done)

- ▶ **GET** /deployment/jobs/{deploymentId} returns a Job following the job resource schema.
- ▶ **GET** /deployment/jobs/ returns a list of Jobs following the job resource schema.

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	64 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final



- ▶ **POST** /jobmanager/jobs/create returns the created Job following the job resource schema.
- ▶ **PUT** /deployment/jobs/{deploymentId} retrieves the updated job following the job resource schema.

### Target model

Attribute Name	Type	Description
ID	uint32	Unique identifier of the Target Cluster/Node
cluster_name	String	The name of the cluster returned by matchmaking service
node_name	String	The name of the node returned by matchmaking service

The GET /deployment/jobs/{deploymentId} will retrieve the Job model as follows:

```
{
  "ID": "51abf632-6f1c-4c23-88aa-692a084d7ace",
  "uuid": "00000000-0000-0000-0000-000000000000",
  "type": 5,
  "state": 1,
  "manifest": "apiVersion: apps/v1\r\nkind: Deployment\r\nmetadata:\r\n  name: hello-cloud\r\n  labels:\r\n    app: hello-cloud\r\nspec:\r\n  replicas: 1\r\n  selector:\r\n    matchLabels:\r\n      app: hello-cloud\r\n  template:\r\n    metadata:\r\n      name: hello-cloud\r\n      labels:\r\n        app: hello-cloud\r\n    spec:\r\n      containers:\r\n        - name: hello-cloud-one\r\n          image: busybox\r\n          command: [ \"sleep\" ]\r\n          args: [ \"100\" ]\r\n          resources:\r\n            requests:\r\n              cpu: 1\r\n            memory: 3Gi\r\n        - name: hello-cloud-two\r\n          image: busybox\r\n          command: [ \"sleep\" ]\r\n          args: [ \"100\" ]\r\n          resources:\r\n            requests:\r\n              cpu: 2\r\n            memory: 3Gi",
  "targets": [
    {
      "id": 2,
      "JobID": "51abf632-6f1c-4c23-88aa-692a084d7ace",
      "cluster_name": "k3s-worker1",
      "node_name": "k3s-worker1"
    }
  ],
  "locker": false,
  "updatedAt": "2023-10-17T07:59:52.494Z"
}
```

The POST /jobmanager/jobs/create responds with the created Job, plus the information retrieved from the matchmaking service:

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	65 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

```
{
  "ID": "51abf632-6f1c-4c23-88aa-692a084d7ace",
  "uuid": "00000000-0000-0000-0000-000000000000",
  "type": 5,
  "state": 1,
  "manifest": "apiVersion: apps/v1\nkind: Deployment\nmetadata:\n  name: hello-cloud\n  labels:\n    app: hello-cloud\nspec:\n  replicas: 1\n  selector:\n    matchLabels:\n      app: hello-cloud\n  template:\n    metadata:\n      name: hello-cloud\n    labels:\n      app: hello-cloud\n    spec:\n      containers:\n        - name: hello-cloud-one\n          image: busybox\n          command: [ \"sleep\" ]\n          args: [ \"100\" ]\n          resources:\n            requests:\n              cpu: 1\n              memory: 3Gi\n        - name: hello-cloud-two\n          image: busybox\n          command: [ \"sleep\" ]\n          args: [ \"100\" ]\n          resources:\n            requests:\n              cpu: 2\n              memory: 3Gi",
  "targets": [
    {
      "id": 2,
      "JobID": "51abf632-6f1c-4c23-88aa-692a084d7ace",
      "cluster_name": "k3s-worker1",
      "node_name": "k3s-worker1"
    }
  ],
  "locker": false,
  "updatedAt": "2023-10-17T07:59:52.494Z"
}
```

The PUT /deployment/jobs/{deploymentId} will respond with the updated Job as follows:

```
{
  "ID": "51abf632-6f1c-4c23-88aa-692a084d7ace",
  "uuid": "00000000-0000-0000-0000-000000000000",
  "type": 5,
  "state": 2,
  "manifest": "apiVersion: apps/v1\nkind: Deployment\nmetadata:\n  name: hello-cloud\n  labels:\n    app: hello-cloud\nspec:\n  replicas: 1\n  selector:\n    matchLabels:\n      app: hello-cloud\n  template:\n    metadata:\n      name: hello-cloud\n    labels:\n      app: hello-cloud\n    spec:\n      containers:\n        - name: hello-cloud-one\n          image: busybox\n          command: [ \"sleep\" ]\n          args: [ \"100\" ]\n          resources:\n            requests:\n              cpu: 1\n              memory: 3Gi\n        - name: hello-cloud-two\n          image: busybox\n          command: [ \"sleep\" ]\n          args: [ \"100\" ]\n          resources:\n            requests:\n              cpu: 2\n              memory: 3Gi",
  "targets": [
    {
      "id": 2,
      "JobID": "51abf632-6f1c-4c23-88aa-692a084d7ace",

```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	66 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

```

    "cluster_name": "k3s-worker1",
    "node_name": "k3s-worker1"
  }
],
"locker": false,
"updatedAt": "2023-10-17T08:50:23.433Z"
}

```

**JobType DataType Enum:** CreateDeployment, GetDeployment and DeleteDeployment.

**State DataType Enum:** Created, Progressing, Available and Degraded.

### 6.1.3 runtime.matchmaker-i

#### POST /matchmake request data model

Attribute Name	Type	Description
apiVersion	String	The API version of the Kubernetes object (Deployment).
Kind	String	The type or kind of the Kubernetes object (Deployment).
metadata	Object	Contains metadata information about the deployment.
metadata.name	String	The name of the deployment ("hello-cloud").
metadata.labels	Object	Labels associated with the deployment.
metadata.labels.app	String	Label specifying the application as "hello-cloud".
Spec	Object	Describes the desired state for the deployment.
spec.replicas	Integer	The desired number of replicas for the deployment.
spec.selector	Object	Defines how the deployment selects which pods to manage.
spec.selector.matchLabels	Object	Specifies the labels used for pod selection.
spec.template	Object	Describes the pod that will be created by this deployment.
spec.template.metadata	Object	Metadata for the pod template.
spec.template.metadata.name	String	The name of the pod template ("hello-cloud").
spec.template.metadata.labels	Object	Labels associated with the pod template.
spec.template.metadata.labels.app	String	Label specifying the application as "hello-cloud".
spec.template.spec	Object	Specification for the pod template.
spec.template.spec.containers	Array of Objects	Defines the containers within the pod.
spec.template.spec.containers.name	String	The name of the container ("hello-cloud-one" or "hello-cloud-two").

Attribute Name	Type	Description
spec.template.spec.containers.image	String	The Docker image used for the container ("busybox").
spec.template.spec.containers.command	Array of Strings	The command to be executed by the container (["sleep"]).
spec.template.spec.containers.args	Array of Strings	The arguments passed to the container (["100"]).
spec.template.spec.containers.resources	Object	Resource requirements for the container.
spec.template.spec.containers.resources.requests.cpu	Integer	CPU request for the container (1 or 2).
spec.template.spec.containers.resources.requests.memory	String	Memory request for the container ("3Gi").

Following is an example of the expected input passed as JSON:

```

{
  "apiVersion": "apps/v1",
  "kind": "Deployment",
  "metadata": {
    "name": "hello-cloud",
    "labels": {
      "app": "hello-cloud"
    }
  },
  "spec": {
    "replicas": 1,
    "selector": {
      "matchLabels": {
        "app": "hello-cloud"
      }
    },
    "template": {
      "metadata": {
        "name": "hello-cloud",
        "labels": {
          "app": "hello-cloud"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "hello-cloud-one",

```



```

"targets": [
  {
    "cluster_name": "10.42.0.85:8080",
    "node_name": "k3s-worker1"
  }
]

```

#### 6.1.4 runtime.execution-manager-i

##### ExternalAdaptorResource

Name	Type	Mandatory	Description
name	String	true	Name of the node
description	<b>ResourceDescription</b>	true	Features of the node
adaptor	String	true	Offloader's plugin used to communicate with the node
projectConf	<b>Property[]</b>	false	Description of how to setup the resource
resourcesConf	<b>Property[]</b>	false	Description of how to access the resource

##### Property

Name	Type	Mandatory	Description
Name	String	true	Name of the property
Value	String	true	Value for that property

##### ResourceDescription

Name	Type	Mandatory	Description
processors	<b>ProcessorDescription[]</b>	false	Embedded processors
memory_size	float	false	Available memory (GB)
memory_type	String	false	Memory technology
storage_size	float	false	Available disk (GB)
storage_type	String	false	Storage technology
storage_bandwidth	float	false	Disk bandwidth (Mbps)

##### ProcessorDescription

Name	Type	Mandatory	Description
name	String	true	Processor Name
units	int	true	number of cores
architecture	String	false	Instruction set

## Application

Name	Type	Mandatory	Description
lang	String [JAVA PYTHON]	true	Implementing language of the function
className	String	true	Class or Module containing the function
methodName	String	true	Name of the method to execute
ceiClass	String	false	If Java, selection of methods to become nested tasks upon invocation
Parameters	<b>Parameter</b> []	true	Description of the parameters to pass in to the method invocation
Target	<b>Parameter</b>	false	Description of the object onto which invoke the method
Results	<b>Parameter</b> []	false	Description of the values generated by the invocation

## Parameter

Name	Type	Mandatory	Description
paramId	int	true	Position of the parameter in the invocation
paramName	String	false	Name of the parameter in the function
type	<b>Data Type</b>	true	Format of the data being passed in
direction	String [IN, OUT, INOUT]	true	Directionality of the parameter
prefix	String	false	Name of the method to execute
stdIOStream	String [STDIN, STDOUT, STDERR, UNSPECIFIED]	false	IO stream that should be linked to the value of this parameter
contentType	Type of data contained	false	If collective type, what kind of data is enclosed in the collection
weight	float	false	importance of the parameter at scheduling stage
value	Object	false	Value of the parameter

## Data Type Enum:

BOOLEAN\_T, CHAR\_T, BYTE\_T, SHORT\_T, INT\_T, LONG\_T, FLOAT\_T, DOUBLE\_T, STRING\_T, STRING\_64\_T, FILE\_T, OBJECT\_T, PSCO\_T, EXTERNAL\_PSCO\_T, BINDING\_OBJECT\_T, WCHAR\_T, WSTRING\_T, LONGLONG\_T, VOID\_T, ANY\_T, ARRAY\_CHAR\_T, ARRAY\_BYTE\_T, ARRAY\_SHORT\_T, ARRAY\_INT\_T, ARRAY\_LONG\_T, ARRAY\_FLOAT\_T, ARRAY\_DOUBLE\_T, COLLECTION\_T, DICT\_COLLECTION\_T, STREAM\_T, EXTERNAL\_STREAM\_T, ENUM\_T, NULL\_T, *DIRECTORY\_T*;

## 6.1.5 continuum.aggregator-i

### GET / response data model

#### Controllers

Attribute Name	Type	Description
Controller	<b>Controller[]</b>	ICOS Controller instance

#### Controller

Attribute Name	Type	Description
Type	string	ICOS Controller type
Name	string	ICOS Controller name
Location	<b>Location</b>	Location of the Controller
ServiceLevelAgreement	<b>ServiceLevelAgreement</b>	SLA information applied on the Controller
API	<b>API</b>	API information of the Controller
Any	any	Complementary Controller data

#### Location

Attribute Name	Type	Description
Name	string	Location name
Continent	string	Location continent
Country	string	Location country
City	string	Location city
Latitude	float64	Location latitude
Longitude	float64	Location longitude

#### ServiceLevelAgreement

Attribute Name	Type	Description
Name	string	SLA Name

#### API

Attribute Name	Type	Description
CommunicationProtocol	string	Communication protocol used by the API
ProtocolVersion	string	Protocol version used by the API
DataFormat	string	Data format used by the API
Authentication	string	Authentication info used by the API
Authorization	string	Authorization info used by the API



## Agents

Attribute Name	Type	Description
Cluster	<b>Cluster</b> []	Cluster instance managed by ICOS

## Cluster

Attribute Name	Type	Description
Type	string	Cluster type
Name	string	Name of the Cluster
Location	<b>Location</b>	Location of the Cluster
ServiceLevelAgreement	<b>ServiceLevelAgreement</b>	SLA information applied on the Cluster
API	<b>API</b>	API information of the Cluster
Node	<b>Node</b> []	List of nodes of the Cluster
Network	Network	Network data
Security	string	Security data
Deployment	Deployment[]	Deployments running on the Cluster
IoT	IoT[]	IoT devices on the Cluster
Any	any	Complementary Cluster data

## Node

Attribute Name	Type	Description
Type	string	Type of Node
Name	string	Name of the Node
Location	Location	Location of the Node
StaticMetrics	<b>StaticMetrics</b>	Static metrics
DynamicMetrics	<b>DynamicMetrics</b>	Dynamic metrics

## StaticMetrics

Attribute Name	Type	Description
CPUCores	float64	Number of CPU cores
CPUMaxFrecuency	string	Maximum CPU frequency
GPUCores	float64	Number of GPU cores
GPUMaxFrecuency	string	Maximum GPU frequency
GPURAMMemory	string	GPU memory capacity
RAMMemory	string	Node memory capacity
Storage	<b>StaticStorage</b> []	Storage static information

### StaticStorage

Attribute Name	Type	Description
Name	string	Name of Storage
Type	string	Type of Storage
Capacity	float64	Capacity of Storage

### DynamicMetrics

Attribute Name	Type	Description
CPUFrequency	string	Current CPU frequency usage
CPUTemperature	float64	Current CPU temperature
CPUEnergyConsumption	float64	Current CPU energy consumption
GPUFrequency	string	Current GPU frequency usage
GPUTemperature	float64	Current GPU temperature
GPUEnergyConsumption	float64	Current GPU energy consumption
RAMUsage	string	Current RAM usage
DynamicStorage	<b>DynamicStorage</b> []	Current storage usage
NetworkUsage	string	Current network usage

### DynamicStorage

Attribute Name	Type	Description
Name	string	Name of Storage
Free	float64	Current capacity of Storage

### Network

Attribute Name	Type	Description
ConnectivityType	string	Type of connection
Latency	float64	Network latency
IPAddress	string	Network IP address
IPGateway	string	Network gateway
Interface	<b>Interfaces</b> []	Network interfaces

### Interfaces

Attribute Name	Type	Description
Name	string	Name of the Interface
Type	string	Type of Interface
Speed	float64	Interface speed in bytes
IP	string	IP address of the Interface
SubnetMask	string	Subnet Mask of the Interface

Attribute Name	Type	Description
IngressUsage	string	Ingress usage of the Interface
EgressUsage	string	Egress usage of the interface

### Deployment

Attribute Name	Type	Description
Status	string	Status of the Deployment
NumberOfContainers	float64	Number of containers of the Deployment
NumberOfApps	float64	Number of apps of the Deployment
Container	<b>Container[]</b>	Containers of the Deployment

### Container

Attribute Name	Type	Description
Name	string	Name of the Container
Node	string	Name of the Node running the Container
Port	Port[]	Ports used by the Container
MemoryUsage	string	Memory usage of the Container
CPUUsage	string	CPU usage of the Container
IP	string	IP address of the Container

### Port

Attribute Name	Type	Description
Port	string	Port name/number

### IoT

Attribute Name	Type	Description
Type	string	Type of IoT device
Status	string	Status of the IoT device
Properties	<b>Properties[]</b>	Properties of the IoT device
IoTNetwork	<b>IoTNetwork</b>	Network data of the IoT device
API	API	API information of the IoT device

## Properties

Attribute Name	Type	Description
Resolution	string	Screen resolution
EngUnits	string	Engineering units of the measured
MinScale	string	Minimum magnitude of the measured value
MaxScale	string	Maximum magnitude of the measured value
TouchScreen	string	Touch Screen capabilities of the device

## IoTNetwork

Attribute Name	Type	Description
ProtocolVersion	string	Protocol version of the IoT device
Latency	float64	Latency of the IoT device
Bandwidth	float64	Bandwidth of the IoT device

Example of the response to GET / request.

```
{
  "10.42.0.63:8080": {
    "name": "10.42.0.63:8080",
    "location": {},
    "serviceLevelAgreement": {},
    "API": {},
    "node": {
      "ocm-worker1.bull1.ari-imet.eu": {
        "name": "ocm-worker1.bull1.ari-imet.eu",
        "staticMetrics": {
          "cpuCores": 4
        },
        "dynamicMetrics": {}
      }
    },
    "deployment": {
      "aggregator": {
        "container": {
          "aggregator-7b55c4fbd7-x8x42": {
            "name": "aggregator-7b55c4fbd7-x8x42"
          }
        }
      },
      "alertmanager": {
        "container": {
          "alertmanager-prom-kube-prometheus-alertmanager-0": {
            "name": "alertmanager-prom-kube-prometheus-alertmanager-0"
          }
        }
      }
    }
  }
}
```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	76 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

```

    },
    "blackbox-exporter": {
      "container": {
        "prom-kube-prometheus-blackbox-exporter-5868775964-ttq58": {
          "name": "prom-kube-prometheus-blackbox-exporter-5868775964-ttq58"
        }
      }
    },
    "config-reloader": {
      "container": {
        "alertmanager-prom-kube-prometheus-alertmanager-0": {
          "name": "alertmanager-prom-kube-prometheus-alertmanager-0"
        },
        "prometheus-prom-kube-prometheus-prometheus-0": {
          "name": "prometheus-prom-kube-prometheus-prometheus-0"
        }
      }
    },
    "coredns": {
      "container": {
        "coredns-6b8bb547dc-hnmvq": {
          "name": "coredns-6b8bb547dc-hnmvq"
        }
      }
    },
    "debug-pod-ubuntu": {
      "container": {
        "debug-pod-ubuntu": {
          "name": "debug-pod-ubuntu"
        }
      }
    },
    "helm": {
      "container": {
        "helm-install-traefik-6tqvs": {
          "name": "helm-install-traefik-6tqvs"
        },
        "helm-install-traefik-crd-qx7mk": {
          "name": "helm-install-traefik-crd-qx7mk"
        }
      }
    },
    "klusterlet": {
      "container": {
        "klusterlet-6d77765698-2k6v4": {
          "name": "klusterlet-6d77765698-2k6v4"
        },
        "klusterlet-6d77765698-9t6r8": {
          "name": "klusterlet-6d77765698-9t6r8"
        },
        "klusterlet-6d77765698-v9dl5": {
          "name": "klusterlet-6d77765698-v9dl5"
        }
      }
    },
  },

```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	77 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
		<b>Version:</b>	1.0
		<b>Status:</b>	Final

```

"klusterlet-manifestwork-agent": {
  "container": {
    "klusterlet-work-agent-7788fbdd48-cd5q7": {
      "name": "klusterlet-work-agent-7788fbdd48-cd5q7"
    }
  }
},
"kube-state-metrics": {
  "container": {
    "prom-kube-state-metrics-7546dd4946-q5fjq": {
      "name": "prom-kube-state-metrics-7546dd4946-q5fjq"
    }
  }
},
"lb-tcp-443": {
  "container": {
    "svclb-traefik-7fe5ecef-f298h": {
      "name": "svclb-traefik-7fe5ecef-f298h"
    }
  }
},
"lb-tcp-80": {
  "container": {
    "svclb-traefik-7fe5ecef-f298h": {
      "name": "svclb-traefik-7fe5ecef-f298h"
    }
  }
},
"local-path-provisioner": {
  "container": {
    "local-path-provisioner-76d776f6f9-8mm62": {
      "name": "local-path-provisioner-76d776f6f9-8mm62"
    }
  }
},
"metrics-server": {
  "container": {
    "metrics-server-7b67f64457-fs8t9": {
      "name": "metrics-server-7b67f64457-fs8t9"
    }
  }
},
"nginx": {
  "container": {
    "nginx-deployment-85996f8dbd-8662g": {
      "name": "nginx-deployment-85996f8dbd-8662g"
    },
    "nginx-deployment-85996f8dbd-cn8m7": {
      "name": "nginx-deployment-85996f8dbd-cn8m7"
    }
  }
},
"node-exporter": {
  "container": {
    "prom-node-exporter-rfklq": {
      "name": "prom-node-exporter-rfklq"
    }
  }
}

```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	78 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
		<b>Version:</b>	1.0
		<b>Status:</b>	Final

```

    },
    "prometheus": {
      "container": {
        "prometheus-prom-kube-prometheus-prometheus-0": {
          "name": "prometheus-prom-kube-prometheus-prometheus-0"
        }
      }
    },
    "prometheus-operator": {
      "container": {
        "prom-kube-prometheus-operator-7649ccb784-6tgld": {
          "name": "prom-kube-prometheus-operator-7649ccb784-6tgld"
        }
      }
    },
    "registration-controller": {
      "container": {
        "klusterlet-registration-agent-59594fccfd-xmdlf": {
          "name": "klusterlet-registration-agent-59594fccfd-xmdlf"
        }
      }
    },
    "traefik": {
      "container": {
        "traefik-57c84cf78d-vdl78": {
          "name": "traefik-57c84cf78d-vdl78"
        }
      }
    }
  }
}

```

<b>Document name:</b>	D3.1 Meta-Kernel Layer Module Integrated (IT-1)	<b>Page:</b>	79 of 79
<b>Reference:</b>	D3.1	<b>Dissemination:</b>	PU
		<b>Version:</b>	1.0
		<b>Status:</b>	Final