D.5.2 Report on Integration the CECCM

Document Summary Information

Project Identifier	HORIZON-CL4-2022-DATA-01. Project 101093129		
Project name	Agile and Cognitive Cloud-edge Continuum management		
Acronym	AC^3		
Start Date	January 1, 2023	**End Date**	December 31, 2025
Project URL	www.ac3-project.eu		
Deliverable	D5.2 Report on Integration of the CECCM		
Work Package	WP5		
Contractual due date	M24: 31st December 2024	**Actual submission date**	M29: May 2025
Type	R (Report, Document)	**Dissemination Level**	PU (Public)
Lead Beneficiary	RHT		
Responsible Author	Ray Carroll (RHT)		
Contributors	Ray Carroll (RHT), Ben Capper (RHT), Ryan Jenkins (RHT), D. Amaxilatis (SPA), N. Tsironis (SPA), T. Sarantakos (SPA), D. Klonidis, N. Psaromanolakis (UBI), Mario Chamorro (UCM), Eduardo Ojeda (IQU), Abdelhak Kadouma (FIN), Mohamed Mekki (EUR), Mohamed Mokhtari (EUR), Meliani Abd Elghani(EUR)		

Funded by the European Union

AC^3 project has received funding from European Union's Horizon Europe research and innovation programme under Grand Agreement No 101093129.

Peer reviewer(s)	Amadou Ba (IBM), John Beredimas (CSG)

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
V0.1	12/02/2025	5%	Initial Deliverable Structure	Ray Carroll (RHT)
V0.2	12/03/2025	60%	First Round of Contributions	All
V0.3	20/03/2025	70%	Initial Review	Ray Carroll (RHT) Ben Capper (RHT)
V0.4	01/04/2025	80%	UC1, UC2, UC3 Updates	Ray Carroll (RHT) Dimitris Amaxilatis (SPA) Abdelhak Kadouma (FIN) Dimitris Klonidis (UBI) Nikos Psaromanolakis (UBI)
V0.5	11/04/2025	85%	ToC, Figures, Executive Summary, Annex	Ray Carroll (RHT)
V0.6	22/04/2025	90%	Peer Review	Amadou Ba (IBM) John Beredimas (CSG)
V0.7	01/05/2025	93%	Document Cleanup	Ben Capper (RHT)
V0.8	19/05/2025	95%	Coordinator Review	Adlen Ksentini (EUR)
V0.9	24/05/2025	97%	Coordinator Review Comments	Ben Capper (RHT) Dimitris Amaxilatis (SPA) Eduardo Ojeda (IQU)
V1.0	24/07/2025	100%	Final Version	

Disclaimer

© AC³ 2023

The content of this document reflects only the author's view. Neither the European Commission nor the HaDEA are responsible for any use that may be made of the information it contains.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the AC³ consortium make no warranty of any kind with regard to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Neither the AC³ consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the AC³ Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© AC³ Consortium. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Table of Contents

© AC³ 2023

Table of Figures

© AC³ 2023

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
AC³	Agile and Cognitive Cloud edge Continuum management
ACM	Advanced Cluster Management
AI	Artificial Intelligence
API	Application Programming Interface
AppD	Application Descriptor
CECC	Cloud Edge Computing Continuum
CECCM	Cloud Edge Computing Continuum Manager
CI/CD	Continuous Integration/Continuous Delivery
CRUD	Create, Read, Update, Delete
GUI	Graphical User Interface
HPA	Horizontal Pod Autoscaler
IFS	Integral Field Spectroscopy
IoT	Internet of Things
JWST	James Webb Space Telescope
KPI	Key Performance Indicator
K8s	Kubernetes
LAN	Local Area Network
LCM	Life-Cycle Management
LISO	Lightweight Edge Slice Orchestration
LMS	Local Management System
ML	Machine Learning
NSD	Network Service Descriptors
NBI	Northbound Interface
OSR	Ontology and Semantic aware Reasoner
OWL	Web Ontology Language

© AC³ 2023

PaaS	Platform as a Service
RAM	Random Access Memory
RDF	Resource Description Framework
RLOs	Resource Level Objects
RL	Reinforcement Learning
ROL	Resource Orchestration Layer
RBAC	Role-Based Access Control
S3	Simple Storage Service
SD-WAN	Software-Defined Wide Area Network
SLA	Service Level Agreement
TMF	TM Forum
UAV	Unmanned Aerial Vehicle
UC	Use Case
VLT	Very Large Telescope
VoD	Video on Demand
XAI	eXplainable AI
XGBoost	Extreme Gradient Boosting

© AC³ 2023

1 Executive Summary

In the present document, **D5.2 Report on Integration of the CECCM,** we provide a comprehensive description of the work carried out thus far on the integration of the Cloud Edge Computing Continuum Manager (CECCM) components within the AC^3 project. As per [1] D5.1 Initial Integration and proofs of concept plan, a Use Case (UC) oriented approach to integration is taken, in which the CECCM components of the architecture are integrated across the three UCs (Internet of Things and Data, Smart Monitoring System using Unmanned Aerial Vehicles, and Deciphering the universe: processing hundreds of TBs of astronomy data), subject to their relevance to the specific UC. The core aim is to clearly demonstrate how the integration of the CECCM components supports and augments the UCs, and the work completed towards that goal. As such, this report gives a comprehensive level of detail on the partial technical implementation of the component integration.

In Sections 3, 4, and 5, on a UC basis, we briefly revisit the core goals and objectives and then describe the architecture of both the UC application and testbed. We then give a detailed description and workflow of how the CECCM components are integrated into the UC, before finally going into a detailed description of the integration of each component. A brief summary of the 3 UCs is given below:

1. UC1, ***IoT and Data***, strives to optimize resource allocation in office buildings. It involves deploying sensors to monitor environmental factors and human presence, with the goal of maximizing occupant health and comfort by adjusting lighting and heating systems in real-time.
2. UC2, ***Smart Monitoring System using UAVs***, revolves around the development and implementation of a Smart Monitoring System utilizing unmanned aerial vehicles (UAV), AI/Machine Learning (ML) technologies, and edge computing to enhance video surveillance and environmental monitoring. Its primary goal is to optimize urban security, traffic management, and environmental tracking through the CECCM.
3. UC3, ***Deciphering the universe: processing hundreds of TBs of astronomy data***, analyses large 3D data cubes of astronomy data, which contain a detailed image and spectral information about galaxies, to extract key insights such as stellar kinematics and population characteristics. Its main challenge is that handling these vast datasets requires significant computing power, memory, and efficient data management, as well as scalable and distributed processing capabilities.

All the UCs will demonstrate the core components of the AC3 architecture, including the GUI, OSR, LCM, and LMS. While UC2 highlights the integration of far-edge nodes, such as drones, both UC1 and UC2 showcase data management capabilities; specifically focusing on the integration of hot and cold data sources, respectively. Additionally, each UC will incorporate AI-based algorithms, as defined in WP3 and WP4, to enhance LCM functionalities and manage application life cycles. This includes features such as AI-driven application migration and AI/XAI-based workload scalability management.

Further, within each UC section, and specifically in their concluding integration summaries, the deliverable provides a clear status of integrated components, identifying those that have been finalized and those that require further work, building upon the initial integration plans outlined in D5.1.

The main conclusion of this report, detailed in Section 6, is that, through the extensive collaborative work carried out by both the UC and component owners, a clear path to realising the AC^3 vision is now in place. Based on the strong foundational work described within this report, the consortium can now progress towards delivering tangible benefits of the CECCM through demonstrations, experimentation, and results within the final phase of the project.

© AC^3 2023

2 Introduction

2.1 Overview – Purpose and objectives

To date, the project has been focused on designing and refining the core AC³ CECCM architecture, as well as cultivating innovation within each of the CECCM component areas. In D5.1, we then outlined the **Initial integration and proofs of concept plan,** and in this deliverable, we follow this up with a detailed report on the progress towards these integration goals. As such, the core objective of this deliverable is to present the current state of the component integration. This document gives a detailed overview of how the components are integrated into each UC, in terms of interface points and interaction flows. It also gives a detailed technical description of the implementation of each of the CECCM components, which gives a clear sense of the level of work that has been carried out to date.

As discussed in D5.1, the approach taken has been to align the integration activities with the UCs and to use these as drivers for selection, implementation, and evaluation of the relevant parts of the AC³ architecture. This gives the integration a clear focus and direction, while also delivering component integrations that present the most value towards the core goals of that UC.

2.2 Link with other project activities

This deliverable is a continuation from [1] D5.1 "Initial integration and proofs of concept plan", and provides insights into the partial developments, integrations, and proof of concept activities related to Task T5.1. The work done in the UC adheres to the requirements defined in D2.4, "Business Analysis of CECC and Use Case Requirements" [2]. Regarding the data management mechanism, D3.3, "Initial Report on Data Management for Applications in CECC" [3], provides direct feedback for the design of the overall system and its interface with AC³ service deployment and management mechanisms. D4.1, "Initial Report on Mechanisms that Enable Green-Oriented Zero Touch Management of CECC Resources" [4], informs the work on resource discovery and monitoring, AI/ML models for resource management, green-oriented LCM decisions for resource management, and networking programmability of CECC. D3.1, "Initial report on the Application LCM in the CEC" [5], navigates through the various components related to the user plane of the CECCM, namely the User Interfaces, Application Profiles, Ontology Modeling Tools and Application Descriptor Models. Moreover, D2.3 "Report on technological tools for CECC" [6] provides a comprehensive overview of the technological tools, laying the groundwork for their integration within the AC³ framework. The careful selection and analysis of these tools in D2.3 was a crucial initial step toward ensuring a smoother integration process in subsequent work packages. Finally, D2.1 "CECC framework and CECCM" [1] provides the architecture framework for all UC.

2.3 Mapping AC³ Outputs

The purpose of this section is to map AC³ Grant Agreement commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to AC³ GA Deliverable & Tasks Descriptions

AC³ GA Component Title	AC³ GA Component Outline	Respective Document Chapter(s)	Justification

DELIVERABLE			
D5.2 Report on Integration of the CECCM: *"Report on the implementation of the CECCM and integration of components."*			
TASKS			
Task T5.1: AC³ components integration	*"Each partner involved will develop their individual components and/or functions and show their project results based on the tests done in their labs, where all essential functions can be tested."*	Sections 3.3, 3.4, 4.3, 4.4, 5.3, 5.4	For each UC, the "Component Integration Design" section describes how and which AC³ components are leveraged by the CECCM software to support the respective UC, while the "Component Integration Status" sections detail the progress of integrating these components to enable that support.
Task T5.2: Testbed integration	*"This task will concern the integration of the software and hardware needed to run the three UCs."*	Sections 3.2, 4.2, 5.2	The "Use Case Architecture" section of each UC describes the core infrastructure set up by project partners to deploy applications and AC³ components for the respective UC.
Task T5.3: Field trials execution	*In this task, the evaluation of the solutions proposed in WP3/4 will be performed through simulation and experimentation of the three PoC.*	Sections 3.5, 4.5, 5.5	For each UC, the "Results" section demonstrates how the experimentations and simulations can validate that the AC³ components and CECCM software meet the KPIs and metrics required by the respective UC applications.

2.4 Deliverable Overview and Report Structure

Sections 3, 4, and 5: UC1, UC2, and UC3

Sections 3, 4, and 5 describe the implementation and integration efforts for UC1 (IoT and Data), UC2 (Smart Monitoring System using UAVs), and UC3 (Processing TBs of Astronomy Data), respectively. While the UCs are separated for clarity, all three sections share a common internal structure to ensure consistency, as follows:

- **Use Case Description and Objectives**: Provides an overview of the UC goals and core functionalities (e.g., optimizing building operations for UC1, enhancing urban surveillance for UC2, processing astronomical data for UC3), setting the stage for the integration work.

- **Use Case Architecture:** This subsection details the architecture of the UC, including the application design and testbed setup, highlighting how they support the UC objectives within the AC³ framework.
- **Component Integration Design:** Explains the design of CECCM component integration for the UC, detailing how components (e.g., EDC Connectors, LiSO, Maestro, OSR) interact with the UC to enhance functionality and performance.
- **Component Integration Status:** Offers a detailed technical update on the implementation status of each integrated CECCM component, including progress on data management, orchestration, and monitoring systems.
- **Results:** Presents any initial results from the integration efforts, such as experimental outcomes or system performance metrics, where available, with further details deferred to future deliverables.

Section 6: Conclusions

This section summarizes the deliverable key findings, emphasizing the collaborative work between UC and component owners. It reaffirms the progress toward the AC³ vision and outlines the next steps for achieving tangible benefits through demonstrations and experimentation.

Section 7: References

This section lists all external sources, including project deliverables and relevant scientific publications, that provide foundational context, data, or methodologies cited within the deliverable. This ensures transparency and enables readers to access original source material for further information.

Section 8: Annex

This section provides a complete view of the application descriptors that can be generated by the OSR for each of the UCs, detailing the microservices, dependencies, and group affinities for each.

© AC³ 2023

3 UC1

3.1 Use Case Description

UC1 introduces an innovative IoT-based, smart sensing and monitoring framework designed to leverage the transformative potential of edge AI technologies within a Cloud Edge Computing Continuum (CECC) infrastructure. It aims to enhance the monitoring and management of infrastructures ranging from individual smart homes to expansive smart grids on a national scale, regardless of the underlying technologies for data collection and data communication. In this context, UC1 is engineered to integrate physical and digital realms more seamlessly than ever, thereby managing and processing a significantly larger volume of IoT data to enable timely decisions and responsive actions based on sensed conditions. It also focuses on data fusion, integrating outputs from diverse sensors to create detailed profiles and detect patterns that help in proactively managing events and minimizing their impact on infrastructure operations. This advanced functionality not only supports basic applications like air quality monitoring but also intends to enable immediate, localized decision-making through a blend of IoT innovation and edge computing intelligence.

3.1.1 Objectives

UC1 is set to showcase the remarkable capabilities of the CECCM, with a keen focus on several ambitious objectives that highlight its potential:

- **Intuitive application definition:** Leveraging an intuitive GUI and Ontology and Semantic aware Reasoner (OSR), the application developer can efficiently define and deploy microservice applications within the CECCM framework. This system will simplify the user experience, making it easier to harness cutting-edge technology, including cloud and edge domains and AI/ML capabilities. Moreover, the system includes several essential components that are integral to data management: Catalogues, which provide descriptions for available data sources; Data Provider Connector, offering access to the data made available; Data Consumer Connector, which initiates the streaming of data from the source to the application microservices; Data Mappers, responsible for transforming incoming data as needed; Data Manipulator, which handles the core application logic; and a Message Broker, responsible for transferring data between the various application components, ensuring smooth communication and synchronization throughout the system.
- **Automatic Deployment and Zero-Touch Management**: UC1 will demonstrate the robust Life-Cycle Management (LCM) capabilities of the CECCM developed in AC^3, enabling the automatic deployment, monitoring, and maintenance of microservice applications. Using AI and ML-driven zero-touch management algorithms, the system can autonomously optimise and sustain application performance.
- **Microservice Deployment and Migration**: The CECCM's resilience will be further highlighted by its ability to deploy and manage microservices across cloud-edge environments. Should the application in an edge deployment become unavailable or face resource limitations, the system can automatically migrate services to alternative cloud infrastructures, ensuring uninterrupted service delivery and continuity.
- **Data Analysis and Decision-Making for Smart Building Installations**: By leveraging AI and ML techniques, the system will enable real-time analysis for the environmental and human presence detection data. This real-time data analysis will contribute to enhanced building monitoring for actionable insights and facilitate AI-based decision-making.

© AC^3 2023

3.1.2 UC1 Stakeholders

3.1.2.1 End Users / Smart Building Operators

The primary beneficiaries of the deployed UC1 application are the building operators (in our case, the operators of the IQU facility). They rely on the system's real-time insights to monitor and manage indoor environmental conditions such as CO_2 levels, temperature, and humidity in conjunction with human presence information. By leveraging these insights, they ensure occupant safety, comfort, and well-being by adapting the building's air circulation, heating, or cooling. Through real-time alerts, analytics dashboards, and historical data trends, they can make informed decisions or even trigger proactive maintenance and improved environmental management practices to ensure the best and most cost-effective measures are taken.

Although building occupants do not interact directly with the system, they also benefit from AI-assisted automation and decision-making that optimize their surroundings and allow them to access limited information about their workplace.

3.1.2.2 Application/Software Developers

The Application and DevOps teams are responsible for designing, implementing, and deploying the microservices that comprise the UC1 application. These include modules for data ingestion from IoT sensors, data preprocessing, ML-based inference (e.g., room occupancy prediction), and real-time alerting mechanisms. They also train the ML model needed based on real-world data available in the AC³ Data Catalogue.

The DevOps team also handles these services' packaging, deployment, and lifecycle automation using the CECCM. By integrating the AC³ components, they benefit from intuitive and efficient automated deployment, monitoring, maintenance, and migration of microservices between the edge and cloud domains to maintain performance. This significantly reduces operational overhead and simplifies the management of AI-powered microservices in an edge environment.

3.1.2.3 Infrastructure Provider / CECCM Integrator

They are responsible for deploying and maintaining the edge and cloud computing environments. This includes:

- Provisioning and configuring physical hardware (IoT sensors, data collectors nodes, and servers)
- Deploying the Kubernetes (K8s)-based Local Management System (LMS) in the computing domains
- Integrating the CECCM components (including LiSO for LCM, LMS, GUI, OSR, algorithms developed as enablers, and so on)

The infrastructure provider ensures that the CECCM is available and operational to support the DevOps and Application teams during application deployment and runtime. They also handle the secure communication and synchronization between data sources, edge, and cloud domains.

3.2 Use Case Architecture

3.2.1 Use Case Application

The UC1 application centres on developing an IoT-enabled system for monitoring and optimizing building operations in real time. By deploying a network of sensors throughout a building environment, the system continuously gathers data on energy consumption, indoor environmental conditions such as temperature, humidity, and CO_2 levels, as well as occupancy patterns and equipment performance. This rich stream of data

provides the foundation for intelligent analysis that supports energy-efficient operation, improved indoor comfort, and predictive maintenance strategies.

The application demonstrates how buildings can evolve into smart, responsive infrastructures by leveraging IoT technologies to make informed decisions automatically or with minimal human intervention. To enable this, the system is built on a robust data infrastructure that supports real-time processing, semantic interoperability across heterogeneous devices, secure and privacy-compliant data sharing, and scalable analytics. Through this approach, we illustrate how IoT-based monitoring can drive sustainability, reduce operational costs, and contribute to the transformation of traditional buildings into intelligent, adaptive spaces that actively participate in wider smart city and digital twin ecosystems.

The core of the application is split into two parts, data input and data processing. The first part is responsible for the continuous acquisition and integration of data generated by the building's IoT infrastructure. A diverse network of sensors is deployed across the building to monitor environmental conditions (such as temperature, humidity, and air quality, etc.). The system ingests high-frequency data streams from these sources through the AC^3 data management, handling issues of heterogeneity and interoperability across different sensor types and communication protocols. It ensures reliable data collection through real-time or near-real-time pipelines built using the AC^3 data management addons. This lays the groundwork for intelligent processing by maintaining an up-to-date and structured view of the building's operational context.

Building upon the data collected from the IoT infrastructure, the second part of the application focuses on advanced data analysis using machine learning models. This includes both unsupervised and supervised learning techniques tailored for detecting anomalies in the building's behaviour and forecasting future conditions. Anomaly detection models are trained to identify deviations from typical patterns, which may indicate sensor faults, equipment failures, or unusual usage behaviours. In parallel, forecasting models predict key variables such as energy consumption, temperature evolution, or occupancy trends, enabling proactive management strategies. These insights can be used to optimize energy use, schedule maintenance, and enhance user comfort, ultimately supporting data-driven decision-making in building operations.

© AC^3 2023

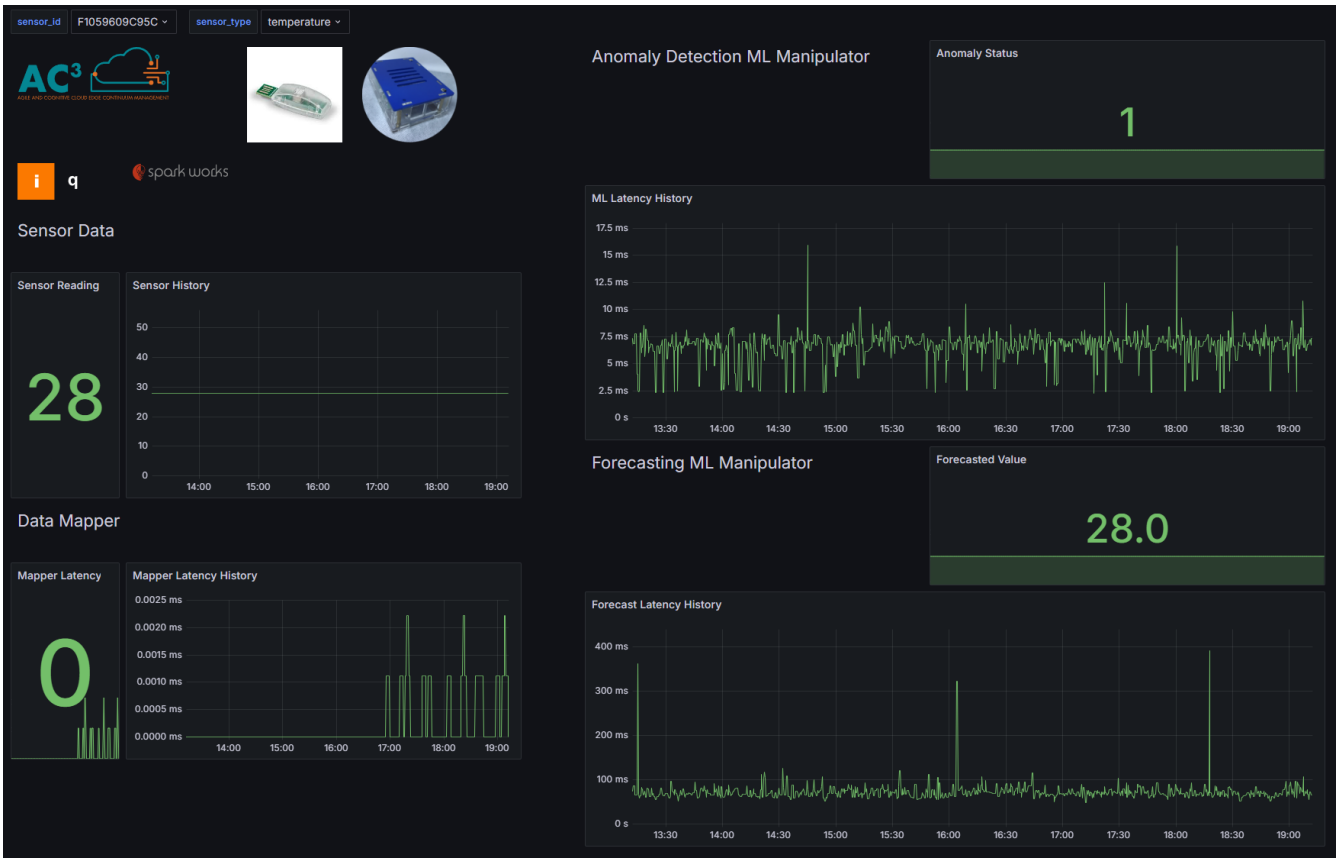

Figure 1 - Dashboard showing sensor readings, processing speed, and machine learning statistics in real time.

Figure 1 shows a dashboard view where the users can review the conditions reported by a specific sensing device. This data includes metrics for both the actual values coming from the sensors and metrics regarding the processing speed during the first part of the application, as well as the statistics during the application of the machine learning models on the received data. These metrics are primarily collected and used inside the UC1 application to evaluate its performance and its operation.

3.2.2 UC Testbed – Hardware and Software

The broad UC1 testbed integrates IoT, edge computing, and cloud technologies, which are detailed in the following section, as illustrated in Figure 2:

Edge Domain: An edge server hosting a K8s cluster forms the edge domain, optimizing resource management for the UC1 AC^3 application. This setup allows for data processing close to the source, minimizing communication costs and latency. However, high IoT traffic can lead to increased processing times beyond the specified SLAs.

Cloud Domain: UC1 utilizes a K8s cluster hosted by ION for cloud services, facilitating service migration in AC^3. This cloud environment can manage significant data volumes, handling peaks by transferring processing from the edge to the cloud, which may increase latency but ensures compliance with SLAs. Performance monitoring tools like Prometheus and Grafana are hosted here for data visualization and analysis.

Data Source Domain: This domain is focused on real-time IoT data collection and transmission, utilizing:

- **IoT Sensors:** The IQU building is equipped with Sensirion SCD41 CO2 sensors and Shelly Motion 2 devices for monitoring.
- **Raspberry Pi4 & Pi5 Devices:** These compact, cost-effective single-board computers run software for managing sensor data and provide 5G connectivity via SIM8200EA-M2 5G HATs or Wi-Fi as an alternative.

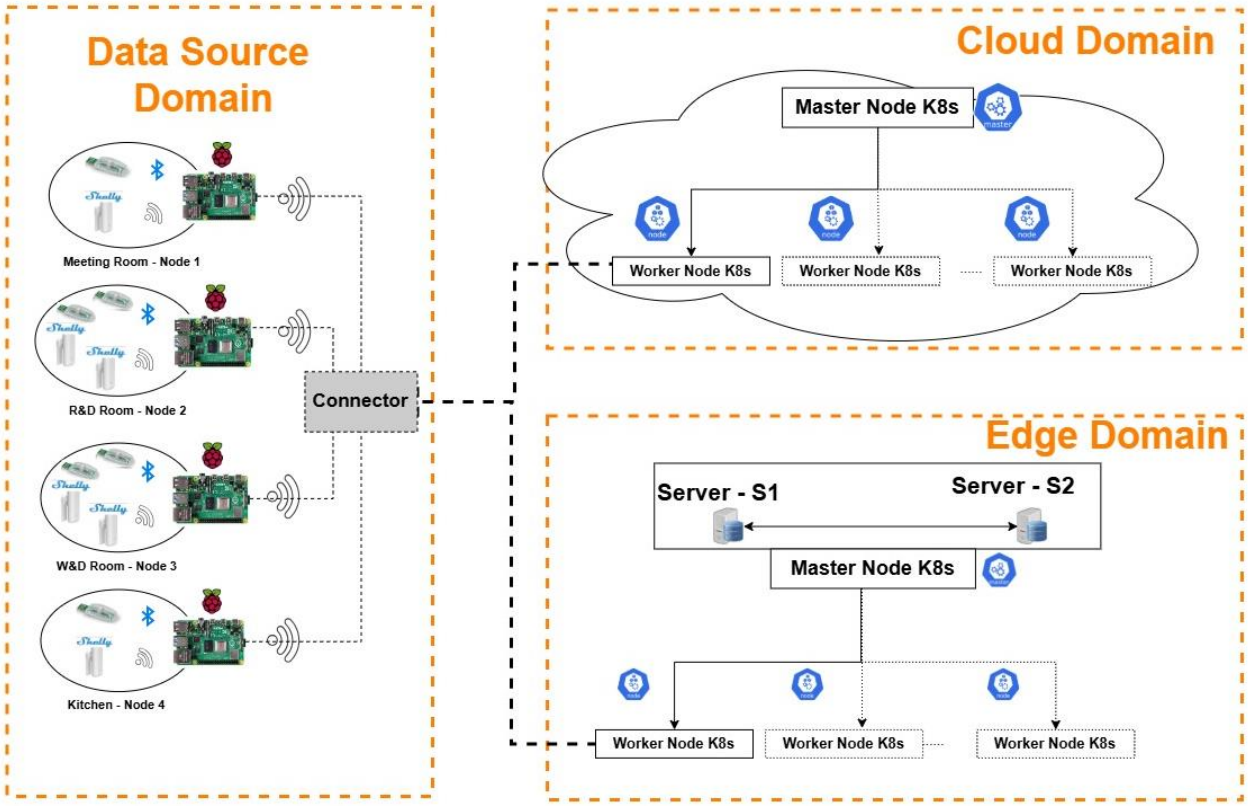

Figure 2. Depiction of the UC1 Data Source, Edge and Cloud Domains and the flow of data between them.

3.2.2.1 Data Source Testbed

The UC1 data source is based on an IoT deployment in the IQU offices that represents the IoT infrastructure to be used for UC1. Figure 3 depicts the setup within an office environment, covering multiple rooms including a Meeting Room, R&D Room, W&D Room, and a Kitchen. The testbed integrates various IoT devices and sensors across these spaces to enable real-time monitoring, automation, and data-driven insights. Throughout the office, multiple environmental sensors are deployed, positioned strategically in key locations such as workspaces, restrooms, and common areas. These sensors measure parameters such as temperature, humidity, air quality (as CO2 concentration), and occupancy, providing a comprehensive dataset for the analysis of the conditions inside the office building, the goal of the AC³ UC1 application. Alongside the sensors, the testbed incorporates Raspberry Pi devices, which serve as data collection nodes. These devices are present in each room to facilitate data aggregation. The primary objectives of this IoT testbed include environmental monitoring, where real-time tracking of air quality and climate conditions provides insights for occupancy, energy optimization, and indoor comfort. The inclusion of edge computing capabilities to be presented in the next subsection enables edge

processing of IoT data before transmission to a central system, enhancing responsiveness and reducing network congestion.

This IoT testbed represents a scalable and modular framework suitable for smart office applications, energy efficiency research, and workplace analytics. By leveraging a combination of environmental sensors, edge computing, and smart automation technologies, it provides a robust infrastructure for studying and optimizing office environments.

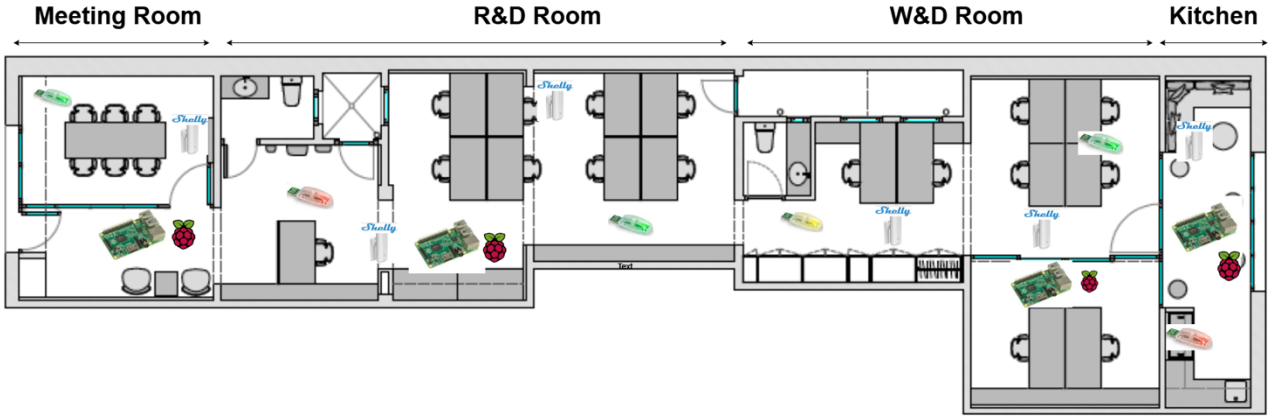

Figure 3. The IQU Offices IoT Data Source Domain that represents the IoT infrastructure of UC1.

3.2.2.2 Edge Compute Testbed

The Edge Compute Testbed for UC1 is hosted at IQU's facilities and is designed to execute low-latency data processing using edge AI capabilities. It supports the deployment of UC1's microservice-based applications and AC³ components through a K8s-based LMS.

K8s Cluster Configuration

The edge domain is structured as a **K8s cluster** with one master and three worker nodes as defined in Table 2. All nodes operate over a secure WireGuard overlay network for inter-node communication and connectivity with the data source domain.

Table 2: UC1 Edge K8s Cluster Details

Node Role	Hostname	CPU	Cores	Memory	Status
Master Node	ac3-master-vm	Intel Xeon Gold 5218	16	192 GB DDR4	Running
Worker Node #1	ac3-node-1-vm	Intel Xeon Gold 5218	16	192 GB DDR4	Running
Worker Node #2	ac3-node-2-vm	Intel Xeon Gold 5218	16	192 GB DDR4	Running

Worker Node #3	ac3-node-3-vm	Intel Xeon Gold 5218	16	192 GB DDR4	Running

All nodes are deployed on a high-performance bare-metal server (**S1**) that was defined in D4.2 [2], section 5.1.4, and benefit from centralized storage via a ZFS-backed 2TB NVMe SSD for high-throughput data access. A second server (**S2**) specified in D4.2 [2] is also available in case more nodes need to be added to the cluster. It features an Intel i9-10900L CPU (10 cores / 20 threads) and a 2 TB NVMe SSD.

LMS and Cluster Software Stack

The edge domain LMS is powered by K8s, and the software stack includes:

- **K8s Version:** v1.30.10
- **Container Runtime:** containerd
- **Container Network Interface:** Calico
- **Overlay Network:** WireGuard (for secure inter-node communication)
- **Internal Monitoring Stack:** Prometheus + Grafana (deployed on both edge and cloud domains)
- **Security:** TLS encryption with Role-Based Access Control (RBAC) enforcement for K8s API access.

This setup ensures that all edge domain services in UC1, such as microservices for sensor ingestion, ML model inference, and messaging, can be deployed and monitored.

Network Setup:

Figure 4 depicts the external-access architecture for the edge K8s cluster, where port forwarding securely exposes the K8s API server. Port 6443 is forwarded to the master node at 10.0.0.8:6443, enabling external access via IQU's domain (dev.iquadrat.com:6443). RBAC enforces fine-grained permissions, and TLS certificates ensure all communication is encrypted and authenticated.

```
Kubernetes control plane is running at https://dev.iquadrat.com:6443
CoreDNS is running at https://dev.iquadrat.com:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Figure 4. Exposed Endpoints of the K8s Cluster API

Furthermore, this configuration strengthens security and facilitates seamless integration and secure data exchange between the data source domain and the edge domain. This enhancement is made possible by the incorporation of the Raspberry Pi devices into the WireGuard overlay, which enables effective communication between these domains.

© AC³ 2023

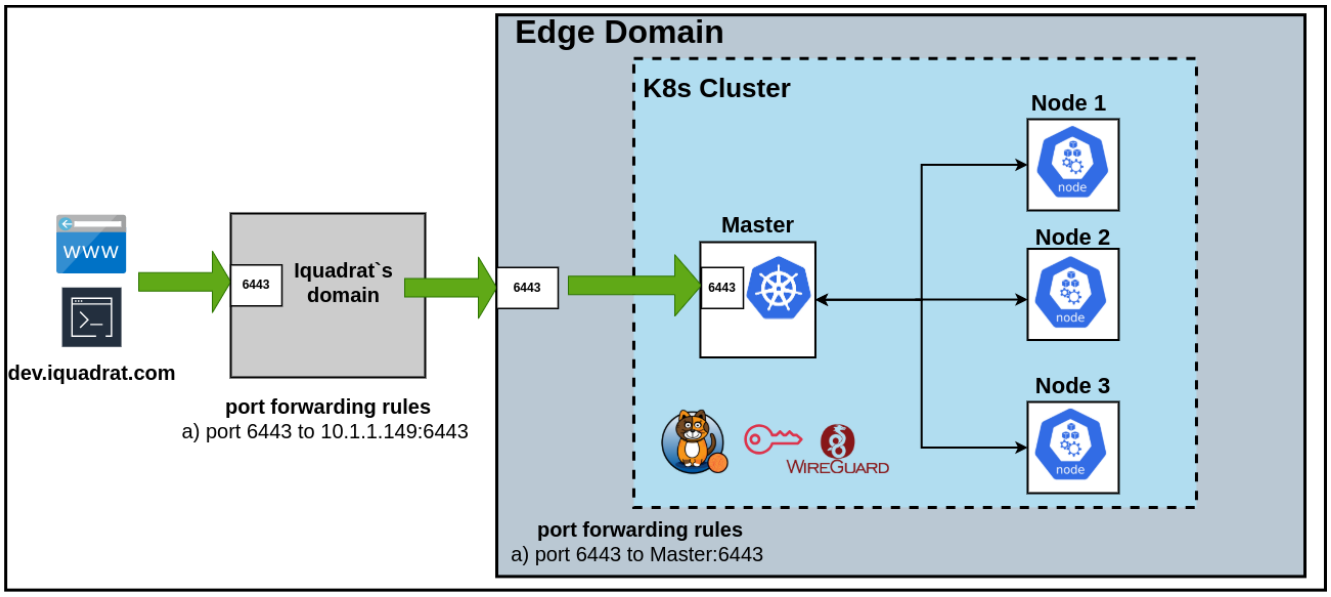

Figure 5. Architecture for Exposing the K8s API Server Through Domain-Level Port Forwarding

3.2.2.3 Cloud Compute Testbed

The Cloud Compute Testbed of UC1 operates using the same software tools as the Edge Compute Testbed. The computing infrastructure used is provided by IONOS as a virtual K8s deployment on its cloud resources to provide more computing capabilities than the edge location.

3.3 Component Integration Design

Multiple components developed in AC³ are integrated with the UC1 application to facilitate the demonstration of the AC³ features used, and their integration is depicted in Figure 6. In more detail:

1. The data source of UC1 is registered in the AC³ Catalogue, and the EDC Connectors are used to retrieve the data from it (presented in green colour).
2. The application's resource usage metrics are exposed to the monitoring framework via the Prometheus collectors deployed (light yellow colour blocks) in both edge and cloud locations. These metrics will be used to train the ML models for the application profiling and migration to implement the intelligent Lifecycle Management.
1. The UC1 application specific components (RabbitMQ, Mapper, ML and UI) are presented in orange colour and can be deployed either at the edge or cloud locations available. In the role of LCM, we have used LiSO from EUR. LiSO, which is described in D2.3 [6], is responsible for the deployment of the application as well as enforcing application runtime decisions based on the AI model recommendations based on the Application Descriptor (AppD) that is generated by the OSR.
3. The computation LMS used in our UC is K8s.

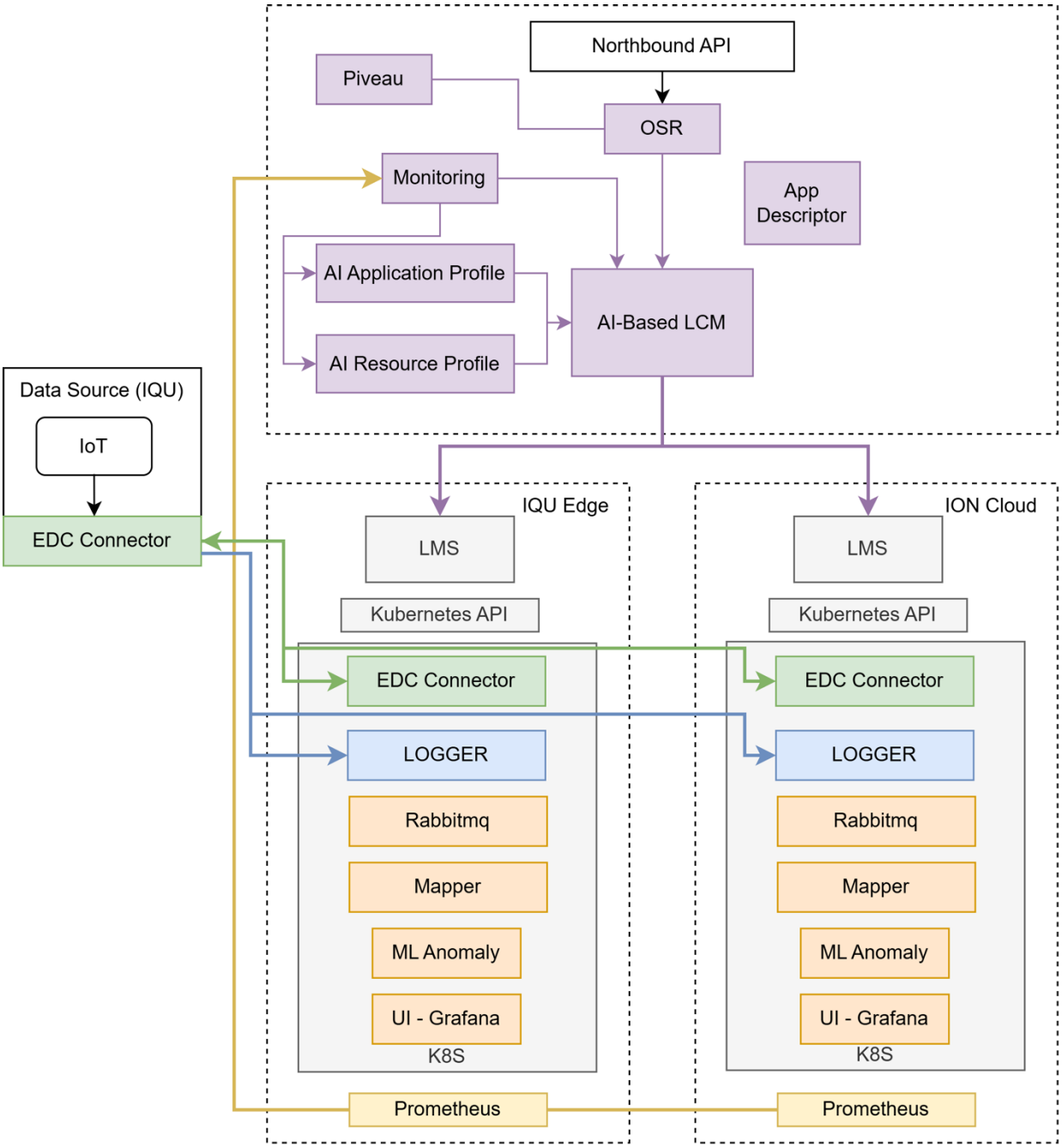

Figure 6. UC1 / AC³ component integration

The UC1 application definition and deployment are outlined in Figure 7. The process begins with the definition of the application services and the data needed through the AC³ GUI. The user can either retrieve and use services that are already defined in the Service Catalogue or define new services that are to be stored in the Service Catalogue to be re-used later on from other applications. The GUI also sends the application details to the App Gateway that will forward a request to the OSR in order to build a combined AppD that includes both the services of the application, as well as the datasets to be used and the data management application addons that are

needed for the application to work. The merged AppD will then be sent to the LCM and to the UC1 Testbed's LMS for the final deployment.

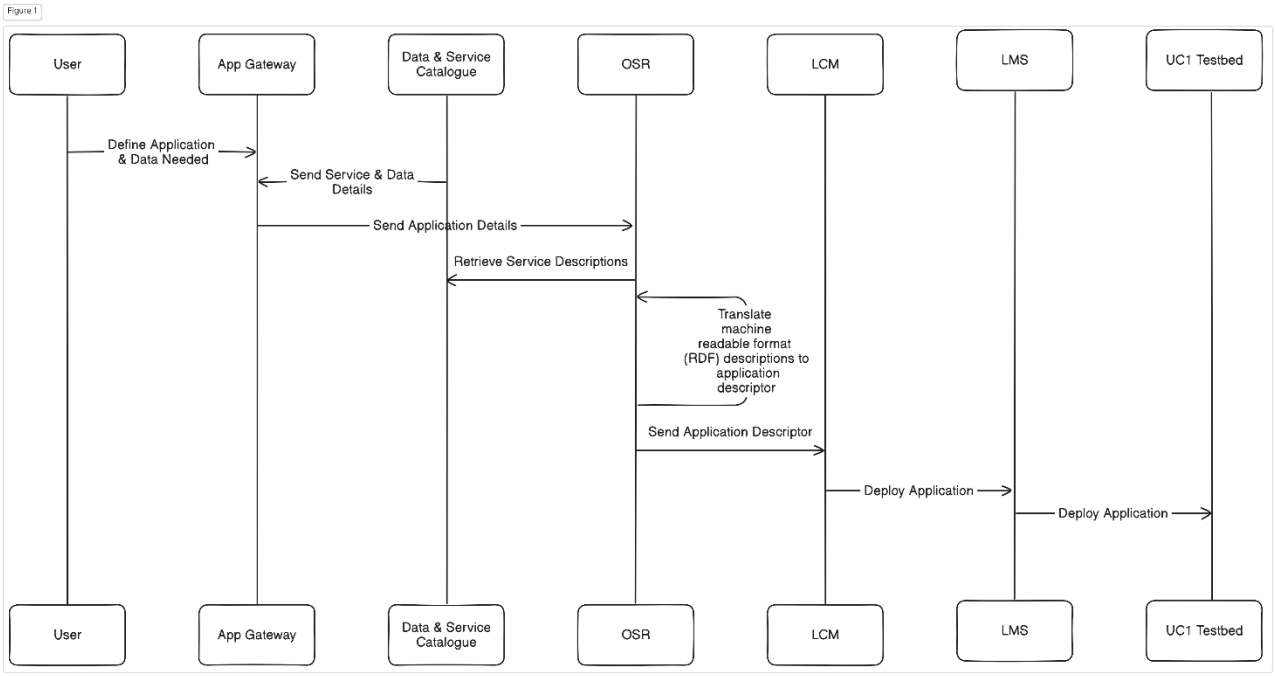

Figure 7. UC1 application definition and deployment

The UC1 data processing workflow diagram is illustrated in Figure 8. IoT data updates are generated from the IoT Sensors and sent to the provider EDC Connector. The data transfer is initiated by the consumer EDC Connector with a process that will be further defined in Section 3.4.3. After this process is complete, the provider EDC Connector starts to forward Sensor Data updates to the Logger deployed as part of the UC1 application. The Logger then forwards them to the RabbitMQ broker, also deployed by UC1. From there, each data point is "mapped" from a dedicated service to an internal data triple that is then distributed to the data analysis components of our application (ML-1 for anomaly detection and ML-2 for value forecasting). The results from each ML model are again published to the RabbitMQ server as an analysis result. These results, as well as the original data, are also forwarded to and displayed in the UC1 UI (based on Grafana) to be made available to our application's users.

© AC³ 2023

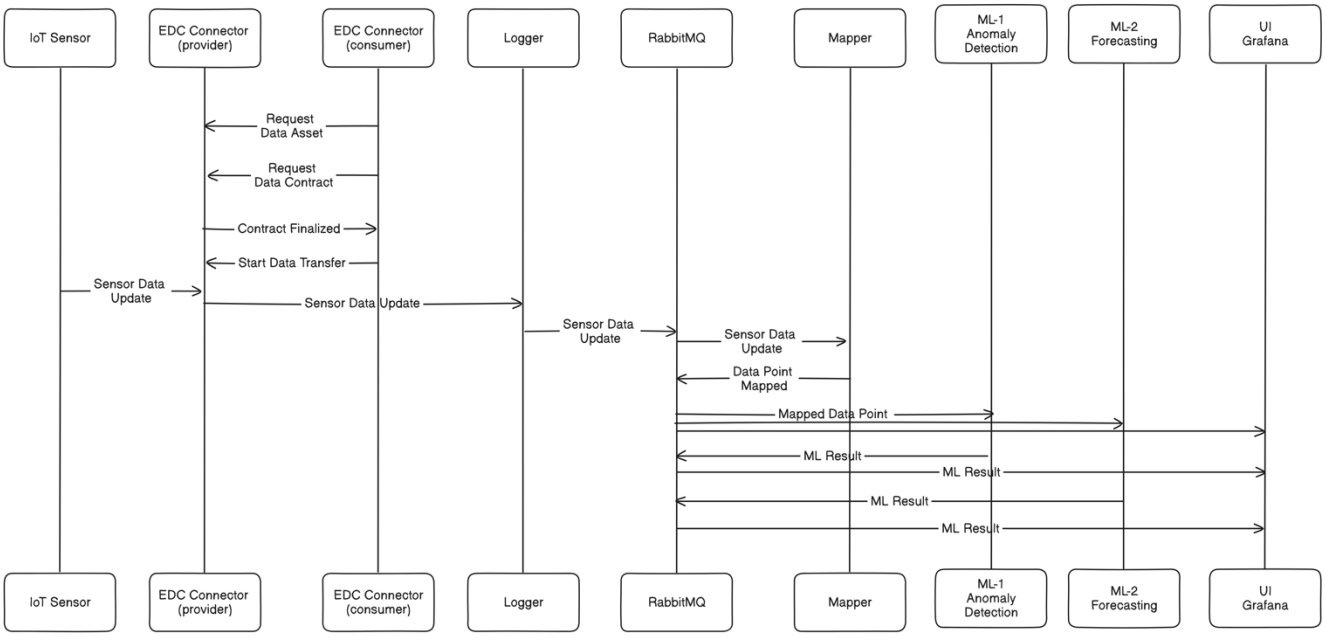

Figure 8. UC1 Data Processing Pipeline

3.4 Component Integration Status

3.4.1 Data Management and Connectors

The dataset of UC1 described in Figure 9Figure 9: UC1 Dataset consists of the data streams originating from the IQU testbed. It is registered in the catalogue following the Gaia-X principles for data sharing and interoperability, providing all the needed information about its data and its characteristics. It is also providing the asset ID “uc1-stream”. This id is used through a dedicated data endpoint at http://ds.uc1.ac3.sparkworks.net:18182/protocol, using the "dspaceconnector" protocol for communication to get access to the stream of data.

To access and utilize the dataset, specific services and connectors are required, as listed in the AC³ catalogue (they will be described in the next subsection):

- A required connector is linked, which serves as an intermediary for secure data exchange.
- Additional service offerings are also linked, ensuring that the dataset can be processed, analysed, and integrated in our application.

The dataset is governed by the BSD-3-Clause license, which allows for redistribution and modification with minimal restrictions, making it suitable for both academic and commercial applications. Two contact points are also available for additional information and requests from the two consortium members that curate the data source.

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix ns0: <https://ac3-project.eu/#> .
@prefix ns1: <https://schema.org/> .
@prefix ns2: <https://w3id.org/gaia-x/development#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
```

```
<https://catalogue.ac3-project.eu/set/resource/dataset/2befff01-885e-4e5d-a216-e2354baae959>
  a <https://w3id.org/gaia-x/development#Dataset> ;
  dc:title "UC1 Dataset 1"@en ;
  geo:lat 41.3976242 ;
  geo:long 2.1560832 ;
  dcat:assetId "uc1-stream" ;
  dcat:contactPoint <https://catalogue.ac3-project.eu/set/resource/dataset/2befff01-885e-
4e5d-a216-e2354baae959/Contact1>, <https://catalogue.ac3-
project.eu/set/resource/dataset/2befff01-885e-4e5d-a216-e2354baae959/Contact2> ;
  dcat:endpointDescription "dspaceconnector" ;
  dcat:endpointURL "http://ds.uc1.ac3.sparkworks.net:18182/protocol" ;
  ns0:requiredConnector <https://catalogue.ac3-project.eu/set/resource/service/10333e33-
0875-4869-9f15-7fc7fccd1d48> ;
  ns0:requiredServiceOffering <https://catalogue.ac3-
project.eu/set/resource/service/d8a6b514-3b80-48ec-90cd-d4a229736fb3>,
<https://catalogue.ac3-project.eu/set/resource/service/5b6372e6-11a0-47dc-8b1c-
761ff81cba25>, <https://catalogue.ac3-project.eu/set/resource/service/115f7472-194d-446c-
9518-efb9f55d3f73> ;
  ns1:description "These data are used in the UC1 of the AC3 project. UC1 focuses on
processing and analysing streams of data originating from environmental sensors installed
inside office or residential buildings." ;
  ns1:name "IQU UC1 IoT Data 1" ;
  ns2:license "BSD-3-Clause" .

<https://catalogue.ac3-project.eu/set/resource/dataset/2befff01-885e-4e5d-a216-
e2354baae959/Contact1>
  a vcard:Kind ;
  vcard:fn "Iquadrat Informatica S.L." ;
  vcard:hasEmail <mailto:j.ojeda@iquadrat.com> ;
  vcard:hasName "Jhofre Ojeda" .

<https://catalogue.ac3-project.eu/set/resource/dataset/2befff01-885e-4e5d-a216-
e2354baae959/Contact2>
  a vcard:Kind ;
  vcard:fn "Spark Works Ltd." ;
  vcard:hasEmail <mailto:tsaradakos@sparkworks.net> ;
  vcard:hasName "Themistoklis Sarantakos" .
```

Figure 9: UC1 Dataset

3.4.2 Service Catalogue

In the AC³ Service Catalogue, we have defined a number of services that are designed to retrieve and process the data generated by the data source of UC1. These are the:

- UC1 Connector, based on the EDC Connector,
- UC1 Logger for incoming data to receive new IoT measurements,
- UC1 Data Mapper to prepare IoT measurements for processing by the IoT application.

In the rest of this Section, we will present the descriptions of these services and justify their properties.

Starting with the **UC1 Connector**, presented in Figure 10, we have the definition of a service based on the "sparkworks/ac3-connector-http-http-consumer:latest" docker image published in the official Docker Hub. This application needs a set of environmental variables to be configured and executed, as well as needs to expose a

set of TCP ports to facilitate communication with the data provider Connector. Regarding the environmental variables, it needs to get access to the following data:

- Ports to be used for the loaded services. These ports are provided as numerical values.
- PROVIDER_DOMAIN: Domain of the data source provider where the connector should send its requests to. This domain is provided as a value that the OSR will replace when building the application's descriptor.
- ASSET_NAME: The name of the asset to be requested from the data source connector. This value is provided as a value that the OSR will replace when building the application's descriptor.
- CONSUMER_DOMAIN: The domain where this application will be launched and become accessible on the public internet. This domain is provided as a value that the OSR will replace when building the application's descriptor.

Additionally, in the service's description we include a set of resource limits that are to be used when launching the service to make sure that the application will fit in the resources available in the deployment environment. The resource limits include CPU, Memory and GPU requirements and are based on the Gaia-X ontology's *ContainerResourceLimits*. Similarly, an SLA entity is added to define SLAs for the service that AC³ should apply. These values include *serviceAvailability*, *maxResponseTime* and *dataThroughput*.

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix ns0: <https://ac3-project.eu/#> .
@prefix ns1: <https://schema.org/> .
@prefix ns2: <https://w3id.org/gaia-x/development#> .

<https://catalogue.ac3-project.eu/set/resource/service/10333e33-0875-4869-9f15-7fc7fccd1d48>
  a <https://w3id.org/gaia-x/development#ServiceOffering> ;
  dc:title "Streaming IoT Connector" ;
  ns0:EnvironmentVariable "WEB_HTTP_MANAGEMENT_PORT=28181", "WEB_HTTP_PORT=28180",
"WEB_HTTP_PROTOCOL_PORT=28182", "WEB_BASE_URL=http://192.168.1.215",
"WEB_HTTP_CONTROL_PORT=28183", "ASSET_NAME=@dcat:assetId",
"PROVIDER_DOMAIN=@dcat:endpointURL", "CONSUMER_DOMAIN=@self-ip" ;
  ns0:ExposedPort "28180:28180", "28181:28181", "28182:28182", "28183:28183" ;
  ns0:image "sparkworks/ac3-connector-http-http-consumer:latest" ;
  ns1:description "This is an edc connector for streaming iot data" ;
  ns1:name "streaming-connector" .

<https://catalogue.ac3-project.eu/set/resource/service/10333e33-0875-4869-9f15-
7fc7fccd1d48/resourceLimits>
  a <https://w3id.org/gaia-x/development#ContainerResourceLimits> ;
  ns2:cpuRequirements 0.5 ;
  ns2:memoryRequirements 512 ;
  ns2:gpuRequirements 0 .

<https://catalogue.ac3-project.eu/set/resource/service/10333e33-0875-4869-9f15-
7fc7fccd1d48/sla>
  a ns0:microservicesSLA ;
  ns0:serviceAvailability 99.9 ;
  ns0:maxResponseTime "Medium" ;
  ns0:dataThroughput "High" .
```

Figure 10. UC1 Connector

The UC1 Logger (Figure 11) is based on the "sparkworks/ac3-amqp-http-request-logger:latest" Docker image, which is published in Docker Hub. This application requires minimal configuration, primarily setting up network ports for communication. Additionally, in the service's description, we include a set of resource limits that are to be used when launching the service to make sure that the application will fit in the resources available in the deployment environment. The resource limits include CPU, Memory, and GPU requirements and are based on the Gaia-X ontology's *ContainerResourceLimits*. Similarly, an SLA entity is added to define SLAs for the service that AC³ should apply. These values include *serviceAvailability*, *maxResponseTime* and *dataThroughput*.

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix ns0: <https://ac3-project.eu/#> .
@prefix ns1: <https://schema.org/> .
@prefix ns2: <https://w3id.org/gaia-x/development#> .

<https://catalogue.ac3-project.eu/set/resource/service/d8a6b514-3b80-48ec-90cd-d4a229736fb3>
  a <https://w3id.org/gaia-x/development#ServiceOffering> ;
  dc:title "Streaming IoT Logger" ;
  ns0:EnvironmentVariable "HTTP_SERVER_PORT=4000" ;
  ns0:ExposedPort "4000:4000" ;
  ns0:image "sparkworks/ac3-amqp-http-request-logger:latest" ;
  ns1:description "This is a receiver for streaming iot data" ;
  ns2:ContainerResourceLimits <https://catalogue.ac3-
project.eu/set/resource/service/d8a6b514-3b80-48ec-90cd-d4a229736fb3/resourceLimits> ;
  ns0:microservicesSLA <https://catalogue.ac3-project.eu/set/resource/service/d8a6b514-3b80-
48ec-90cd-d4a229736fb3/sla> ;
  ns1:name "streaming-logger" .

<https://catalogue.ac3-project.eu/set/resource/service/d8a6b514-3b80-48ec-90cd-
d4a229736fb3/resourceLimits>
  a ns2:ContainerResourceLimits ;
  ns2:cpuRequirements 1 ;
  ns2:memoryRequirements 1024 ;
  ns2:gpuRequirements 0 .

<https://catalogue.ac3-project.eu/set/resource/service/d8a6b514-3b80-48ec-90cd-
d4a229736fb3/sla>
  a ns0:microservicesSLA ;
  ns0:serviceAvailability 99.9 ;
  ns0:maxResponseTime "Medium" ;
  ns0:dataThroughput "High" .
```

Figure 11. UC1 Logger

For the UC1 mapper (Figure 12), we define a service based on the "sparkworks/sw-mapper-ac3:0.5" Docker image, published in the official Docker Hub. This application requires a set of environmental variables to be configured and executed, as well as the exposure of a set of TCP ports to facilitate communication with the message broker and other services. Regarding environmental variables, the service requires the following data:

- RABBITMQ_HOST: Specifies the RabbitMQ message broker host.
- RABBITMQ_PORT: Defines the port for RabbitMQ communication.
- RABBITMQ_USERNAME: Username used for authentication.

© AC³ 2023

- RABBITMQ_PASSWORD: Password for authentication.
- QUEUE_IN: The input queue name where incoming data is received.
- QUEUE_OUT: The output queue name where mapped data is sent.

Additionally, in the service's description we include a set of resource limits that are to be used when launching the service to make sure that the application will fit in the resources available in the deployment environment. The resource limits include CPU, Memory and GPU requirements and are based on the Gaia-X ontology's *ContainerResourceLimits*. Similarly, an SLA entity is added to define SLAs for the service that AC³ should apply. These values include *serviceAvailability*, *maxResponseTime* and *dataThroughput*.

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix ns0: <https://ac3-project.eu/#> .
@prefix ns1: <https://schema.org/> .
@prefix ns2: <https://w3id.org/gaia-x/development#> .

<https://catalogue.ac3-project.eu/set/resource/service/115f7472-194d-446c-9518-
efb9f55d3f73>
  a <https://w3id.org/gaia-x/development#ServiceOffering> ;
  dc:title "Mapper Service" ;
  ns0:EnvironmentVariable "RABBITMQ_HOST=edgebroker", "RABBITMQ_PASSWORD=xyzpass",
"RABBITMQ_USERNAME=mapperuc1", "RABBITMQ_PORT=5672", "QUEUE_OUT=mapperuc1.mapped",
"QUEUE_IN=mapperuc1.data" ;
  ns0:ExposedPort "5026:5026", "8026:8026" ;
  ns0:image "sparkworks/sw-mapper-ac3:0.5" ;
  ns1:description "This is a mapper service for live IoT data." ;
  ns1:name "edgemapper" .

<https://catalogue.ac3-project.eu/set/resource/service/115f7472-194d-446c-9518-
efb9f55d3f73/resourceLimits>
  a <https://w3id.org/gaia-x/development#ContainerResourceLimits> ;
  ns2:cpuRequirements 0.5 ;
  ns2:memoryRequirements 512 ;
  ns2:gpuRequirements 0 .

<https://catalogue.ac3-project.eu/set/resource/service/115f7472-194d-446c-9518-
efb9f55d3f73/sla>
  a ns0:microservicesSLA ;
  ns0:serviceAvailability 99.9 ;
  ns0:maxResponseTime "Medium" ;
  ns0:dataThroughput "High" .
```

Figure 12. UC1 Mapper

3.4.3 Data Source Deployment

The Data Source of UC1 is interfaced with AC³ using an EDC Connector developed based on the Eclipse EDC connector samples. This provider connector is deployed as a containerized application at the testbed's infrastructure and is interfaced with the locally running MQTT broker that aggregates the data generated by the IoT devices. Once the connector is deployed, a new asset is registered using a set of parameters that define the source of the incoming IoT updates (MQTT server host, port, username, password, and topic), and the ID of the asset that will be used for this source. This can be accessed then through the provider's management API:

```
{
  "@context": {"@vocab": https://w3id.org/edc/v0.0.1/ns/},
  "@id": "uc1-stream",
  "properties": {
     "name": "iot-data",
     "type": "streaming",
   }
}
```

A policy must then be defined on the provider side to govern access to UC1 data. A JSON payload is used to define a policy that permits the usage of the asset:

```
{
  "@context": {
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  },
  "@id": "no-constraint--1",
  "policy": {
    "@type": "odrl:Set",
    "odrl:assigner": {
      "@id": "provider"
    },
    "odrl:target": {
      "@id": "asset-1"
    },
    "odrl:permission": [],
    "odrl:prohibition": [],
    "odrl:obligation": []
  }
}
```

Next, a contract definition is created to govern the data transfer. A JSON payload is used to define the contract terms, linking the asset to a policy that permits access:

```
{
   "@context": {
     "edc": "https://w3id.org/edc/v0.0.1/ns/"
   },
   "@id": "contract-definition",
   "accessPolicyId": "no-constraint-policy",
   "contractPolicyId": "no-constraint-policy"
 }
```

The consumer connector then needs to initiate the negotiation of the data transfer to the UC1 AC³ application. This is achieved using a request payload for the specific asset:

```
{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  },
  "@type": "NegotiationInitiateRequest",
  "counterPartyAddress": "http://ds.uc1.ac3.sparkworks.net:8282/protocol",
```

```
  "protocol": "dataspace-protocol-http",
  "offer": {
    "offerId": "..."
  },
  "policy": {
    "@id": "",
    "@type": "odrl:Offer",
    "odrl:assigner": {"@id": "provider"},
    "odrl:target": {"@id": "uc1-stream"},
    "odrl:permission": [],
    "odrl:prohibition": [],
    "odrl:obligation": []
  }
}
```

Once the negotiation is finalized (status FINALIZED), the response provides a contract agreement ID, which is used to initiate the data transfer. A JSON payload is used to initiate the data transfer through the consumer's management API:

```
{
    "@context": {
        "edc": "https://w3id.org/edc/v0.0.1/ns/"
    },
    "@type": "TransferRequest",
    "connectorId": "provider",
    "counterPartyAddress": "http://ds.uc1.ac3.sparkworks.net:8282/protocol",
    "protocol": "dataspace-protocol-http",
    "contractId": "...",
    "assetId": "uc1-stream",
    "transferType": "HttpData-PUSH",
    "dataDestination": {
        "type": "HttpData",
        "baseUrl": "http://ionos-s1.sparkworks.net:4000"
    }
}
```

Once the transfer process is initiated, the data starts flowing to the baseUrl provided in the request noted above. The logger application that is executed there is capable of decoding the received data and forwarding it to the AC³ application for further processing.

3.4.4 Application Descriptor – OSR

The Graphical User Interface (GUI) and Ontology Semantic Reasoner (OSR) have been successfully integrated to support the definition, composition, and deployment of applications within UC1. This integration allows developers to define their application requirements through a structured and intuitive GUI, while the OSR automates the translation of these inputs into a machine-readable YAML-based AppD. This mechanism ensures that the required configurations, inter-service dependencies, and deployment parameters are correctly structured and semantically validated.

The snippets below present a high-level structure of the AppD, which includes metadata, microservices configuration, network interconnections, global SLA requirements, and deployment constraints. This structure is generated automatically based on the user input, ensuring both flexibility and compliance with deployment expectations.

To demonstrate how microservices are defined in practice, the code snippets below provide an example microservice from UC1, detailing its resource requirements, environment variables, SLA targets, and port mappings. This structure ensures that each service is configured accurately to match deployment conditions, including location-specific constraints (e.g., edge or cloud).

In scenarios where data sources are involved, like UC1, the OSR queries the Piveau catalogue to fetch relevant services and connectors. The following snippets illustrate an example of a microservice automatically extracted from Piveau. These data-related services are appended to the descriptor to ensure the application can access and process data as expected.

This end-to-end flow streamlines the deployment process across the Cloud-Edge Continuum (CECC), reducing manual overhead while supporting flexible, scalable application design.

```
ApplicationName: "UC1 IoT Data Processing"
Version: "3.0"

Microservices_configuration:

Global_SLA:
       ServiceAvailability: "99.9%"
       MaxLatency: "500 ms"
       MaxResponseTime: "Low"
       DataThroughput: "High"
```

```
Microservices_configuration:

          MicroserviceName: "edgeapplication"
          Version: "0.4"
          Image: "sparkworks/data_manipulator_uc1:0.4"
          ID: "edgeapplication"
          ResourceRequirements:
                 Cpu: "4 vCPUs"
                 Memory: "8Gi"
          ReplicaCount: "1"
          Ports:
                 "5001:5001"
          EnvironmentVariables:
                 Name: "RABBITMQ_PORT" Value: "5672"
                 Name: "RABBITMQ_HOST" Value: "edgebroker"
                 Name: "RABBITMQ_USERNAME" Value: "m1"
                 Name: "RABBITMQ_PASSWORD" Value: "7Iqk7uu1Ot"
                 Name: "QUEUE_OUT" Value: "mapperuc1.processed.ml"
                 Name: "QUEUE_IN" Value: "mapperuc1.mapped.ml"
```

© AC³ 2023

```
Microservices_configuration:
- MicroserviceName: "consumer"
    Version: "latest"
    Image: "sparkworks/ac3-connector-http-http-consumer:latest"
    ID: "consumer"
    Ports:
      - "28180:28180"
      - "28181:28181"
      - "28182:28182"
      - "28183:28183"
    EnvironmentVariables:
      - Name: "WEB_BASE_URL"
        Value: "http://ionos-s1.sparkworks.net"
      - Name: "WEB_HTTP_PORT"
        Value: "28180"
      - Name: "WEB_HTTP_MANAGEMENT_PORT"
        Value: "28181"
      - Name: "WEB_HTTP_PROTOCOL_PORT"
        Value: "28182"
      - Name: "WEB_HTTP_CONTROL_PORT"
        Value: "28183"
      - Name: "ASSET_NAME"
        Value: "uc1-stream"
      - Name: "PROVIDER_DOMAIN"
        Value: http://ds.uc1.ac3.sparkworks.net:18182/protocol
```

3.4.5 LCM

The UC1 integrates Eurecom's Lightweight Edge Slice Orchestration (LiSO) component to efficiently manage the lifecycle of its dynamic, container-based applications, along with the robust Cloud Edge Continuum infrastructure. This strategic decision not only strengthens the orchestration framework already established in UC2 but also facilitates a seamless integration of processes and encourages the reusability of resources across a wide range of UCs within the AC³ architecture.

3.4.5.1 Application Deployment

LiSO consists of two primary orchestration layers and an image registry, as defined in D2.3 [6]. It is integrated into UC1, as depicted in Figure 13 which illustrates a simplified architecture of UC1 to highlight the LCM integration:

- **The Service Orchestration Layer** [6] from LiSO is responsible for translating the UC1 AppD generated by the OSR into Resource Level Objects (RLOs) such as K8s deployment specifications or helm charts. This translation is a crucial step that enables the seamless deployment of the UC1 microservice-based application on the edge or cloud LMS. By effectively transforming the AppD into RLOs, LiSO ensures that the application can fully utilize the resources available in the Cloud Edge Continuum.
- **Resource Orchestration Layer (ROL) i**nterfaces directly with the edge and cloud K8s LMS using a plugin mechanism through the LMS northbound interface (NBI), executing resource-level operations such as deployment, scaling, and termination.

- Since LiSO fetches and stores container images specified via URLs in the AppD/VNFD files for seamless service deployment, a **Container Image Registry** is deployed in the Edge Domain as shown in Figure 13: Overview of UC1 Architecture, highlighting the integration of LiSO's orchestration layers and image registry.

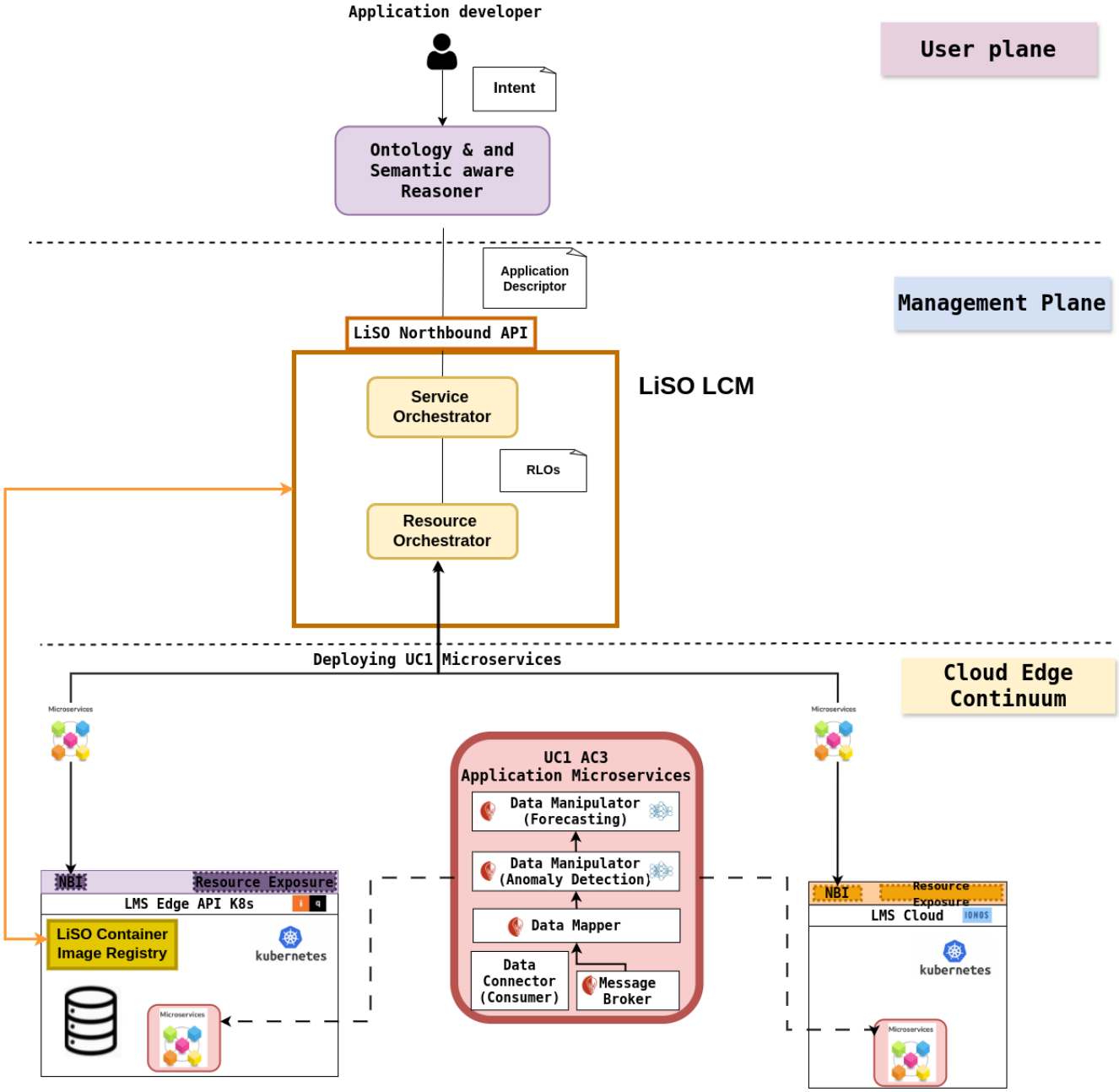

Figure 13: Overview of UC1 Architecture, highlighting the integration of LiSO's orchestration layers and image registry

Note that more details are given in section 4.3.3 about the integration of LiSO with OSR.

3.4.5.2 Application Adaptation

As part of our **Application Adaptation** process, UC1's LMS must seamlessly join the AC³ architecture. To achieve this, we leverage Eurecom's LiSO architecture, its ROL component allows the LMS to register asynchronously in a collaborative, multi-stakeholder environment. Figure 13 illustrates how ROL interacts directly with the northbound interface (NBI) of edge domain, enabling the LMS to adapt and integrate dynamically into the overall continuum.

In UC1, the LMS is implemented using K8s, the industry-standard platform for container orchestration. This setup enables efficient resource management and microservice deployment at the edge and cloud domains in UC1. K8s oversees both computing resources and the lifecycle of services within the edge domain, ensuring high availability and fault tolerance.

The LMS in UC1 is responsible for orchestrating and running containerized applications dynamically based on local resource availability and workload demands. By using K8s, UC1 can take advantage of K8s robust orchestration features, such as automatic scaling, self-healing (i.e., auto-restarting failed containers), and resource-based scheduling to optimize local edge domain operations.

The edge and cloud LMSs work closely with the LiSO LCM, which in turn interacts with the OSR component to obtain AppDs and utilize the instructions to deploy microservices. This setup facilitates the deployment of applications using these AppDs which is a key output of the User Plane.

This integration significantly enhances the system's capacity to speed up and adapt swiftly to evolving workload demands, a vital feature for UC1, particularly in scenarios such as real-time environmental data processing and occupancy detection. Consequently, this advancement enhances overall efficiency and facilitates real-time execution and processing within UC1.

3.4.5.3 Migration Algorithm

One of the key components enhancing the architecture is the migration block, which should be integrated into the LiSO LCM (as shown in Figure 13). The migration module continuously monitors cloud-edge computing resources to identify hidden patterns that can optimize service allocation between cloud and edge servers. It is powered by a reinforcement learning (RL) algorithm introduced in Section 5.2 of D3.2.

Following the initial service placement, the migration module proposes resource reallocations to ensure the application's latency requirements are maintained, especially when resource utilization begins to degrade. By leveraging real-time data on computing resources at both the cloud and edge levels, along with network latency metrics, the migration block drives its decisions. The RL algorithm learns and adapts to select the most suitable server for migration, ensuring consistently low latency performance.

3.4.5.4 AI-based LCM and Decision Enforcement Algorithms

As part of the AC³ integration strategy, UC1 incorporates Zero-touch configuration and application management capabilities. The approach is built upon specifically the work initiated in Sections 5.3.2 and 5.3.3 of D4.1 [4] based on the two distinct XAI-enabled algorithms developed in WP4. These algorithms can offer UC1 predictive and

© AC³ 2023

explainable decision-making capabilities, enabling detailed resource control, proactive scaling, and SLA preservation.

XAI-Enabled Fine Granular Resources Autoscaler

This algorithm provides a fine-grained vertical autoscaling mechanism for the UC1 application. It leverages eXplainable AI (XAI) techniques to analyze resource usage trends in relation to their limits and predict when resources should be adjusted to maintain performance.

The most relevant elements to be integrated into UC1 are: (1) an ML predictor based on XGBoost to detect potential QoS degradation by analyzing CPU and memory usage patterns; (2) an explainability module using SHAP to identify the most influential resource metrics contributing to performance issues; and (3) a decision Engine that leverages SHAP insights to trigger targeted, fine-grained vertical scaling actions (CPU, memory, or both), ensuring efficient resource utilization and SLA compliance.

The following outlines several key benefits:

- Explainability: Operators understand why decisions are made.
- Efficiency: Resources are scaled only when and where needed.
- Zero-touch: Autonomous operations reduce human intervention.

XAI for Prediction of Infrastructure Usage

This strategy may be incorporated into UC1 to accurately predict future resource utilization (CPU, memory) at the infrastructure level, facilitating proactive and optimized management decisions through clear and understandable AI models.

The aim of integrating this algorithm is to reach the following benefits:

- Reduces system downtime or degraded performance by anticipating overload scenarios.
- Enables resource-aware scheduling, improving overall edge efficiency.

3.5 Remaining Integration

Working on the next steps of UC1, we can summarize them in the following:

- Deployment of the UC1 cloud computing location. This is needed for the evaluation of the “Seamless Microservice Deployment and Migration” and the “Time to process and react to sensor data” scenarios. The process will follow a similar approach to the one followed in our edge deployment, with a K8s installation orchestrated using the same tools. The main difference would be the extra resources made available for our installation, as it would be capable of processing larger volumes of data.
- Deployment of edge and cloud computing location monitoring applications. This is needed for the “Seamless Microservice Deployment and Migration” and “AI-powered Infrastructure Monitoring & Control Service at the Edge” scenarios. Using the monitoring information, the CECCM would be able to decide when and what type of migration is needed between the cloud and edge compute locations.
- Comparative analysis of the behavior of the UC1 application in the two testbed locations. This is part of the “Time to process and react to sensor data”. Once we can deploy our application at both locations, we will be able to assess how much data each location can process and handle, and what the benefits

© AC³ 2023

are from moving the computation closer to the data sources of our application, instead of transferring them to the cloud.

- Evaluate the deployment of the UC1 application to the edge and cloud locations of our testbed. This is part of the "Zero-touch configuration, application management, and data management", allowing us to understand how much easier the process of deploying our application is.

3.6 UC1 Integration summary

The table below offers an overview of the integration status for UC1. It summarizes the key components and their progress, highlighting both the advancements made and the areas still in development.

Architecture component	Sub-Component	Description	Integration status
Application gateway (GUI)		Allow the application developer to define its application components and SLA.	In Progress
OSR		Allow the generation of the AppD	In Progress
LMS Edge		We will execute the micro-services that run at the network's edge for lower latency and bandwidth optimisation.	In Progress
LMS Cloud		Will execute the micro-services that cannot run at the network's edge due to high resource usage or data volumes, and latency is not a constraint. Similarly, for network unavailability at the edge.	In Progress
Catalogues		Hosts the component templates for the application	Complete
Application and resource management	AI-based LCM and Decision Enforcement	1. Manage the microservices Life Cycle 2. Migration algorithm that adapts if the edge resource degrades or moves to the edge a micro-service	1. In progress 2. In progress
	Zero-touch configuration and application management, data management	Predict and describe infrastructure resources and implement automated corrective measures.	Not started

	AI-Based Resource profile	Describe the resources of the infrastructure	Not started
	AI-Based Application profile	Predicting Application Behaviour	Not started
	Monitoring	Monitoring the micro-services KPI	Not started
Data Management	Catalogues	Provides descriptions for the available data sources	Complete
	Data Provider Connector	Provides access to the data made available	Complete
	Data Consumer Connector	Initiates the streaming of data from the data source to the application micro-services	Complete
	Data Mappers	Transforms incoming data as needed	Complete
	Data Manipulator	Core application logic	Complete
	Message Broker	Responsible for transferring data between application components	Complete

Table 3: UC1 Integration summary

This summary aims to provide a clear snapshot of the current integration status and the steps taken toward the full integration of the UC within the project.

4 UC2

4.1 Use Case Description

UC2 focuses on the deployment of a sophisticated Smart Monitoring System designed to enhance urban security, traffic management, and environmental surveillance. This system integrates UAVs, IoT technologies, AI/ML algorithms, and edge computing to enable intelligent, responsive, and efficient monitoring capabilities within a smart city context.

UC2 demonstrates how real-time data from UAV-mounted cameras and distributed IoT sensors can be processed using edge computing resources and analysed through AI-driven techniques. The CECCM plays a central role in managing this distributed infrastructure, enabling the seamless execution of surveillance and monitoring applications across the cloud-edge continuum. The system supports both live video streaming and video-on-demand (VoD) functionalities, with the ability to store video footage for further analysis and insights generation. Ultimately, UC2 exemplifies a powerful integration of technologies to support data-driven decision-making for urban environments.

4.1.1 Use Case Objectives

The objectives of UC2 align closely with the goals of the AC³ framework, focusing on flexible deployment, intelligent processing at the edge, and automated management of microservice-based applications. The key objectives include:

- Simplified Application Definition: Use the GUI and OSR to define, manage, and deploy microservice-based applications easily.
- Zero-Touch Management and Orchestration: Leverage CECCM for autonomous configuration, life-cycle management, and microservice orchestration using AI/ML.
- Flexible Behaviour Reconfiguration: Dynamically switch between functionalities like object tracking, activity detection, and surveillance using semantic requests.
- Resilient Microservice Deployment: Enable optimal placement and seamless migration of services across UAVs and edge nodes in response to resource availability.
- Edge Intelligence and Analytics: Run AI models (e.g., Deepstream, YOLO) on UAVs and edge devices for real-time object detection, behaviour analysis, and traffic monitoring.
- Environmental Monitoring and Response: Use UAV-mounted IoT sensors for real-time tracking of conditions such as CO_2 and temperature.
- Scalable and Adaptive Operations: Automatically manage workloads and scale services across distributed cloud-edge resources to meet changing demands.

4.1.2 UC2 Stakeholders

4.1.2.1 Users / System Administrators

The primary users of the deployed UC2 application include stakeholders responsible for urban surveillance and public safety. These users include building security personnel in residential complexes (e.g., flat surveillance teams), municipal surveillance operators, and city infrastructure monitoring units. In cases of unusual events or security alerts, law enforcement agencies such as the police may also access the system to review live video

feeds or retrieve evidence from stored footage. These stakeholders rely on the system for configuring edge devices, viewing live streams, receiving real-time alerts, and analysing labelled video or sensor data to ensure situational awareness and rapid response.

4.1.2.2 Application Developers / DevOps

Application developers are responsible for defining and building the microservices required for UC2 surveillance tasks, such as video streaming, object detection, and telemetry analysis. Using the CECCM's GUI, they define AppD's that specify service components, resource needs, and deployment policies. The DevOps team handles the deployment process, leveraging the CECCM's LCM system to orchestrate service placement across cloud, edge, and far-edge nodes. This reduces manual deployment effort and ensures high availability and scalability through automated lifecycle management, including service migration and fault tolerance.

4.1.2.3 Infrastructure Provider

The infrastructure provider supplies and maintains the CECCM platform used in UC2. This includes the setup and integration of core components such as the GUI, OSR, LCM, and LMS across cloud and edge environments. The provider ensures that these components are fully operational and compatible with K8s, K3s, and SD-WAN networking technologies, enabling seamless deployment and orchestration of microservices for the application developers and DevOps teams.

4.2 Use Case Architecture

The architecture of UC2 is designed to support AI-driven video surveillance by leveraging the AC³ framework. It integrates UAV-mounted and stationary IoT devices with microservice-based video analytics components, orchestrated through the AC³ framework. The system enables distributed processing and real-time insights across far-edge, edge, and cloud environments. The architecture follows a modular and scalable approach, relying on containerized services managed by K8s-based orchestration layers and interconnected through SD-WAN to ensure seamless communication and deployment flexibility.

4.2.1 Use Case Application

The architecture of UC2 involves a smart monitoring system built on UAVs, IoT devices, and video analytics using AI at the edge. The core components include a system administrator interface, edge devices (Raspberry Pi and Nvidia Jetson), a video analytics system powered by DeepStream and ML, and the AC³ framework. The administrator interface allows configuration and control of edge devices, including monitoring live streams, receiving real-time alerts, and viewing segmented images for telemetry analysis. Through this interface, system administrators interact with the system's frontend, backend, and analytics microservices, performing actions like adding or deactivating devices.

At the far edge, UAVs are equipped with Nvidia Jetson devices for on-device video analysis and AI processing, while Raspberry Pi units handle streaming without analytics capabilities. The video analytics system uses DeepStream and YOLO for real-time object detection and activity recognition, supporting both live and VoD functionalities. The AC³ framework orchestrates the infrastructure, managing computing resources across the cloud-edge continuum. It enables service relocation, fault tolerance, and latency reduction through AI-based orchestration and lifecycle management of microservices. Figure 14 illustrates this setup, where video processing is efficiently managed across a central server, regional edge servers, and far-edge devices deployed in the parking lot.

© AC³ 2023

4.2.1.1 Application Components

The UC2 application is composed of multiple microservices deployed across a distributed infrastructure that includes cloud, edge, and far-edge nodes. These services are containerized and managed using K8s and K3s for scalability and orchestration. The central server is the key component responsible for handling CRUD operations related to users, devices, and regions, with data persistence handled via an SQL database such as PostgreSQL. It serves the frontend API, which allows users and administrators to register devices, launch video streams, upload videos, and query telemetry data.

The system also includes a regional server that manages IoT devices and cameras within designated areas. This server operates as a reverse proxy and performs local video processing using Nvidia DeepStream. It plays a critical role in enabling federated learning, where models deployed on regional servers and IoT devices are aggregated into a global model maintained at the central server.

At the far edge, UAVs are equipped with Nvidia Jetson devices capable of performing on-device AI and real-time video analytics, while Raspberry Pi units are limited to video streaming due to lower processing capabilities. Microservices requiring high computing resources or low latency are placed on the LMS Edge using the K8s API, while those with lightweight requirements, such as video capture and object detection, are deployed on UAVs using K3s. Frontend services are hosted on the LMS Cloud (IONOS Cloud), and the networking between these distributed nodes is managed by an SD-WAN controller.

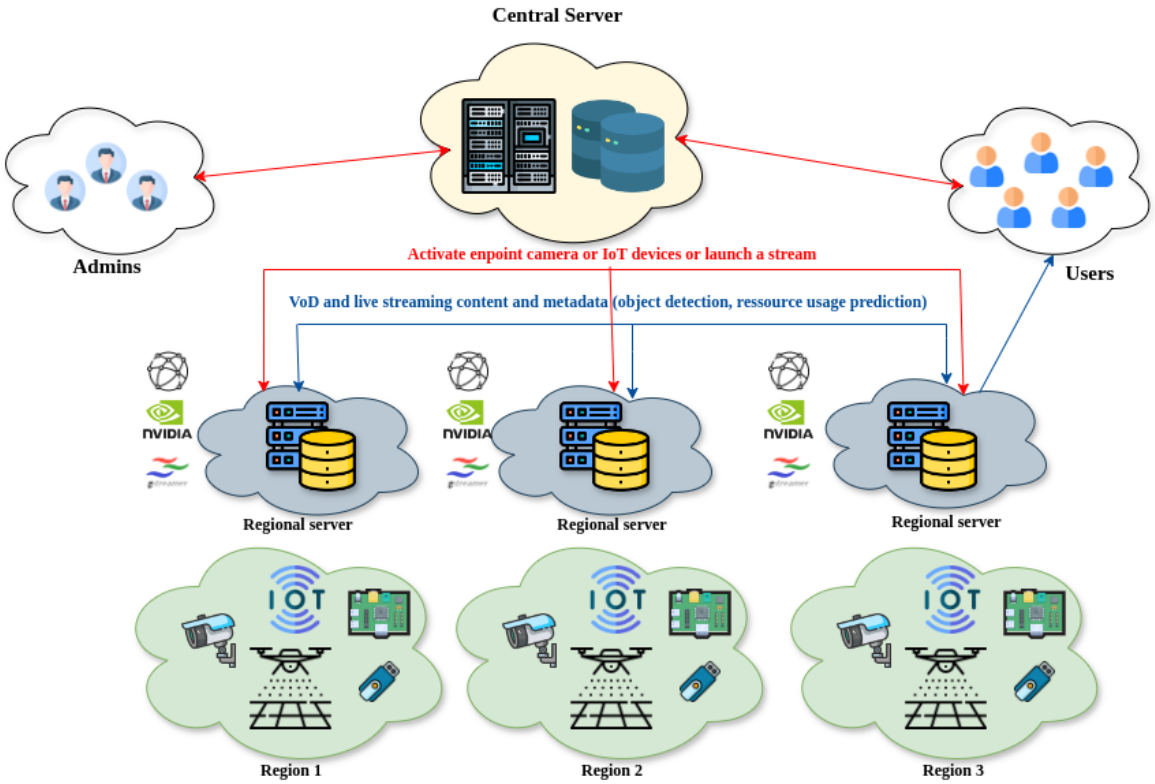

Figure 14. Architecture of the video surveillance and environmental monitoring system, illustrating data processing across central servers, regional edge nodes, and far-edge devices in an urban parking lot.

4.2.1.2 Component Interaction

The interaction between microservices in UC2 follows a structured flow orchestrated by the AC³ framework. The process begins with the application developer, who uses a GUI to define the application components, their interdependencies, and SLA requirements. This information is submitted in the form of an AppD, which is then processed by the OSR. The OSR interprets the semantics of the descriptor, validates dependencies, and ensures policy compliance before handing it over to the LCM.

The LCM coordinates the deployment of services to the appropriate infrastructure layer—cloud, edge, or far edge—based on available resources and runtime policies. It manages the full lifecycle of microservices, including initial placement, scaling, migration, and failure recovery. To interconnect the distributed computing nodes, the LCM uses the SD-WAN controller through its NBI to establish flow configurations and ensure seamless communication across the continuum. Monitoring tools are integrated to track microservice KPIs, while AI-based algorithms are used for placement decisions and predicting application and infrastructure behavior. For example, if a far-edge node's resources degrade, the system can trigger a migration to maintain service continuity.

The communication between distributed components is facilitated by LMS Networking via the SD-WAN controller, which enables connectivity across cloud, edge, and far-edge clusters. This setup ensures that services can interact reliably despite dynamic resource conditions. Metadata from object detection, sensor readings, and activity analysis is processed and fused across these layers, allowing administrators and users to access real-time insights via the frontend API. Access control is enforced by a dedicated authorization service, ensuring that only permitted users can interact with specific components or data streams.

4.2.2 UC Testbed – Hardware and Software

As previously mentioned, the UC2 application is based on a microservices architecture and will be deployed across a multi-cluster environment connected between them using SD-WAN. In this section, we describe the hardware and software setup of the testbed used for this deployment. The whole picture is presented in Figure 15. The testbed spans two geographically distinct locations: the IONOS cloud data centre located in Germany, and EURECOM's edge and far-edge infrastructure located in France.

4.2.2.1 Infrastructure Details

IONOS Region

The IONOS infrastructure consists of three main clusters and is built on machines with 32 GB of RAM, 16 CPU cores, and approximately 120 GB of HDD storage. All nodes run Ubuntu 22.04 as the operating system.

- Management Cluster:
  - This is a single-node cluster with 4 CPU cores, 8 GB of RAM, and 20 GB of HDD storage. It runs a vanilla K8s setup and is dedicated to hosting core management software such as the AI-based LCM (LiSO, in this UC) and the SD-WAN controller.
- SD-WAN Edge Cluster:
  - Also, a single-node setup, this cluster has 4 CPU cores, 8 GB of RAM, and 20 GB of HDD storage. It runs the SD-WAN edge component, enabling connectivity between the cloud cluster and other infrastructure setups.

© AC³ 2023

- Cloud Cluster: This is a two-node cluster
    - Control Plane Node: Equipped with 4 CPU cores, 8 GB of RAM, and 64 GB of HDD storage, this node hosts core K8s components (such as the API server) along with AC³ components like the K8s LMS and the VIM adaptation agent. A local DNS server is also deployed to manage domain name resolution for external services.
    - Worker Node: This node, with 10 GB of storage, is dedicated to deploying user-facing microservices such as the NGINX reverse proxy and the application frontend.

This two-node design was chosen to separate the execution of application components from AC³ management components.

The cloud cluster is connected to the SD-WAN cluster through a local area network (LAN) provided by IONOS.

EURECOM Region

The second deployment region is hosted by EURECOM and is composed of two sub-regions: the EURECOM Edge and the EURECOM Far Edge.

EURECOM Edge

The EURECOM Edge infrastructure consists of approximately 500 GB of storage, 68 GB of RAM, and 22 CPU cores, distributed across two clusters:

- ***SD-WAN Edge Cluster:***

This cluster is a single virtual machine with 2 CPU cores, 4 GB of RAM, and 20 GB of storage. It is dedicated to running the SD-WAN edge component, providing connectivity between the EURECOM edge and other regions.

- ***Edge Cluster:***

This is a single-node cluster running on K3s. It includes 20 CPU cores, 64 GB of RAM, and 480 GB of disk space. It hosts both AC³ components—such as the K3s LMS and the VIM adaptation agent—and selected application microservices, including the database backend. Due to the relatively higher resource availability compared to the far edge, logical isolation between AC³ components and application microservices is maintained using K8s namespaces.

Similar to the IONOS setup, the SD-WAN edge and edge clusters are interconnected via a LAN provided by EURECOM.

EURECOM Far Edge

The second part of the EURECOM region is the EURECOM Far Edge, represented by a single-node K3s cluster running on an NVIDIA Jetson Orin device equipped with a GPU. This node has 64 GB of RAM and 64 GB of storage. This cluster is used to deploy the same AC³ components as in the edge cluster—specifically, the K3s LMS and the VIM adaptation agent—as well as the image processing service. As in the edge setup, logical isolation is enforced using K8s namespaces to separate AC³ components from application-level services. For connectivity, the far-edge cluster, running on the UAV, is connected via a 5G base station deployed at EURECOM, enabling communication with the other parts of the infrastructure.

© AC³ 2023

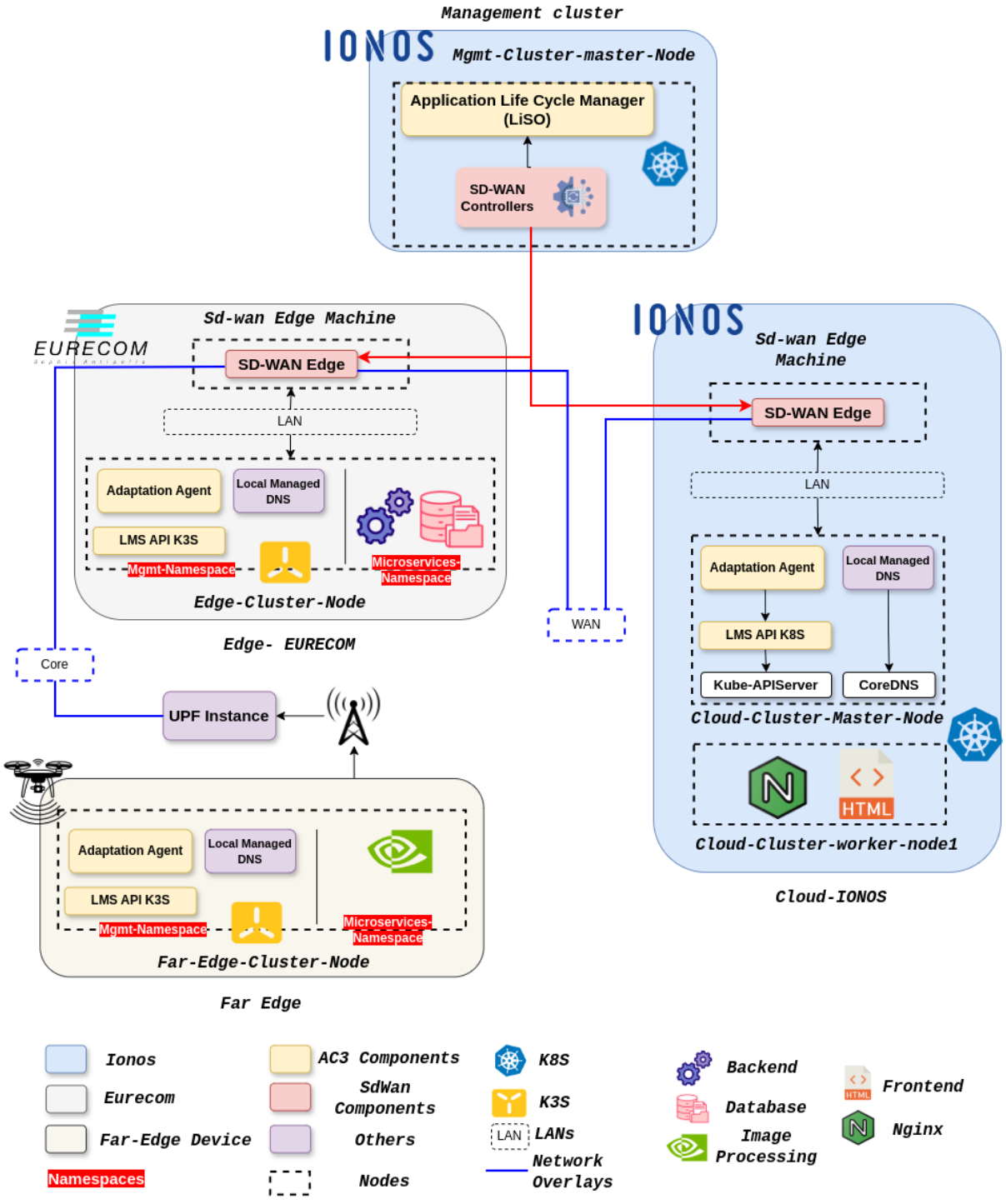

Figure 15. Testbed Architecture

4.3 Component Integration Design

The UC2 smart monitoring application leverages multiple components of the AC³ CECC Manager framework, working together as illustrated in Figure 16, where the implemented AC³ components are highlighted. These components orchestrate UAV-based video analytics and sensor data streams across the cloud–edge continuum.

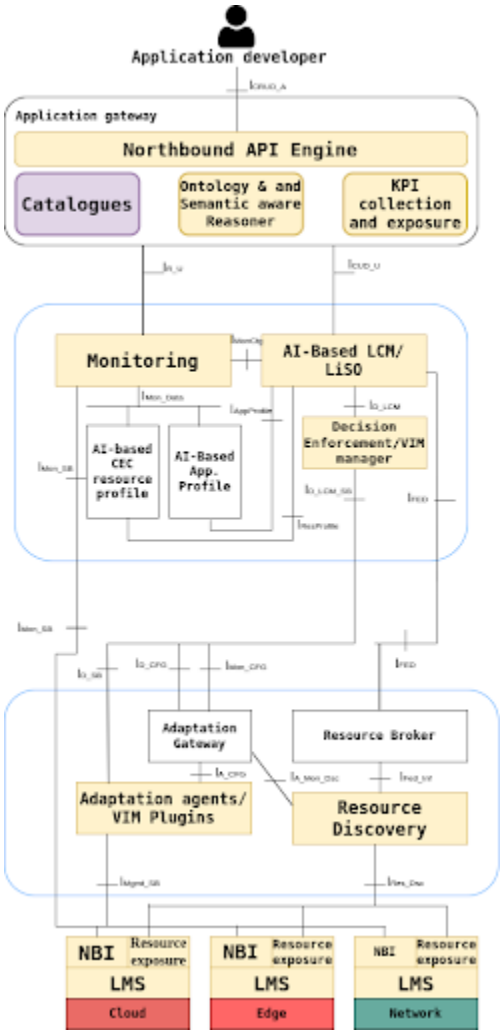

Figure 16. Mapping of the AC³ Component Architecture to the Current Implementation in UC2.

The UC2 application is composed of five microservices, each deployed in a suitable region based on its role and interdependencies. For example, the Frontend microservice needs to be accessible over the internet but does not require low-latency access to other microservices or proximity to camera devices. Therefore, deploying the frontend in the cloud conserves edge resources, which can then be allocated to microservices that require edge deployment. A similar rationale applies to the Nginx microservice, which acts as a gateway by exposing the services to end users through a single point of access.

The DeepStream microservice requires direct access to the camera to stream video and utilizes the NVIDIA Jetson GPU for traffic analysis. It sends both the video stream and the object detection results to the Backend service. Therefore, this microservice is deployed on the drone, which is equipped with both the camera and the Jetson device. The placement of the microservices and their dependencies is illustrated Figure 17.

© AC³ 2023

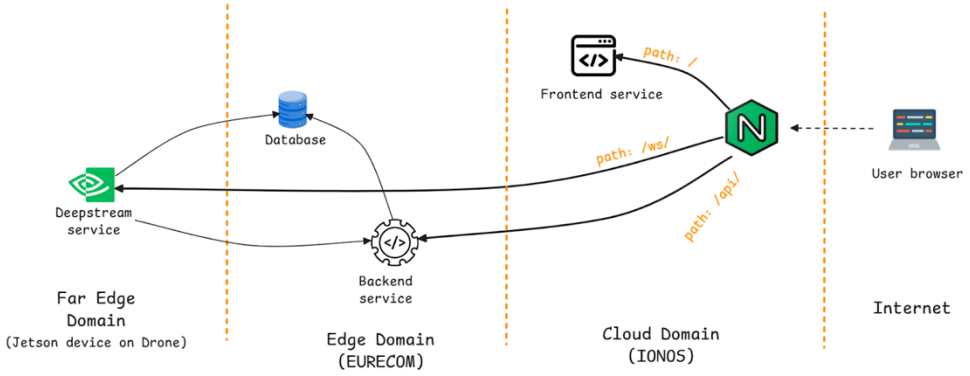

Figure 17. Microservice Architecture and Placement in UC2

To deploy the application, the implemented CECC Manager components are used, as illustrated in Figure 18. First, the application developer uses the northbound API (via the GUI) to define the UC2 application using the available application and data source blueprints. Once the application composition is submitted, the request is processed by the OSR. The OSR validates the UC2 AppD's and policies, then, merges the service and data definitions into a single AppD, ensuring that all semantic rules and requirements are met.

Next, the application creation request is forwarded to the LCM, which selects the appropriate infrastructure (cloud or edge) for each microservice and initiates the application onboarding process in the selected regions. To achieve this, the LiSO LCM sends requests to the Virtual Infrastructure Manager (VIM), which acts as the Decision Enforcement component.

Each deployment site has a corresponding Adaptation Agent, implemented as a VIM. The latter manages the local management systems (e.g., K8s and K3s) and image registries to onboard the microservice images. Once the application is onboarded, LiSO configures the inter-cluster network via the SD-WAN Controller, ensuring that microservices deployed across different regions can communicate with each other.

Following onboarding, LiSO begins the application instantiation process. As with onboarding, LiSO's requests pass through the VIM Manager and are executed by the VIM to instantiate the microservices across the three target clusters: IONOS (Cloud), EURECOM Edge, and onboard the drone (i.e., far edge) equipped with an NVIDIA Jetson device.

Finally, LiSO completes the network configuration by exposing the user-facing microservices to the internet through the SD-WAN Controller, which updates the SD-WAN Edge network rules accordingly. Once the application is fully instantiated, LiSO returns information about the running instances, including IP addresses, ports, and the links to access the internet-exposed microservices.

While the application is running, the Monitoring System, described in the next section, continuously collects performance metrics from the deployed services and infrastructure; for example, container CPU/memory and streaming framerates from the Jetson nodes. These metrics feed back into the LCM to support application adaptation via resources scaling or microservices migration.

In summary, the CECCM components (Service Catalogue, OSR, LCM, Monitoring and Data Management) enable the UC2's video analytics services to run continuously and migrate seamlessly across UAV, far-edge, and cloud nodes as conditions change.

© AC³ 2023

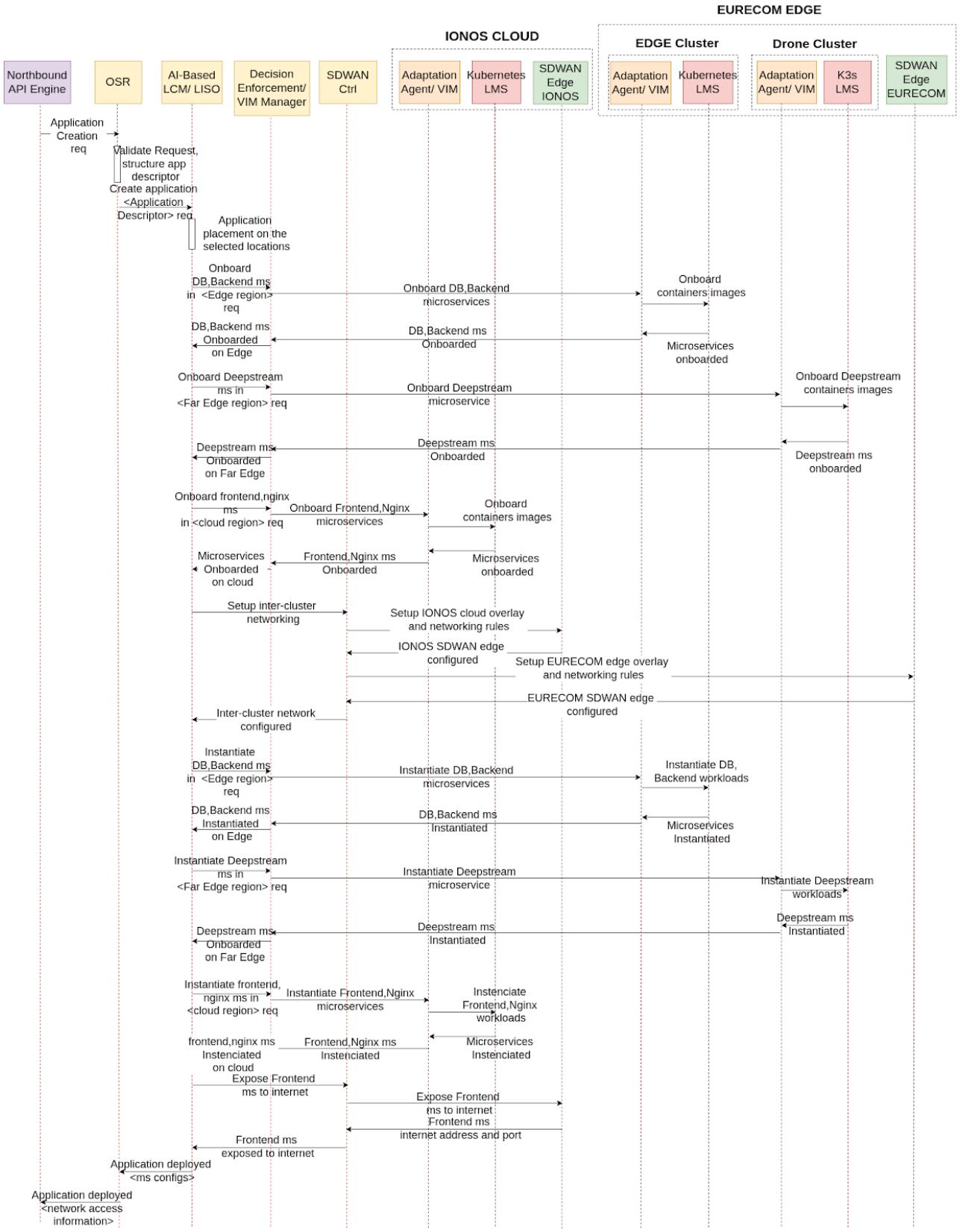

Figure 18. Detailed Workflow for UC2 Application Deployment Using the AC³ CECC Manager Framework

4.4 Component Integration Status

4.4.1 Application Interface

The user-facing application developed for UC2 offers a comprehensive and interactive interface designed to manage the UAVs, monitor system performance, and review real-time and historical detections. The interface has been developed with usability in mind, providing a smooth user experience for system administrators and operators.

Figure 19 shows the user authentication screen, which provides a secure login experience enhanced with CAPTCHA verification to ensure bot protection.

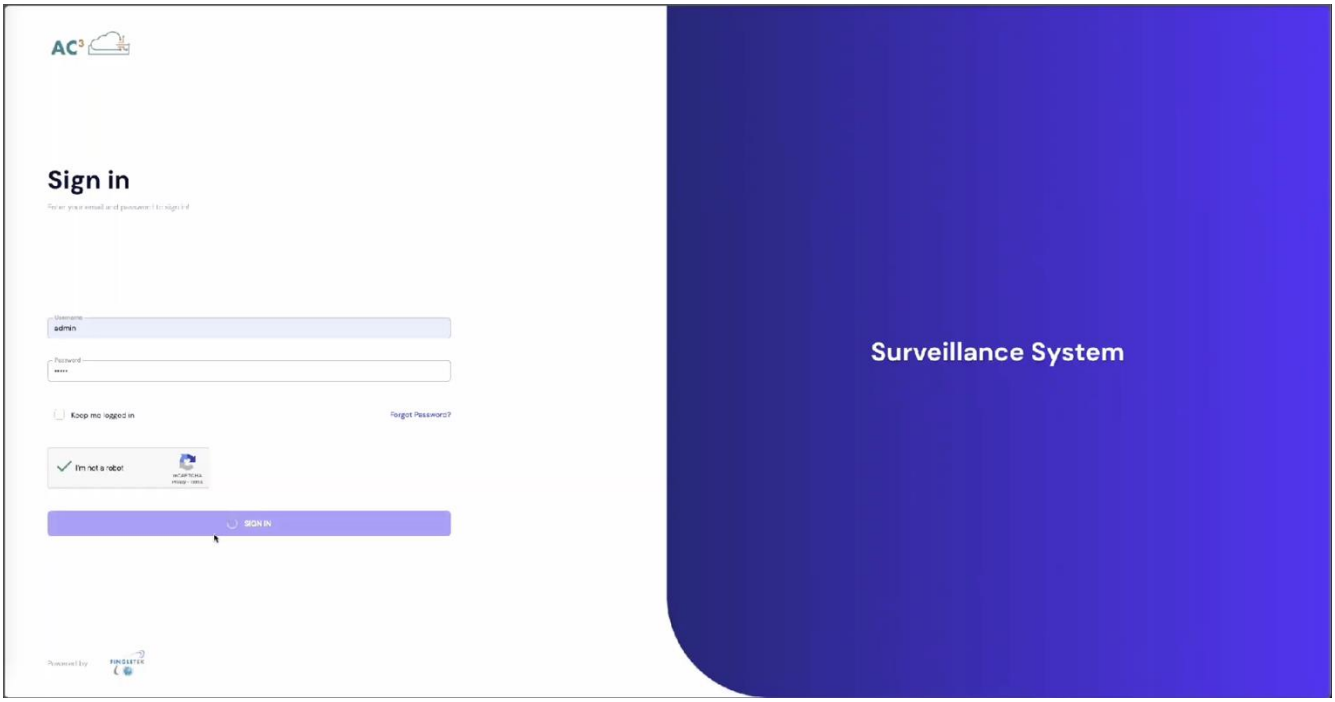

Figure 19. UC2 User Authentication screen

After successful login, users are directed to the Regions dashboard, as illustrated in Figure 20. Here, they can create and manage edge servers by filling in configuration details such as IP address and status.

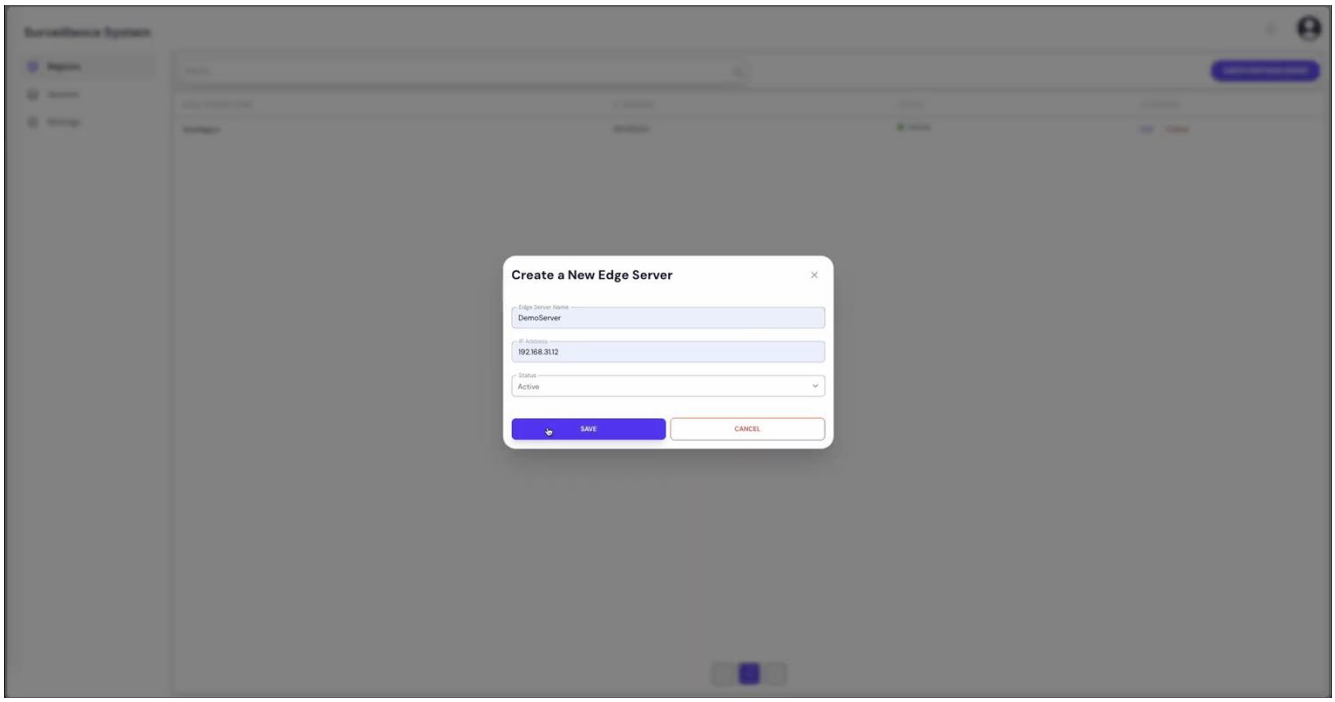

Figure 20. UC2 Edge Server configuration screen

Figure 21 demonstrates the process of adding a far edge device to an existing edge server. Users can define the device's location, type, and associated sensors through a structured form. Once saved, the devices appear in the system and can be controlled from the interface.

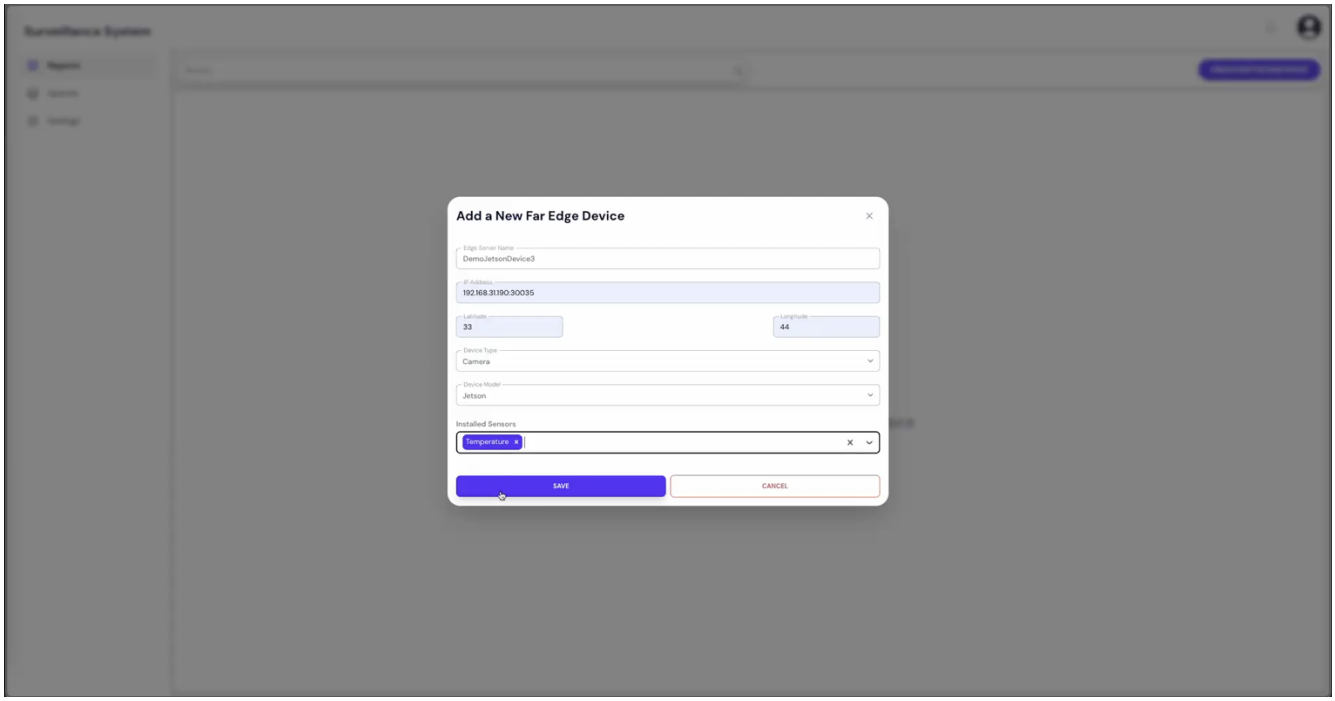

Figure 21. UC2 Far Edge Server configuration screen.

Figure 22 presents the device management screen, where users can view the status of devices, including their IP address and operational status. Control actions such as viewing live data or deactivating a device are available directly from this interface.

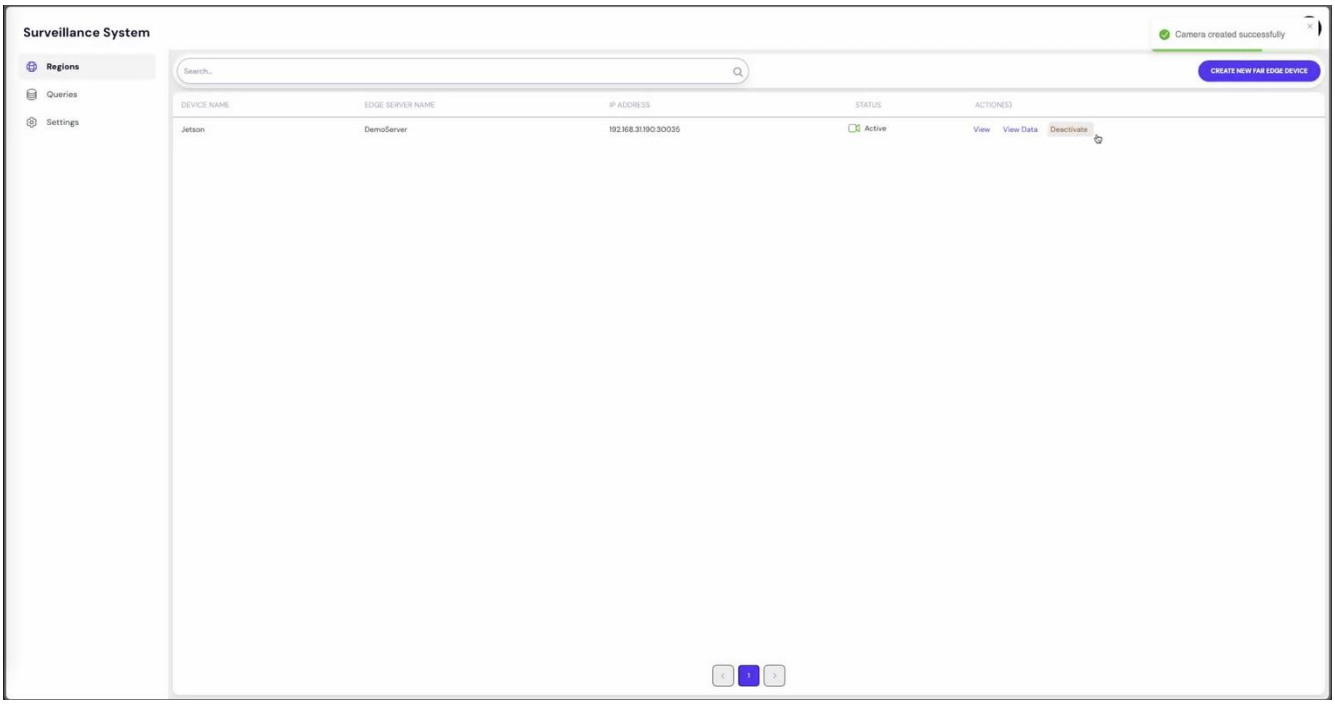

Figure 22. UC2 device management

Figure 23 highlights the real-time detection and monitoring dashboard, which includes live video feeds with bounding boxes for object detection (e.g., people, cars, signs), system resource usage (CPU, GPU, memory), and a dynamic timeline chart visualizing detection events over time.

Figure 23. Real-time detection and monitoring dashboard

Finally, Figure 24 showcases the query and archive review panel, allowing users to filter, retrieve, and browse through large volumes of historical detection events using attribute-based filters. Results are presented in a paginated grid of annotated snapshots, enabling deep forensic analysis.

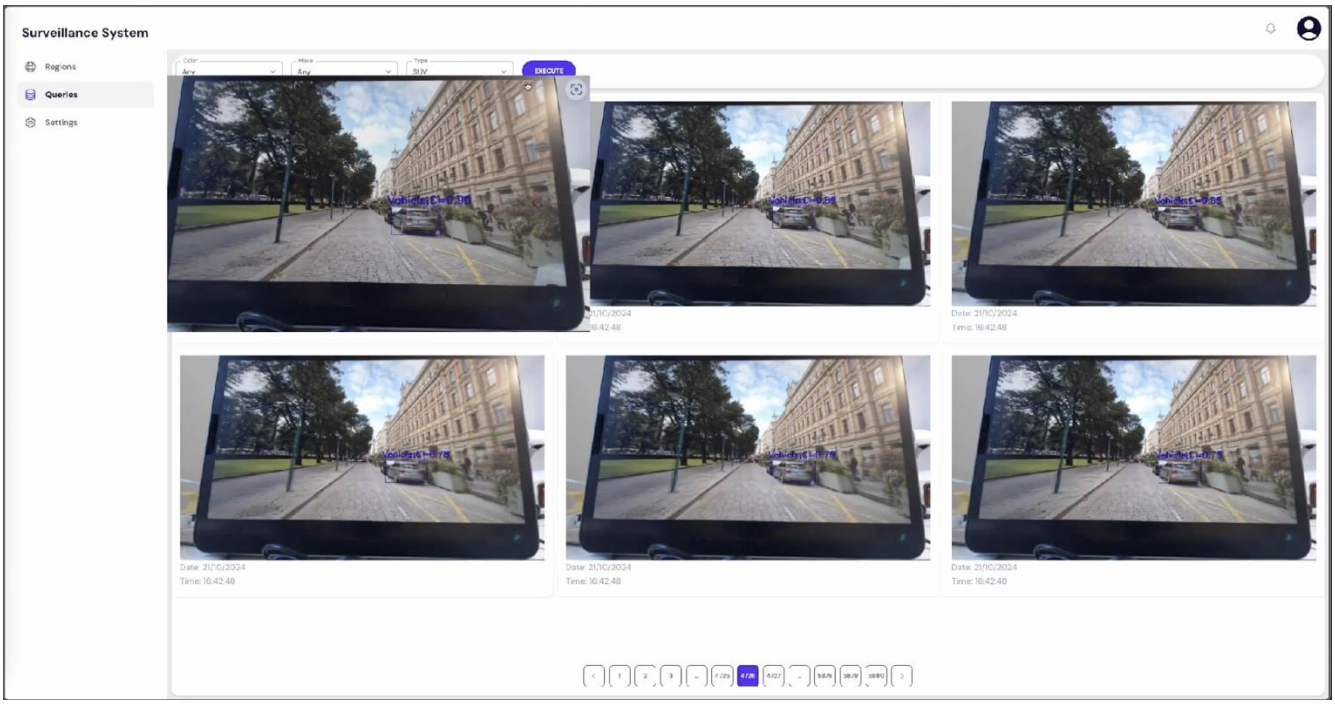

Figure 24. Query and archive review panel

4.4.2 GUI for Developer (Application Gateway)

The GUI, also known as the Application Gateway, is the primary interface through which application developers define their application deployment configurations. It offers two modes of interaction:

- Interactive Form-Based Input: Users can fill in an intuitive multi-step form that captures all necessary information, including application metadata, microservices configuration, networking preferences, and SLA constraints. This form is designed to simplify the process of describing a complex microservice-based application, even for non-expert users.
- Structured JSON Upload: Alternatively, users can upload a predefined JSON file conforming to the AppD schema. This method allows advanced users to work in a more automated way, reusing or customizing existing configurations.

Figure 25 illustrates the first step of the form, where the user provides basic application metadata such as the application name and version. It should be noted that even though we are showing the GUI interfaces only for UC2, all UCs will use this GUI as an entry point to AC³ CECCM to define, configure, and deploy their respective applications.

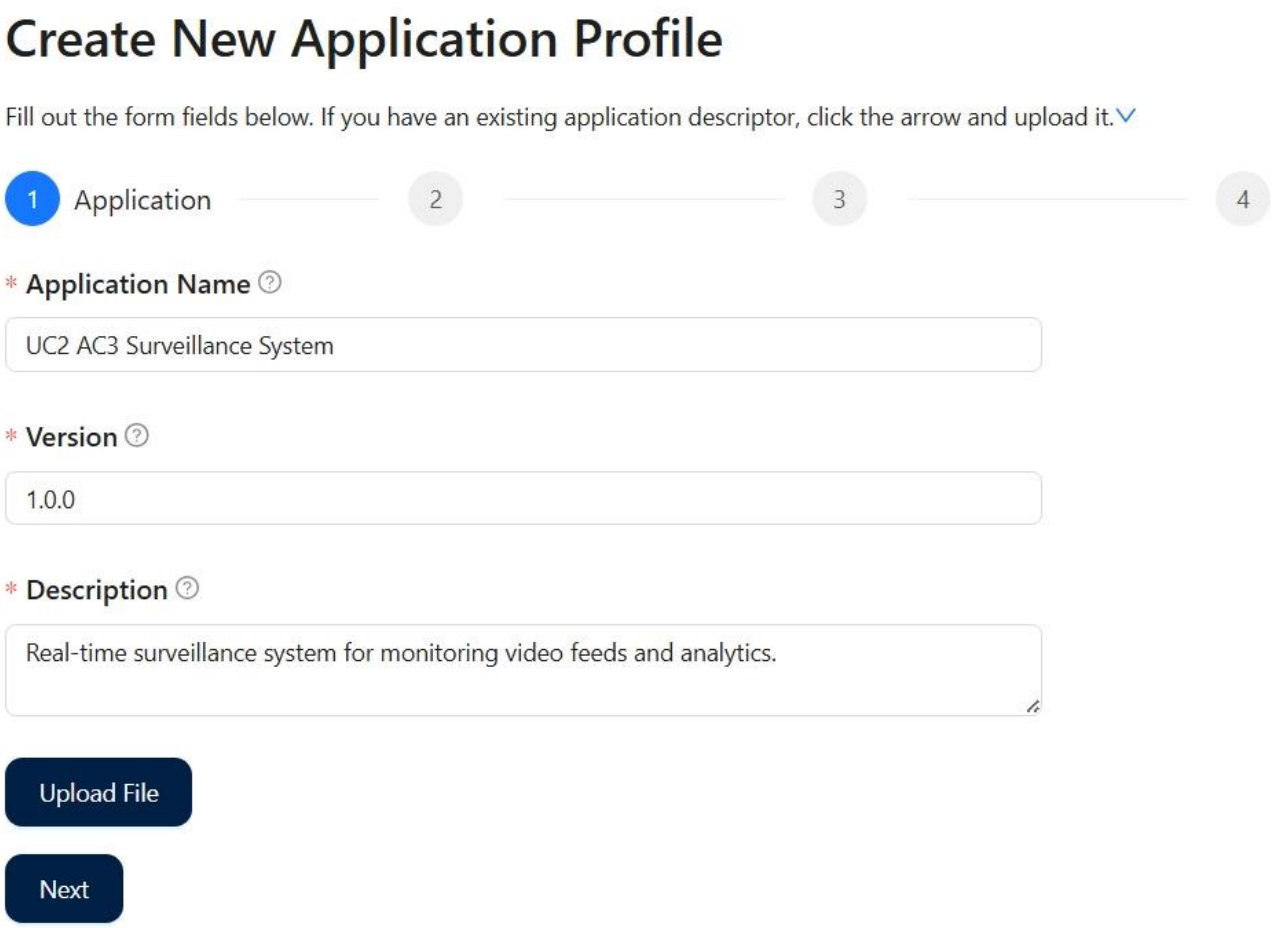

Figure 25. UC2 Application metadata input

Figure 26 shows a deeper level of interaction, where the user configures individual microservices by specifying container image, resource requirements, environment variables, and service-specific SLAs.

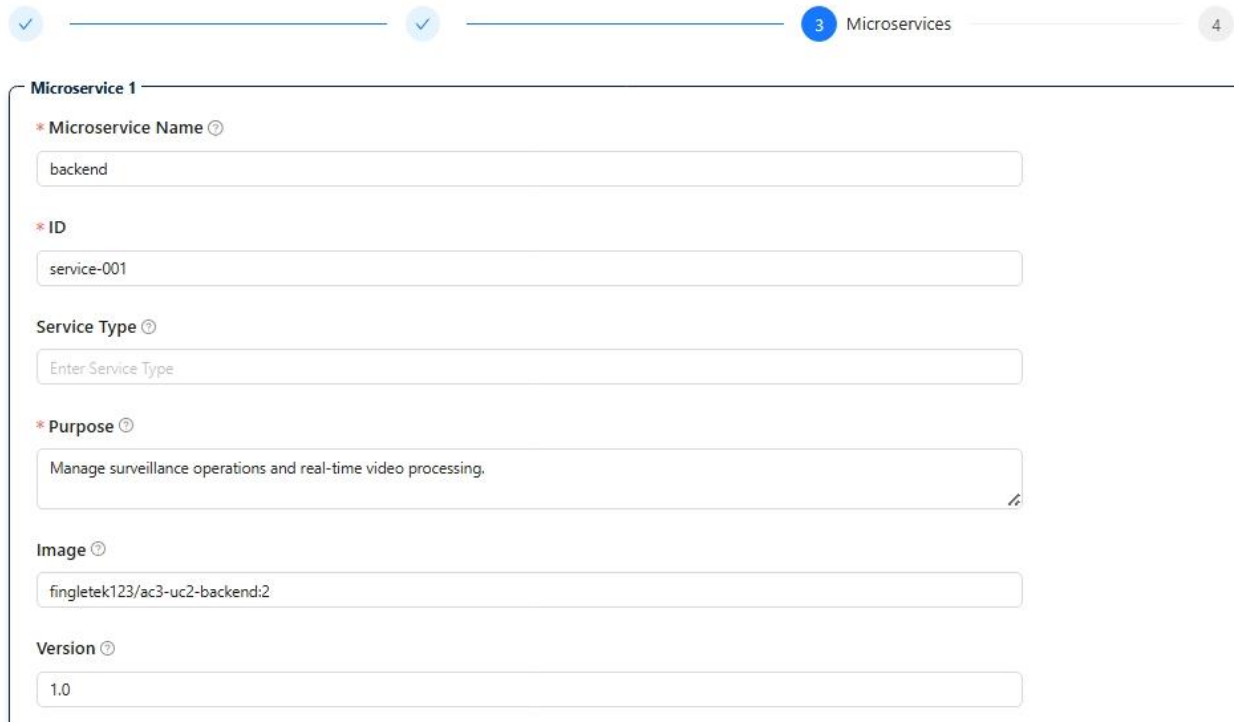

Figure 26. Microservices configuration form

Figure 27 demonstrates how users define network interconnections, describing which services communicate with each other, the protocol used, and expected network SLAs.

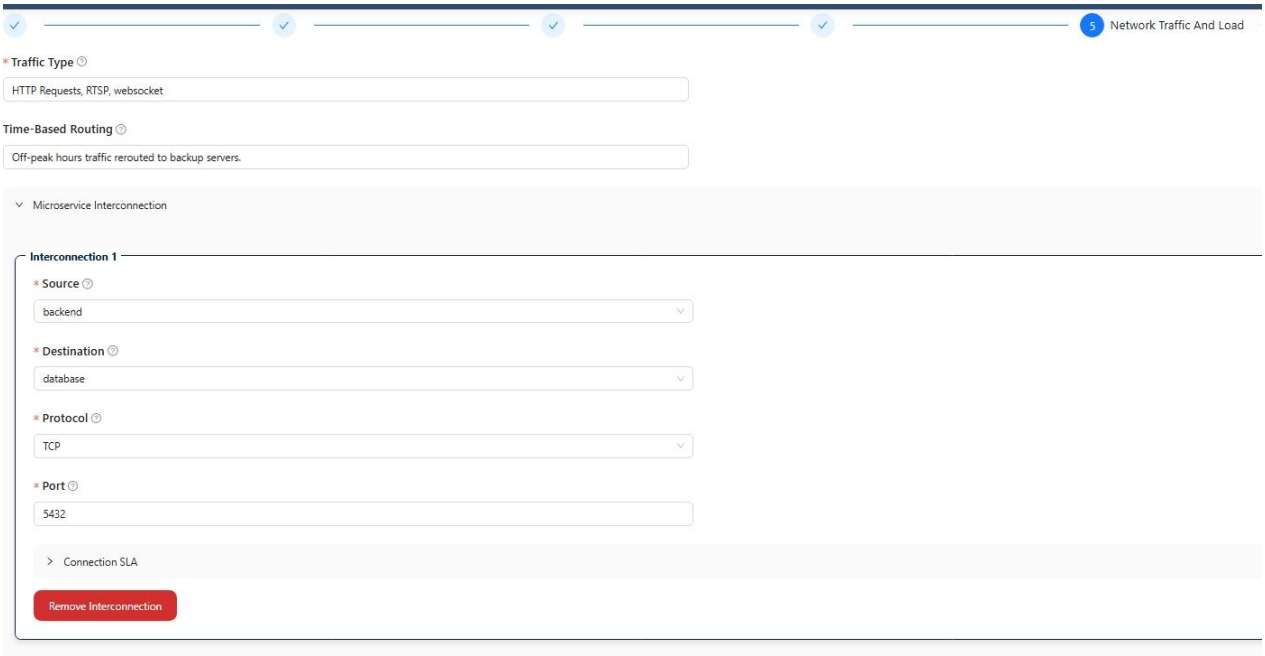

Figure 27. Networking Graph Configuration

These inputs are then passed to the OSR, which handles the backend logic for descriptor generation.

4.4.3 Ontology and Semantic Reasoner

The OSR is a central component responsible for translating high-level user inputs into a standardized, machine-readable AppD, expressed in YAML format. This descriptor captures all aspects of the application, including microservice definitions, interconnections, SLAs, and deployment constraints. Upon receiving the structured input from the GUI, the OSR performs a number of operations:

- Validation of the input data based on predefined ontologies and JSON schemas.
- Composition logic, which includes interpreting dependencies, aggregating resource requirements, and organizing networking rules.
- Retrieval of complementary services if needed from Piveau, especially for data connectors or utility services linked to the specified datasets.
- Descriptor generation, where all this information is synthesized into a deployable YAML file compatible with the Cloud-Edge Continuum infrastructure.

Following is the high-level structure of the UC2 AppD generated by the OSR. It includes application metadata, a microservices configuration section, networking information section, and SLA specifications.

```
ApplicationName: "Surveillance System"
Version: "1.0.0"

Microservices_configuration:

Global_SLA:
      ServiceAvailability: "99.9%"
      MaxLatency: "500 ms"
      MaxResponseTime: "Low"
      DataThroughput: "High"
```

Following is a concrete example of the Deepstream microservice configuration. This includes CPU/GPU requirements, image name, exposed ports, and required environment variables for AI-based video analytics.

```
  • MicroserviceName: "deepstream"

    Version: "1.1.0"
    Image: "capy8ra/ac3-uc2-ds: 28"
    ID: "deepstream"
    Dependencies:
          - "backend"
          - "database"
    ResourceRequirements:
    Cpu: "4 VCPUs"
    Memory: "16Gi"
    Storage: "N/A"
    Gpu: "NVIDIA GPU (specific model based on throughput)"
```

```
      MicroservicesSLAs:
      ServiceAvailability: "99.9%"
      MaxResponseTime: "Low" DataThroughput: "High"
      ReplicaCount: "1"
      EnvironmentVariables:
            - Name: "LOG_LEVEL"
            Value: "INFO"
            - Name: "DB_HOST"
            Value: "db"
            - Name: "DB_PORT"
            Value: "5432"
            - Name: "DB_NAME"
            Value: "ac3"
            - Name: "DB_USER"
            Value: "postgres"
            - Name: "DB_PASSWORD"
            Value: "root"
            - Name: "NO_DISPLAY"
      Value: "1"
      Protocol: "TCP/RTSP"
      InternetAccess: "false"
      GeographicalArea:
      Region: "Edge"
      LocationType: "edge"
```

Following is the networking graph extracted from the descriptor, illustrating service-to-service communication paths and the corresponding SLAs for each connection.

```
Networking_graph:

   • Source: "backend"

      Destination: "db"
      Protocol: "TCP"
      Port: "5432"
      ConnectionSLAs
            Latency: "Less than 500 ms"
            Availability: "99.9%"
            Bandwidth: "High"
            ErrorRate: "Less than 1%"


   • Source: "frontend"

      Destination: "deepstream"
      Protocol: "TCP"
      Port: "8585"
      ConnectionSLAs:
            Latency: "Less than 500 ms"
            Availability: "99.9%"
```

```
            Bandwidth: "High"
            ErrorRate: "Less than 1%"


    • Source: "frontend"

      Destination: "backend"
      Protocol: "TCP"
      Port: "8000"
      ConnectionSLAs:
            Latency: "Less than 500 ms"
            Availability:
            "99.9%"
            Bandwidth: "High"
            ErrorRate: "Less than 1%"
```

By automating the composition and validation of AppD's, the OSR enables scalable and accurate deployment across the CECC infrastructure. The generated descriptor is ultimately passed to the LCM for deployment.

4.4.4 LCM

In UC2, application LCM functionalities are implemented using the LiSO network and service orchestrator developed by EURECOM. LiSO exposes a northbound REST API that enables the orchestration of services and their constituent applications. Services are described using a Network Service Descriptor (NSD), which defines the specific configuration of microservices, their interdependencies, and the infrastructure and resource requirements of the applications.

As illustrated in Figure 28, LiSO is composed of several key components, including the orchestrator, VIM manager and the VIM. The VIM manager translates high-level orchestration requests into a series of object creation or deletion operations for the underlying VIM. While the VIM interacts directly with LMS APIs such as K8s, K3s, and OpenShift. Additionally, the VIM handles the management of container images for microservices and oversees local image registries.

4.4.4.1 Application Deployment

LiSO exposes its rest API to the OSR, meaning that any application creation or deletion is triggered by the OSR. In order to enable the communication between the two components, we introduce a translator at the OSR level, the role of the translator is to translate the AC^3 application descriptor into an NSD. The translator sends the resulting NSD as part of the application creation request to LiSO.

At the LiSO level the NSD is decomposed by microservice, and each microservice follows the onboarding and instantiation steps.

For the onboarding, LiSO constructs the microservice package, which is a descriptor of the microservice. LiSO requests the VIM Manager to onboard the microservice. The VIM manager then sends a request to the VIM to push the microservice image to the local registry used at the deployment location and create a namespace (in the case of K8s LMS) or a project (in the case of OpenShift LMS) for the application. The microservice container image can be collected in three different ways: either a container image repository available at a registry

accessible to the VIM, a Git repo with a Dockerfile so that the VIM can build the image and push it locally, or as a tar file location containing a save of the container image (the output of docker save).

At the instantiation stage, LiSO creates an instance ID for the application; an onboarded application can be instantiated multiple times; each instance has its own ID and is independent in its lifecycle management from the other instances. Once the instance ID is created, a request is then sent to the VIM Manager, which will request the VIM to create all the objects needed to run the application, including services, deployments, configmaps, volumes, etc. Once the objects are created, LiSO ensures that the application's microservices are in a running state. This check goes through the VIM Manager and the VIM. After the application is running, LiSO receives all information about the microservice, including network configuration.

Figure 28 summarizes the AppD translation from the OSR to K8s Objects.

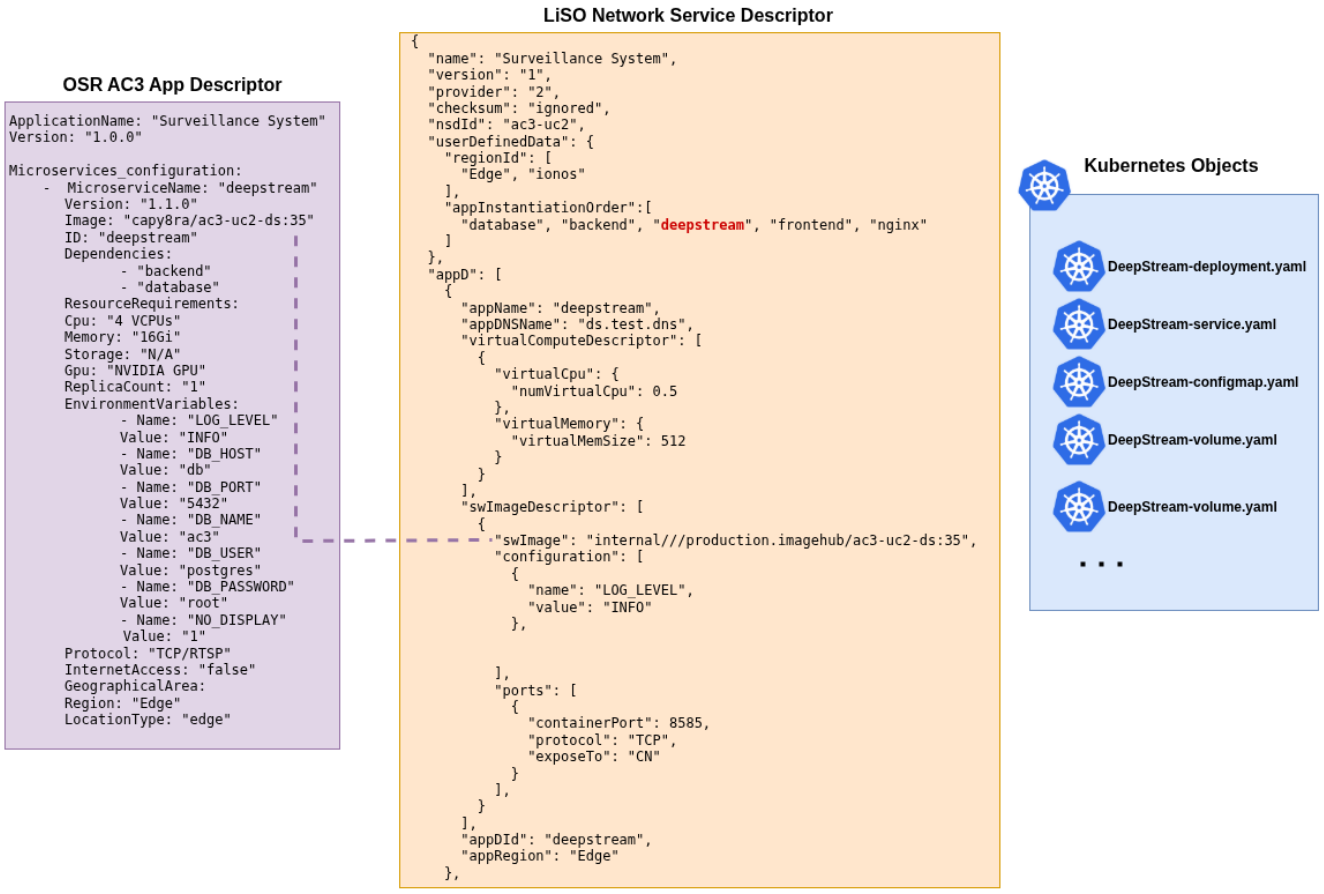

Figure 28. Translating the AC³ AppD to a LiSO Network Service Descriptor

4.4.4.2 Application Adaptation

LiSO uses a monitoring system and resource exposure, as shown in Figure 29, to continuously monitor the running application and the underlying infrastructure. In terms of application adaptation mechanisms, LiSO implements two mechanisms:

- Application Fault Detection: LiSO is capable of detecting application instance failures and can

automatically trigger the redeployment of the affected application. This mechanism relies on periodic polling of the LMS API to monitor the status of running workloads. For example, in a K8s environment, LiSO checks whether the pods are in the “Running” state. If they are not, and the issue persists beyond a predefined time interval, LiSO initiates the recreation of the application to restore its functionality.

- Detection of Application Performance Degradation: LiSO leverages monitoring data to detect suboptimal resource configurations, particularly when the resources allocated to an application are insufficient for its proper functioning. Depending on the preferences set by the application owner, LiSO can then:
    - Mark the application's configuration as insufficient. As a result, any subsequent status requests for the application will include a "resources-misconfiguration": "true" field in the response.
    - Invoke the XAI-enabled vertical resource autoscaler, developed in WP4, to dynamically adjust the application's resource allocation and correct the misconfiguration.

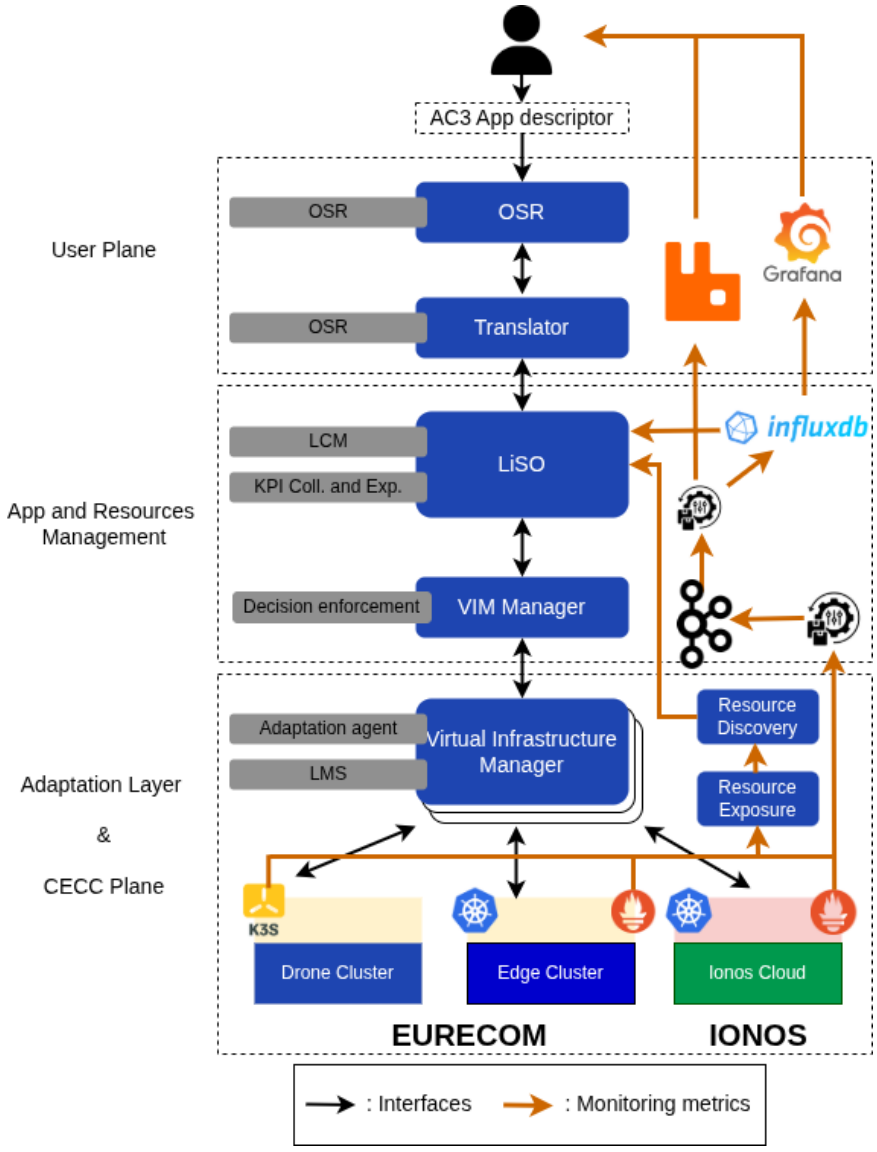

Figure 29. Detailed Architecture of LiSO and Its Mapping to AC³ Framework Components

4.4.4.3 AI LCM algorithms

LiSO's decision engine incorporates AI/ML algorithms to optimize lifecycle management, employing:

- Predictive Analytics: Historical telemetry (CPU load, network throughput, sensor trends) is fed to a prediction model that forecasts future workload and resource demand. Based on these predictions, LiSO proactively scales the application's resources.
- Microservices Migration: The main algorithm we will showcase in UC2, implemented by LiSO, is the microservice migration mechanism, which is thoroughly detailed in deliverable D3.2. The core idea is that when an edge node becomes overloaded or encounters network degradation, LiSO can perform live container migrations or reassign tasks to alternative nodes with available resources. This helps maintain consistent frame rates and low latency.

The outputs of these AI algorithms feed directly into LiSO's orchestration decisions, effectively closing the loop of continuous learning and orchestration.

4.4.5 Monitoring

4.4.5.1 Monitoring Framework Integration

In order to gain comprehensive insight into the performance of managed applications and the underlying infrastructure resources, LiSO employs a custom monitoring framework. This framework is designed to provide end-to-end visibility into application performance across all infrastructure locations where the application's microservices are deployed.

The monitoring system uses a metrics collector for each application at each deployment location. These local collectors gather performance metrics relevant to the application instance in their specific region. All keys UC2 metrics (resource usage, frame rates, sensor readings, etc.) are collected from local monitoring systems available at the cluster level, such as Prometheus. Then, the collectors enrich the collected data with contextual information such as the application ID, microservice ID, and deployment location. The structured metrics are then sent to a Kafka message broker.

From Kafka, a central collector, one per application, retrieves metrics from all deployment regions, aggregates them per service, and forwards the results to an external RabbitMQ broker. This real-time metrics stream enables application owners to access up-to-date performance data, which is particularly useful when real-time monitoring is required. In parallel, the application metrics collector stores the metrics in an InfluxDB time-series database. These stored metrics are then visualized through custom dashboards, tailored per application or per application owner, using an external Grafana server. The LCM module has access to this database and can query metrics across different applications as needed.

Note that both the metric collection streams and their exposure are isolated per application. Access to the external Grafana and RabbitMQ instances requires valid credentials, ensuring data privacy and enabling secure multi-tenancy.

4.4.5.2 Monitoring Metric

In UC2, we focus on the metrics that most accurately reflect the application's performance, including:

- **Resource Usage:** CPU and GPU utilization, memory usage, and disk I/O for each node and container.

- **Network Metrics:** Throughput and packet rates on network interfaces; latency and bandwidth usage between edge nodes.
- **Application Performance:** UC2-specific KPIs such as video frame rate (frames per second), end-to-end latency, and frame-encoding quality.
- **Sensor Values:** Relevant sensor readings (e.g., camera frame timestamps, environmental sensor values) generated by UC2 devices.
- **Service Health:** Counters for active connections, request rates, error rates, and container restart counts.

All metrics are presented on Grafana dashboards. Operators can view CPU/memory usage over time alongside the application's frame rate, making it easy to correlate load with performance. Alerts can be set on these metrics (e.g., "frame rate < 25 fps" or "CPU > 90 % for 5 min") to trigger notifications or further LiSO actions.

4.4.6 Compute LMS

K8s has been adopted as the Compute Local Management System for UC2. We have deployed a lightweight K8s distribution, K3s, at the far-edge site (on the NVIDIA Jetson/RPi hardware) and a K8s cluster in the central cloud. LiSO is connected to these clusters' APIs and is capable of launching container pods on either. The initial deployments (video analytics and data services) ran successfully on the Jetson nodes, confirming that the Compute LMS is functioning. Resource profiles of the clusters (CPU, memory, GPUs) have been entered into the CECCM so that LiSO can target the appropriate nodes. In short, the compute environment is in place and recognized by the CECCM: UC2 services can be instantiated on the K8s-managed edge and cloud as needed.

4.4.7 Network LMS

We use the SD-WAN controller and SDWAN Edges at each region as the network LMS for the UC2. The SDWAN Controller provides an API to the LCM to programmatically configure the network. The configuration includes interconnecting the microservices running in different regions: IONOS Cloud and EURECOM Edge. Further, the SDWAN Controller also allows the LCM to expose a microservice to the internet. All the configurations are enforced at the SD-WAN Edges level, where overlay networks are created for intra-cluster communication and use DNAT rules to expose application ports to the internet.

4.5 Remaining Integration

4.5.1 Deployment via GUI

The components needed to finalize the deployment process are already integrated. However, to enhance the user experience, application owners should be able to deploy their applications through a simpler interface than the command line. To achieve this, the deployment workflow needs to be integrated into the AC^3 web portal.

4.5.2 Integration of Predictive Models with AI-based LCM

As previously mentioned, this UC demonstrates the microservice migration algorithm. To support this, the AI models that trigger the migration process must be integrated with the LCM and other decision-making components responsible for selecting the new microservice location.

© AC^3 2023

4.5.3 Final Use Case Test

The UC has already been validated in a multi-cluster environment using both local and 5G networks. However, to test in a more realistic scenario, we still need to mount the object detection model on a drone.

4.6 UC2 Integration Summary

Table 4: UC2 Integration summary

Architecture component	Sub-Component	Description	Integration status
Application gateway (GUI)		Allow the application developer to define its application components and SLA.	Complete
OSR		Allow the generation of the AppD	Complete
LMS Edge		Will manage the micro-services that cannot run at the far edge (due to low computing resources) and centralised cloud for low latency or bandwidth optimization.	Complete
LMS Far Edge		Will manage the micro-services that will run on the far edge device (i.e., UAV). Video capture and object detection microservices.	Complete
LMS Cloud		Will manage and run the front-end micro-service.	Complete
LMS Networking		Will interconnect the clusters (K3s, K8s and ION Cloud).	Complete
Application and resource management	Monitoring	Monitoring the micro-services KPI	Complete
	AI-based LCM and Decision Enforcement	1. Manage the micro-services Life Cycle 2. Migration algorithm that adapts if the far-edge resource degrades or moves to the far-edge a micro-service	1. In progress 2. In progress
	Zero-touch configuration and application management (predict drones' availability)	Predict and describe infrastructure resources and implement automated corrective measures.	In Progress
	AI-Based application profile	Predicting Application Behaviour	In Progress
	AI-Based resource profile	Describe the resources of the infrastructure	In Progress

5 UC3

5.1 Use Case Description

UC3 focuses on advancing our understanding of galaxy evolution across cosmic time through the processing and analysis of large-scale 3D astronomical data cubes generated via Integral Field Spectroscopy (IFS). These data cubes are sourced from advanced instruments, including MEGARA at the 10.4 m Gran Telescopio de Canarias, MUSE at the Very Large Telescope, and MaNGA at the 2.5 m Sloan Telescope. Combining spatial and spectral information, the data cubes provide critical insights into stellar kinematics, population characteristics such as age and metallicity, and the underlying processes driving galaxy formation. A representative example involves the analysis of the nearby galaxy UGC 10205, where MEGARA data cubes are processed to map continuum emission and fit spectra using Full-Spectrum Fitting techniques. UC3 addresses significant computational challenges, including the management of vast data volumes, orchestration of complex data pipelines, and assurance of robust system availability. These issues often exceed the capabilities of traditional standalone systems, necessitating a more scalable and distributed approach.

5.1.1 Use Case Objectives

UC3 aims to leverage the AC^3 framework to establish a scalable and distributed infrastructure for processing large-scale astronomical data, while enhancing resource efficiency and supporting cutting-edge research into galaxy evolution. The specific objectives are as follows:

- Achieve a minimum 50% reduction in processing time for 5GB data cubes compared to standalone nodes by integrating cloud and edge resources, ensuring efficient handling of hundreds of terabytes of astronomical data.
- Ensure 100% system reliability to support uninterrupted research, mitigating the risk of downtime during data processing and analysis workflows.
- Utilize containerized microservices with spectral analysis tools, including pPXF, STECKMAP, and STARLIGHT, to extract key parameters such as stellar velocity, metallicity, and higher-order kinematic moments like skewness and kurtosis, advancing insights into galaxy evolution.
- Deploy the UC3 testbed via the CECCM framework across OpenShift clusters with cross-cluster networking via the AC^3 network operator, enabling dynamic task distribution, zero-touch deployment, and AI-driven resource optimization.
- Enhance resource management through parallel processing and container orchestration, minimizing memory and CPU consumption while maintaining scalability.
- Establish a benchmark for large-scale astronomical analysis, led by UCM and RHT, to empower the scientific community in accelerating discoveries and fully harnessing the capabilities of modern telescopes.

5.1.2 UC3 Stakeholders

5.1.2.1 Users / Astronomers

The primary beneficiaries and users of the deployed UC3 application are the astronomers. They will experience an improved and simplified workflow as the application allows them to more efficiently process astronomical data of varying instruments, and to effectively gather and analyse results from the various processing applications. By providing a cohesive environment for managing data acquired from a variety of astronomical instruments, the system directly supports their research objectives and accelerates their analytical processes.

5.1.2.2 Application Developers / DevOps

Application developers are responsible for building the UC3 application. This involves providing the necessary wrapping for the three specialised astronomy processing applications (Starlight, PPXF, and Steckmap). Their work focuses on developing the UC3 application to handle the efficient ingestion of astronomical data and the structured management of the processed outputs from these astronomy applications.

The DevOps team would deploy the application and would also stand to gain significant advantages through reduced operational workload. The integrated LCM system, which provides zero-touch management, automates many of the routine and complex tasks associated with deploying, scaling, and maintaining the application. The result is a substantial reduction in operational burden, allowing the DevOps team to allocate their expertise to more strategic initiatives rather than day-to-day management.

5.1.2.3 Infrastructure Provider

The Infrastructure Provider will be responsible for integrating and offering the CECCM. This involves the technical work of assembling and deploying the various components that constitute the CECCM, such as Maestro, the OSR, the Network Operator, and data connectors. They ensure the CECCM is operational and available to the Application Developers / DevOps team for deploying and managing their applications across the federated cloud-edge environment.

5.2 Use Case Architecture

To realise the vision of UC3, we have designed both an application and a testbed that enables scalable, distributed processing of large-scale astronomical datasets while leveraging the AC^3 framework's advanced orchestration and resource management capabilities. The architecture is tailored to address the computational challenges of handling vast datasets from IFS instruments such as MEGARA, MUSE, and MaNGA, ensuring efficient data ingestion, processing, and analysis across a federated cloud-edge infrastructure. The application is structured into two core components, the Orchestrator and the Processor, designed as an event-based system to support scalability and loose coupling.

For instance, the AC^3 Network Operator is implemented to establish and dynamically manage connectivity between these K8s clusters, creating a cohesive virtual application network that allows services and workloads to communicate seamlessly across physical infrastructure boundaries. Furthermore, Maestro, functioning as the LCM platform, orchestrates the deployment of the application by interpreting the AppD generated by the OSR and generating the necessary K8s resource descriptors. Maestro also adapts the application's execution through capabilities such as vertical resource autoscaling, horizontal pod autoscaling, or migration, based on insights from AI model recommendations. This comprehensive architecture not only handles the processing of vast astronomy datasets but also enhances system reliability and reduces processing times through distributed computing and AI-driven lifecycle management.

5.2.1 Use Case Application

Professional telescopes, equipped with large mirrors, are designed to collect massive amounts of light from various celestial sources. These mirrors are responsible for redirecting and focusing the incoming light towards the detectors, which can count the number of photons falling on each part of the detector. Through a digital-to-analogue conversion (DAC) system, these analogue signals are converted into digital signals, which facilitates data manipulation. Among the digital data produced by the telescopes are data cubes, which are used in our UC.

© AC^3 2023

Core to the UC application are the pre-existing astronomy analysis applications (Starlight, PPXf, Steckmap) that are used today by the astronomers at UCM. These applications are heterogeneous, with a mix of off-the-shelf and custom-built software, built on differing computing architectures with varying underlying requirements for execution. To make these applications operate in a scalable way and to be managed by the AC³ architecture, the applications have been containerized and made deployable on K8s.

The application is split into 2 core parts, the Orchestrator and the Processor. The Orchestrator is responsible for ingesting the observation data batch and splitting it into smaller chunks for parallel analysis by the Processors. The Processor then utilises the appropriate analysis software to execute the data analysis and returns the results to the Orchestrator for correlation. We have designed the application as an event-based system to support scalability and loose coupling of the components. This allows us to scale the number of Processors in a simple way, and to also distribute Processors across a federated infrastructure.

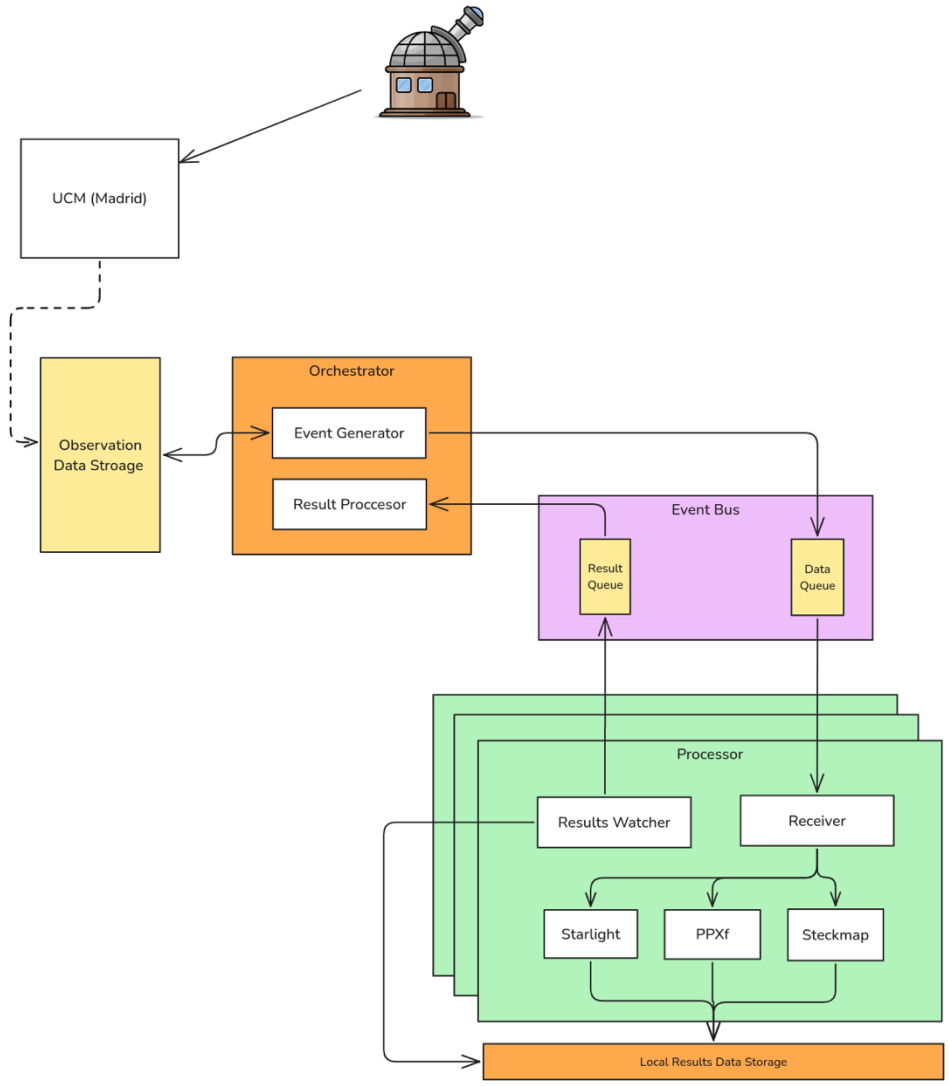

Figure 30. UC3 Application Architecture Orchestrator

As illustrated by Figure 30, the orchestrator contains 2 core components, the Event Generator and the Results Processor. The Event Generator monitors the local data storage location waiting for input data and configuration

files, detailing the data batch type as well as some analysis processing parameters. The data is then split into sub-batches of a fixed size (based on configuration options). The data and config files are then wrapped in an event and dispatched to the processing queue in the messaging bus.

In this UC, we have selected RabbitMQ as the messaging broker, using the AMQP messaging protocol. Here we have defined 2 queues, one for data processing payloads and the other for results coordination.

5.2.1.1 Processor

The processor is the heart of the UC3 application and contains several critical components. First and foremost are the Data Analysis applications themselves. Starlight, pPXF and Steckmap. These applications are effectively off-the-shelf components, which introduce the challenges of varying architectures, features, and execution models. For instance, Starlight is a binary executable built in Fortran, which executes a single batch per execution of the application. In this situation, we need to build adaptation logic to enable us to trigger the application execution remotely, in a repeatable and scalable way. We currently have full support for Starlight execution, while pPXF support is currently in development with Steckmap to follow shortly.

The Receiver component is responsible for listening for events, unpacking the payloads and routing the data batch to the appropriate processor application. Routing is based on the event type, where configuration and observation data are delivered to the appropriate data staging location and the application execution is triggered. The receiver also ensures that the Processor does not attempt to process multiple batches of different types concurrently, to mitigate performance issues and ensure the predictability of the processing time. Once processing has completed, the receiver is notified and is free to begin processing the next batch.

The Watcher monitors the output data, packages this into result events, and dispatches them to the appropriate queue for the Orchestrator to correlate.

5.2.2 UC3 Testbed – Hardware and Software

The UC Testbed, illustrated in Figure 31, is composed of applications and components distributed across multiple K8s clusters, deployed in different data centres and cloud environments. This architecture facilitates multi-site deployments, resource scaling, and advanced monitoring capabilities across platforms.

The testbed consists of three core environments: the **Astronomy Lab**, the **LCM Cluster**, and the **UC Application Clusters**. Within the Astronomy Lab we include the telescopes themselves, which are the source of the data observations. These instruments capture light from distant astronomical objects and channel it through precision optics to photon-sensitive detectors. The resulting signals, initially analogue, are digitised via DAC systems to enable further computational processing. A key output of this process is the generation of data cubes, which are central to our UC. Also, within the Astronomy Lab is a K8s-based environment running Python scripts for data manipulation and processing.

The LCM Cluster hosts the Maestro orchestration system, with the presence of a K8s cluster currently under verification. The UC Application Clusters feature a dual-cluster deployment supporting multi-cluster operations for application scalability.

In the Application clusters, the primary environment is an OpenShift-based deployment platform where applications are initially provisioned and tested. A secondary OpenShift cluster complements it, enabling multi-cluster deployment scenarios to improve resilience and scalability. These clusters work together to simulate real-world distributed systems, enabling dynamic microservice orchestration and lifecycle management.

5.2.2.1 Infrastructure Details

Compute Resources per Application Cluster:

- **CPU**: 24 vCores
- **Memory**: 64 GB RAM
- **Storage**: 200 GB SSD

Monitoring and Management Tools:

- Prometheus Operator for metrics collection and observability
- UC3 Application deployed across OpenShift clusters
- Maestro as a core orchestrator
- Advanced Cluster Management (ACM) Operator for centralized governance
- Multi-Cluster Scheduler for efficient workload placement
- Network Operator for network automation and policy enforcement

Multi-Cluster Control Plane

The multi-cluster control plane is built on a single-node OpenShift cluster that acts as the management hub. This control plane serves as the central, unified layer responsible for orchestrating, governing, and monitoring all resources and applications across the distributed testbed environments. It integrates key operators to oversee and coordinate critical functions, including application scheduling, comprehensive observability, and intricate network configuration across multi-site deployments.

By leveraging OpenShift, Prometheus, and advanced orchestration tools, this architecture supports scalable, observable, and resilient deployments tailored to microservice-based applications in a distributed, multi-cluster environment.

© AC³ 2023

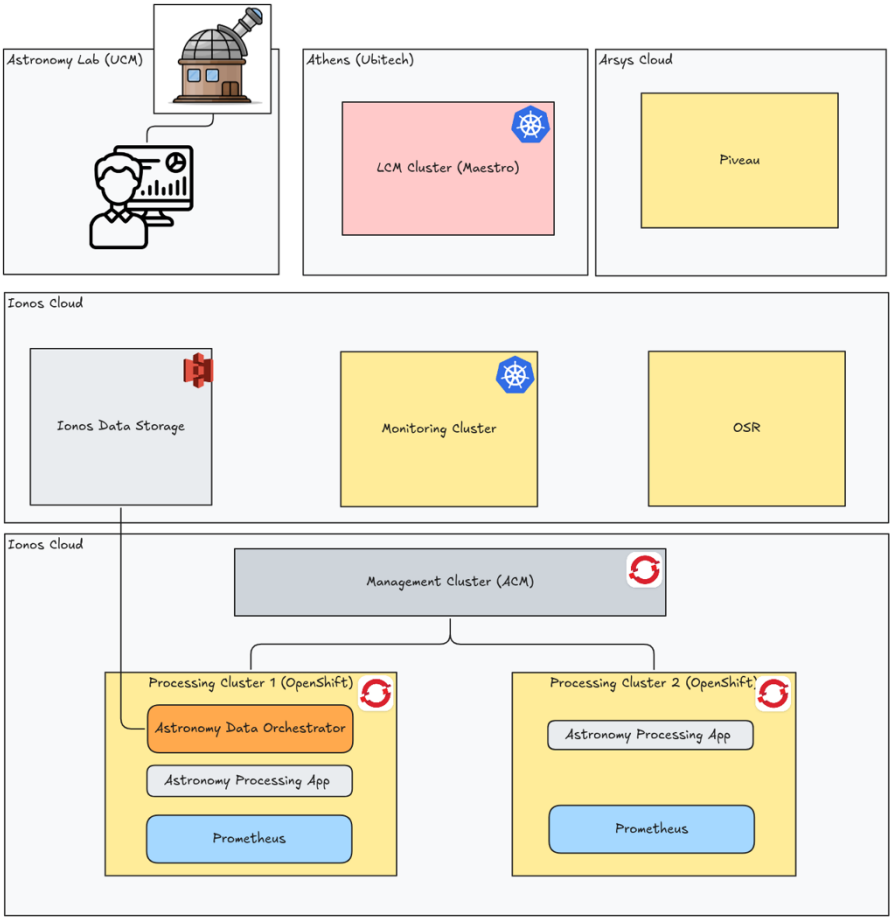

Figure 31. UC3 Infrastructure / AC³ Component Integration

5.3 Component Integration Design

As shown in Figure 32, there are several key AC³ components that we integrate with the UC3 application in order to demonstrate the collective benefit of the AC³ architecture. In terms of Data Management, the application leverages the EDC Data Connector to manage the seamless transfer of large volumes of galaxy observation data from the astronomer's lab environment to the processing application. The data source is also registered and described in the Piveau catalogue for discovery**.**

The relevant application and resource usage metrics are exposed to the monitoring framework via the Prometheus collector deployed in each application cluster. These metrics are for training the ML models as part of the application and resource profiling, as well as for inference to trigger intelligent Lifecycle Management actions.

In the role of LCM, we have used the Maestro orchestrator from UBITECH. Maestro is responsible for the deployment of the application as well as enforcing application runtime decisions based on the AI model recommendations. Specifically, Maestro interprets the AppD generated by the OSR for our data processing application and generates the K8s-based LMS resource descriptors required for deployment on the K8s LMS. Maestro will also adapt the execution of the application (e.g., executing vertical resource autoscaling, horizontal pod autoscaling, or migration) based on the output of the resource/application ML models.

© AC³ 2023

The Compute LMS utilised in this UC is K8s. Specifically, we employ a mix of OpenShift and K8s for individual compute clusters as well as an additional multi-cluster control plane in the form of ACM and a multi-cluster scheduler. Maestro will generate the multi-cluster scheduler manifests that allow the scheduling and deployment of the application across multiple clusters.

Finally, to ensure that any potential deployment or scaling of the application across multiple clusters maintains connectivity between components, we leverage the Network Programmability operator developed in WP4. The operator can be instructed to create new Layer 7 tunnels between applications deployed on different clusters

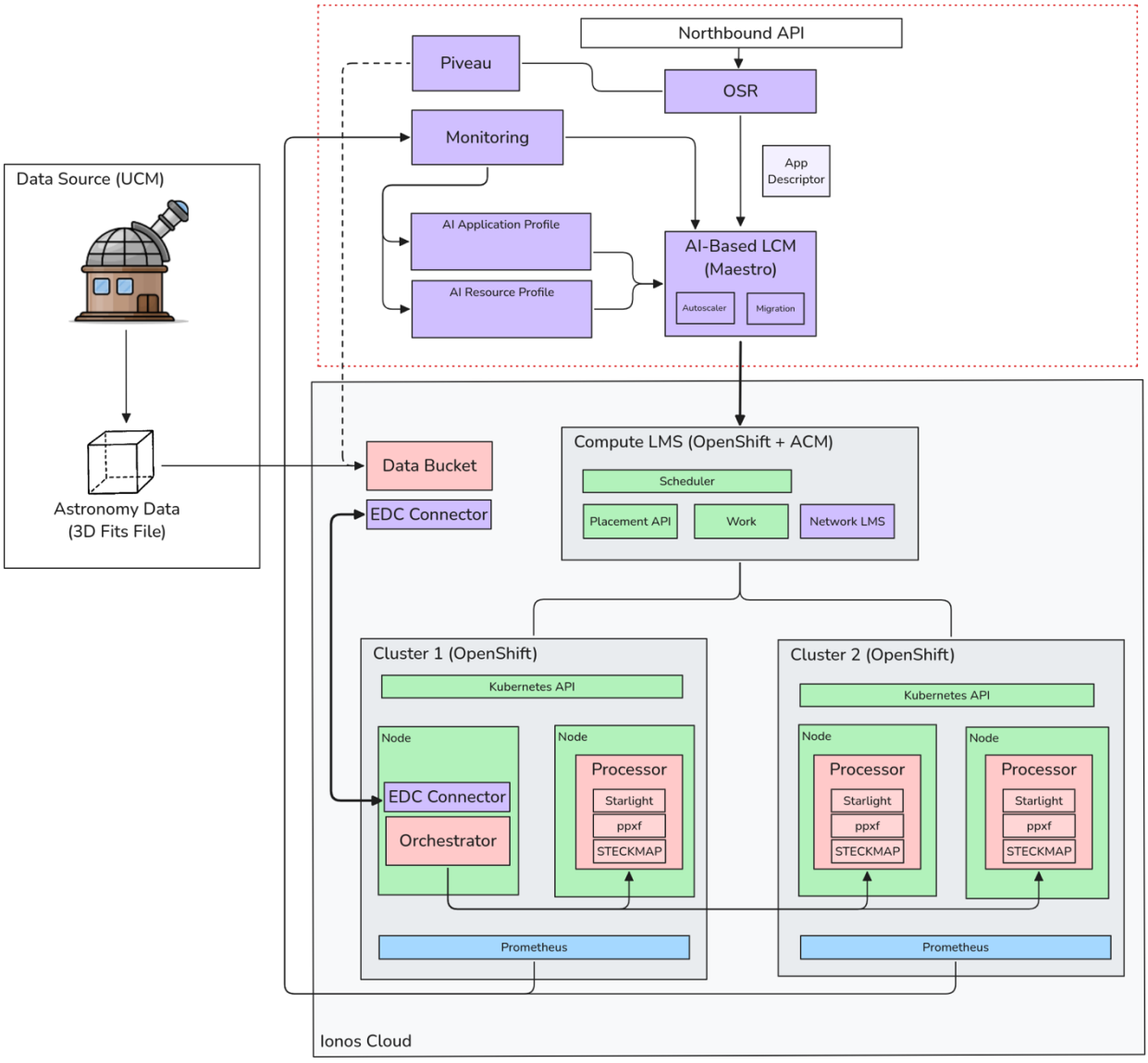

Figure 32. UC3 / AC³ Component Integration – How UC3 leverages the components developed in AC³

The UC3 Application Onboarding diagram (see Figure 33) outlines the process for deploying applications within the system. It begins with the user specifying the application details in a JSON format via the GUI, part of the Application Gateway. The GUI forwards service details to the Service Catalogue, which can also be leveraged by the user to select pre-existing services to use in their application. The GUI also sends the application details to

the App Gateway, which forwards them to the OSR to be translated into a machine-readable format called the Resource Description Framework (RDF) which is used to represent and exchange graph data. The application details are then sent to the application gateway and consolidated into the final AppD. The AppD is then included in the service catalogue and sent to the LCM, which coordinates the deployment of the application on the testbeds.

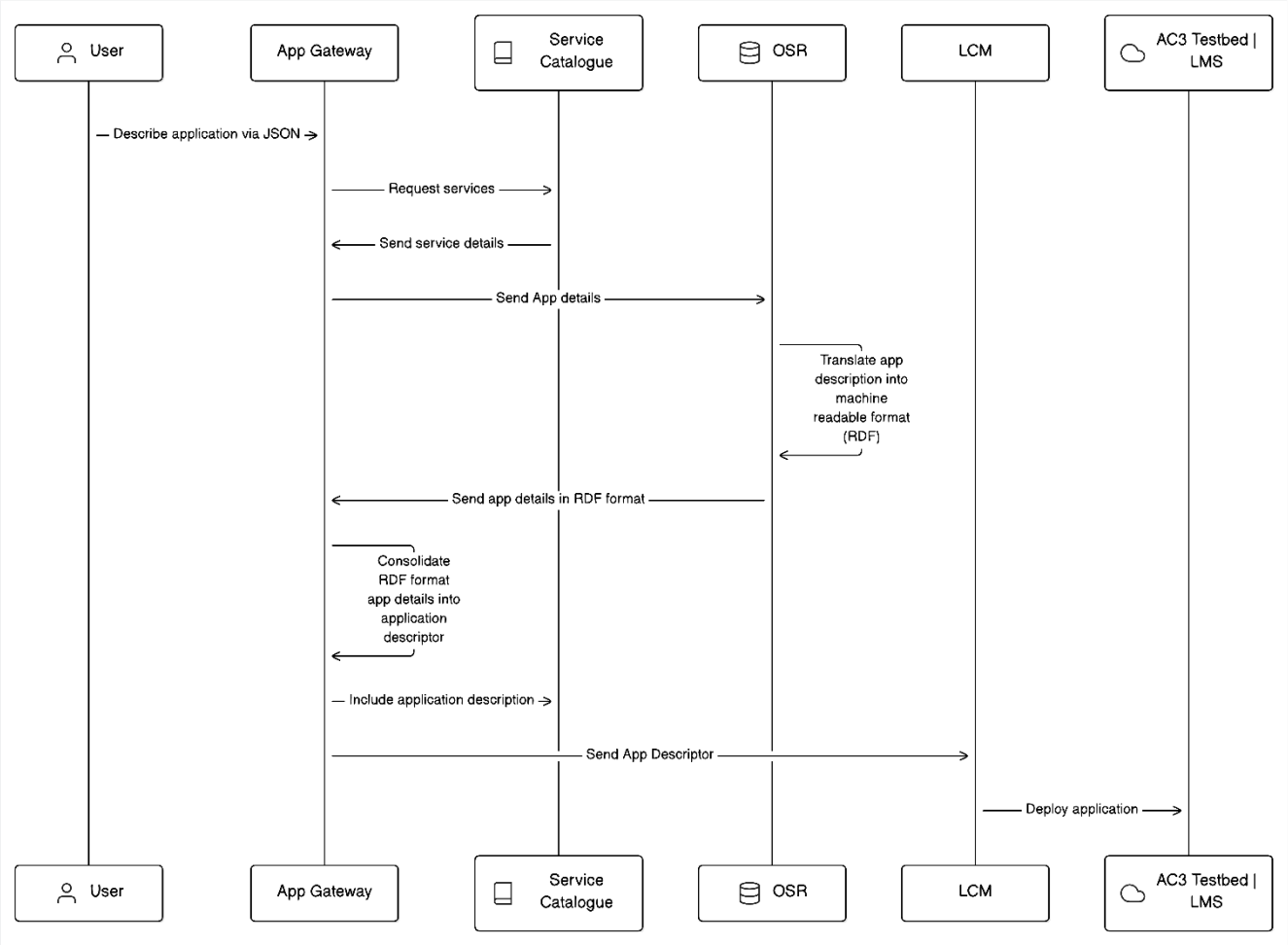

Figure 33. UC3 Application Onboarding Pipeline

The UC3 Processing Workflow diagram (see Figure 34) illustrates the data processing pipeline for UC3. Astronomy data from the source is first saved to the UCM Ionos Simple Storage Service (S3) storage bucket. The UCM Provider Connector transfers this data to the Red Hat Consumer Connector, which stores it in the Red Hat S3 storage bucket. The UC3 Orchestrator watches for new data and retrieves results as needed, coordinating with RabbitMQ to send batch data to the UC3 Processors for computation. Processed results are then stored back in Red Hat S3 and transferred to UCM Ionos S3 for final storage. This sequence ensures efficient data handling and processing, leveraging the work/event dispatching model central to our architecture.

© AC³ 2023

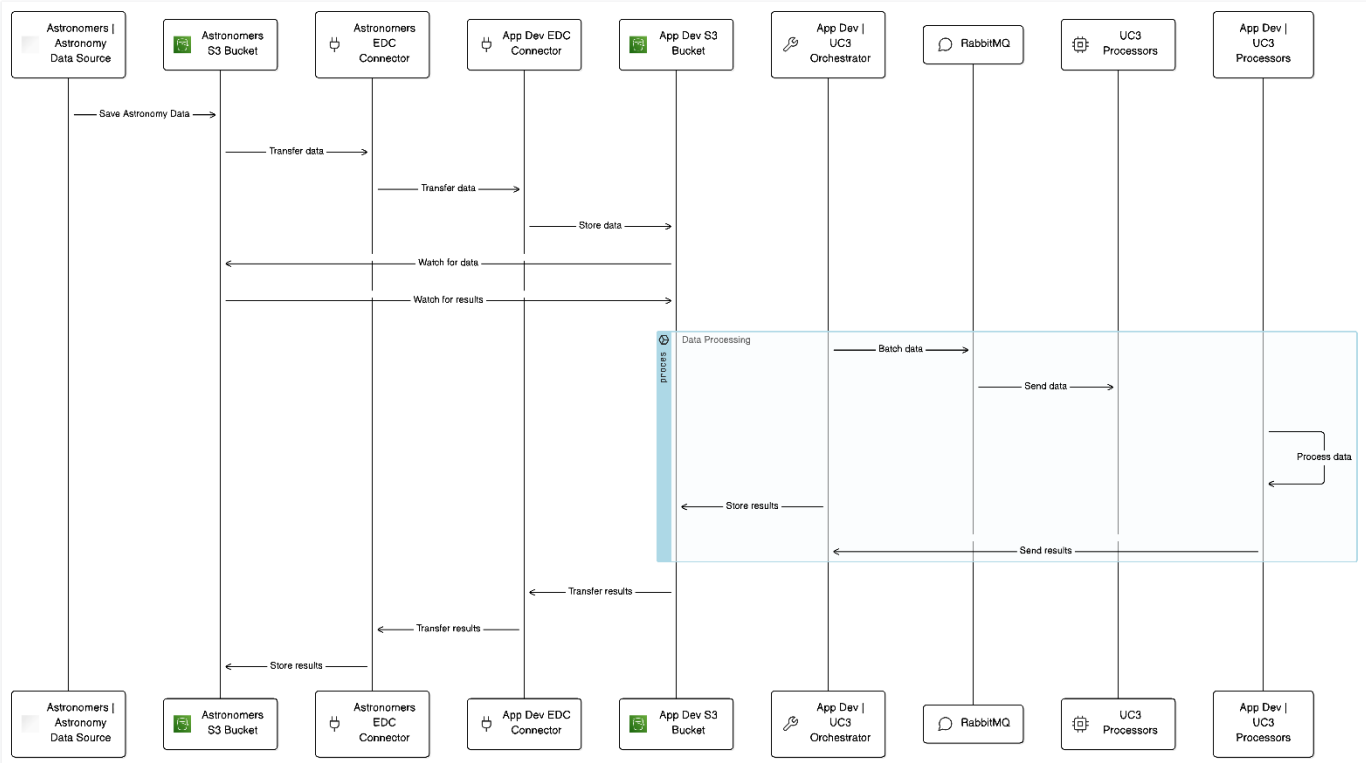

Figure 34. UC3 Data Processing Pipeline

The UC3 Monitoring diagram (see Figure 35) details the runtime monitoring and resource management mechanisms in place. The UC3 Orchestrator, RabbitMQ, and processors are the core components responsible for preparing and processing the data. As such, this process must be monitored to ensure KPIs are met and resource utilisation is efficient. Prometheus monitors queue length to gauge workload demands, while the UC3 Processors are monitored for CPU and RAM usage. These metrics are collected and fed into the Monitoring Framework, which informs the App and Resource Profiles. The LCM leverages this data to make informed decisions, triggering scaling or migration actions to maintain system performance and resource efficiency. This monitoring process ensures that the system can dynamically adapt to varying operational conditions, supporting the overall stability of the UC.

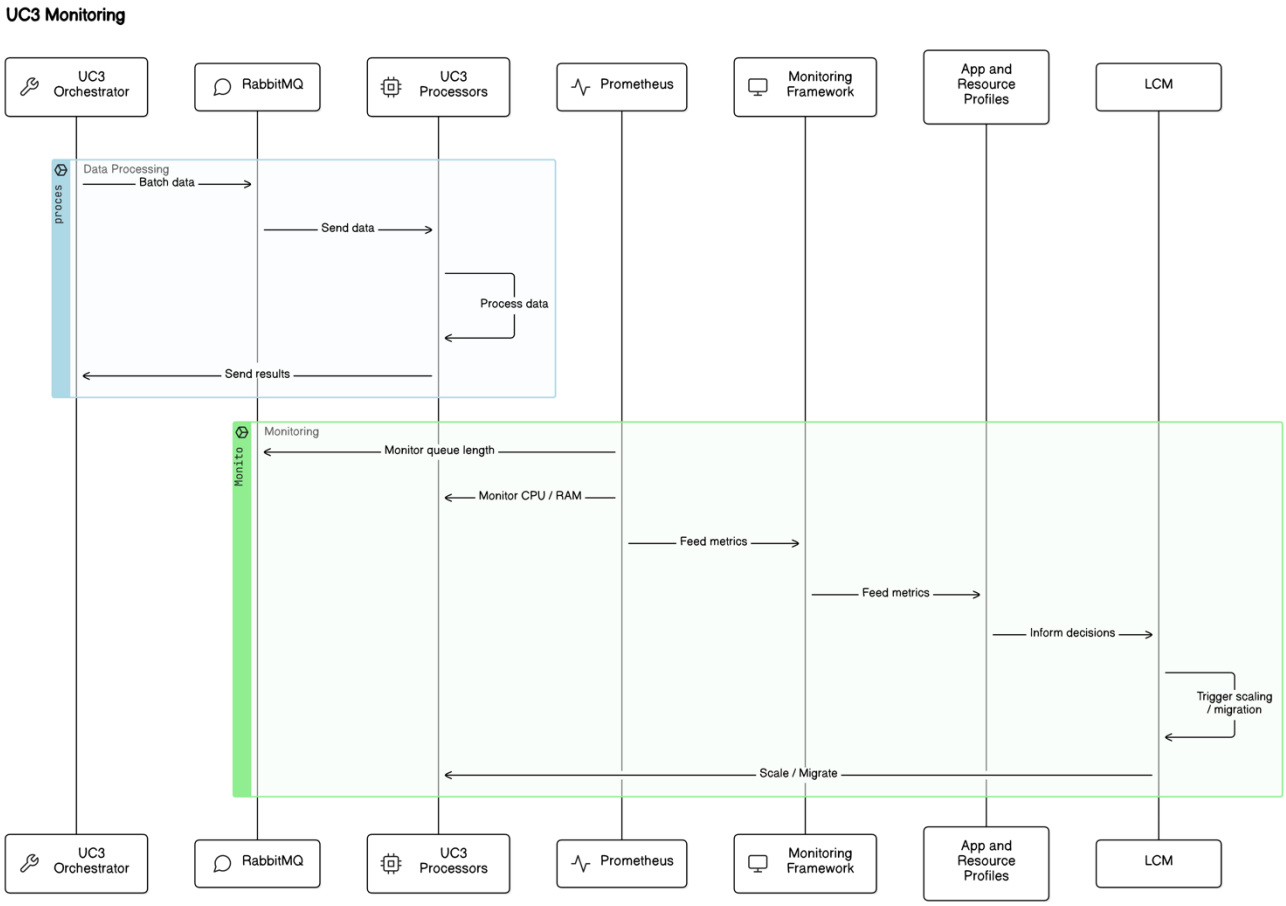

Figure 35. UC3 Monitoring Sequence Diagram

5.4 Component Integration Status

5.4.1 Application Interface

The UC3 application GUI shown in Figure 36 provides users with a structured view of the astronomy data batches stored in the S3 buckets, facilitating the management and monitoring of the batch processing workflow executed by the UC. The bucket directory structure is as follows:

- **Top-Level Directory** (**/batch_name_date**): Displayed in the application UI as a collapsible folder, labelled with the batch identifier and date (e.g., batch_001_2025-04-01). This folder groups all data related to a specific batch, allowing users to expand it and view its contents.
- **Status file** (**/batch_name_date/status**): The status file details the list of files, the batch's state (waiting, currently processing, finished), along with the start time, completion time, and duration.
- **Config Subdirectory** (**/batch_name_date/config/**): The config subdirectory contains a configuration file which provides parameters used by the processing applications to correctly process each batch, these can be viewed or downloaded by users to verify the batch parameters.
- **Input Data Subdirectory** (**/batch_name_date/input_data/**): The input data subdirectory lists the astronomy data files to be processed. Users can view these files in the UI or download them to inspect the raw data before processing begins.

- **Results Data Subdirectory** (**/batch_name_date/results_data/**): The results subdirectory contains the processed data generated by the astronomy processing applications such as Starlight or PPXF.

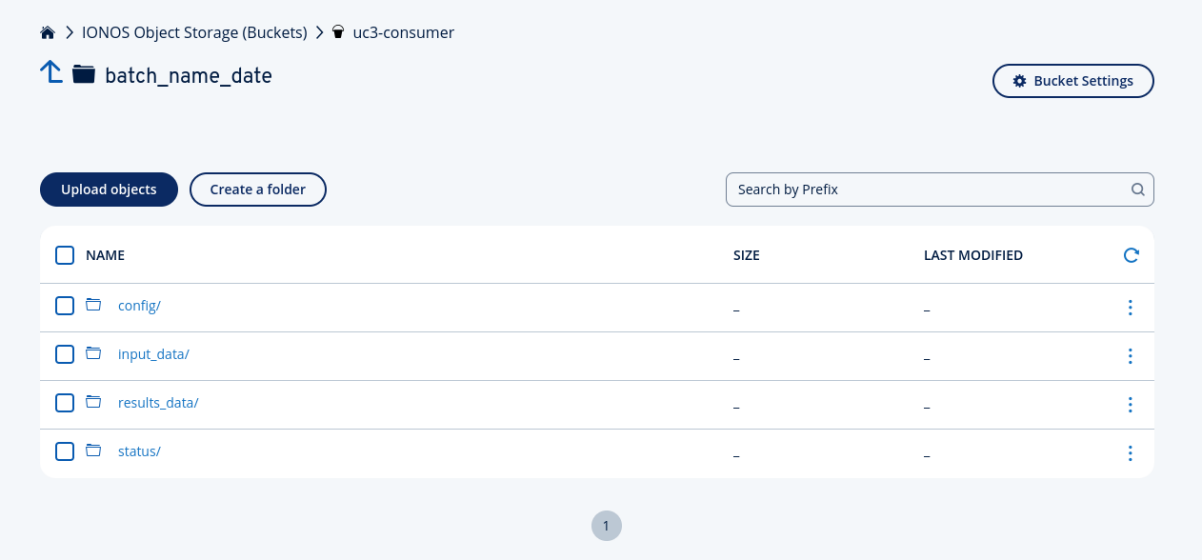

Figure 36. S3 Bucket File Structure

5.4.2 Data Management and Connectors

5.4.2.1 Piveau catalogue

The Piveau Catalogue, comprising both its data and service components, supports the management and accessibility of astronomy datasets for the processing applications. The data catalogue is designed to store key metadata for these datasets, including attributes such as dataset name, description, ownership details, and applicable usage licenses, alongside unique asset IDs. As part of the ongoing development, a sample astronomy dataset has been successfully integrated into the data catalogue. However, connector endpoints, which are essential for enabling data transfer, are not yet included within the data catalogue, and the corresponding connector has not been integrated into the service catalogue. At this stage, the system is not yet operational for users. Once fully implemented, the catalogue will allow users to negotiate access agreements directly with data providers, utilising their own consumer-side data connectors to secure permissions for downloading and using datasets in accordance with the specified policies and contract.

To realise this functionality, subsequent development phases will prioritise the integration of connectors into the service catalogue. This process will involve embedding critical technical metadata, including required environment variables, exposed ports, connector names, descriptions, minimum computing resource requirements, and references to container images. Upon completion, this integration will empower application developers to select, configure, and deploy the appropriate connectors necessary for seamless data transfer to or from their applications. These enhancements will ensure robust interoperability between the Piveau Catalogue and UC3 applications, supporting the broader objective of establishing a scalable, efficient, and user-centric data management framework within the research ecosystem. The system remains under active development and is not yet deployed for end-user utilisation.

© AC³ 2023

5.4.2.2 Data type and examples (.fits)

The data used in this UC is produced by IFS instruments, which generate astronomical data in the form of three-dimensional data cubes. These cubes combine spatial and spectral information, capturing information from contiguous regions of the sky. Each data cube consists of two spatial dimensions, representing the x and y coordinates on the sky, and one spectral dimension, representing the wavelength. This format allows astronomers to analyse the spatial distribution of various spectral features across an observed field, providing detailed insights into the physical and chemical properties of celestial objects. By examining the light from different regions within the data cube, researchers can study the composition, kinematics, and evolution of galaxies, stars, and other astronomical phenomena with high precision. Specifically, we will focus on observations of galaxies.

These data cubes are stored in the Flexible Image Transport System (FITS) format, the standard data format used in astronomy. FITS files are designed to store, transmit, and manipulate scientific data, particularly images and spectra. Each FITS file consists of a primary header and data unit (HDU), which contains metadata describing the data, followed by one or more extensions that store the actual data. This format supports multi-dimensional data arrays, making it ideal for storing the complex three-dimensional data cubes produced by integral field spectroscopy. FITS files are highly versatile and can include additional information such as calibration data, observational parameters, and processing history, ensuring that all necessary context is preserved for accurate data analysis. By using the FITS format, we ensure compatibility with a wide range of astronomical software and facilitate efficient data sharing and collaboration within the research community.

Our UC leverages data from three different IFS instruments: MEGARA [7], MaNGA [8], and MUSE [9].

- **MEGARA**: MEGARA is an optical Integral-Field Unit and Multi-Object Spectrograph designed for the Gran Telescopio Canarias. It provides high-resolution spectroscopy data, which is crucial for detailed studies of stellar populations and kinematics within galaxies. The data cubes from MEGARA typically have dimensions of 40 × 43 × 4300, resulting in 1720 spectra per cube. Each MEGARA data cube weights approximately 62 MB, with the total volume of MEGARA data amounting to 7.1 GB

- **MaNGA**: Part of the Sloan Digital Sky Survey, MaNGA (Mapping Nearby Galaxies at APO) obtains spectra across the entire face of target galaxies using custom-designed fiber bundles. MaNGA's goal is to understand the life history of present-day galaxies by providing two-dimensional maps of various stellar and ionized gas properties. The data cubes from MaNGA typically have dimensions of 74 × 74 × 6732, yielding 5476 spectra per cube. However, these dimensions can vary for different galaxy observations, and the sizes provided are indicative. The minimum and maximum sizes of the full data-cubes are 133 MB and 757 MB, respectively, with a total volume of approximately 4.23 TB.

- **MUSE**: The Multi Unit Spectroscopic Explorer (MUSE) is a panoramic integral-field spectrograph operating at the Very Large Telescope (VLT) of the European Southern Observatory. MUSE combines a wide field of view with improved spatial resolution provided by adaptive optics, making it a powerful tool for discovering and studying faint and distant astronomical objects. The data cubes from MUSE typically have dimensions of 319 × 320 × 3721, equating to 102080 spectra per cube. Similar to MaNGA, the dimensions of MUSE data cubes can vary depending on the specific observations of different galaxies, and the sizes mentioned are approximate. The minimum and maximum sizes of the full data-cubes are

2.3 GB and 28 GB in .gzip format, and 2.9 GB and 32.6 GB in FITS format, with a total volume of approximately 3.2 TB for the files in .gzip format.

The data from these instruments vary in size and dimensions due to the different fields of view each instrument covers. This variability reflects the diverse observational capabilities and scientific goals of each instrument, highlighting the need to adapt existing astronomical software tools to be user-friendly for data from different instruments and telescopes. Example .fits data files are shown in Figure 37 and Figure 38.

```
name                                  DATE-OBS= '2018-09-19T00:06:20.18' / Start
of observation                         BUNIT   = 'Jy      '             / Unit of
pixel value                              DETECTOR= 'CCD2231-84-0-E74'   /
Detector Model                                   EXPTIME =                720. /
[s] Exposure time                                DARKTIME=             720.841 /
[s] Exposure time                                READMODE= 'NORMAL  '           /
Read Mode                                        RSPEED  = 'SLOW    '           /
Readout Speed                                    GAIN1   =                1.73 /
[e- / ADU] Gain Top Channel                      GAIN2   =                 1.6 /
Gain Bottom Channel                              RDNOISE1=                 3.4 /
RON Top Channel (e-)                             RDNOISE2=                 3.4 /
RON Bottom Channel (e-)                          NUM_INDX=                  1. /
Image index in sequence                          NUM_IMAG=                  5. /
Number of images in sequence                     IPA     =             196.146 /
Instrument position (degrees)                    AIRMASS =      1.2045240089715 /
AIRMASS                                          AIRMASS1=      1.2045240089715 /
AIRMASS at begining of observation               AIRMASS2=      1.2045240089715 /
AIRMASS at end of observation                    AMSTART =      1.2045240089715 /
AIRMASS at begining of observation               AMEND   =      1.2045240089715 /
AIRMASS at end of observation                    RA      = '01:22:39.158'       /
Telescope right ascension (h:m:s)                DEC     = '34:26:11.615'       /
Telescope declination (d:m:s)                    RADEG   =    20.66315630344987 /
Telescope right ascension (degrees)              DECDEG  =     34.4365596465142 /
Telescope declination (degrees)                  AZIMUTH =     71.3031431896873 /
Tel. azimuth at start time (deg)                 ELEVAT  =     56.0845807146726 /
Elevation at start of observation (deg)          ROTANG  =     104.855811111111 /
Rotator FCCS_F position angle (deg)              ASGRA   = '01:22:40.138'       /
ASG right ascension (h:m:s)                      ASGDEC  = '34:30:09.615'       /
ASG declination (d:m:s)                          M2UX    =     2.62703990936279 /
GTC secondary mirror UX position                 M2UY    =     3.78100991249084 /
GTC secondary mirror UY position                 M2UZ    =    -2.03623008728027 /
GTC secondary mirror UZ position                 M2RX    =  0.00097221601754427 /
GTC secondary mirror RX position                 M2RY    = -0.000684012018609792
```

Figure 37. Example .fits data

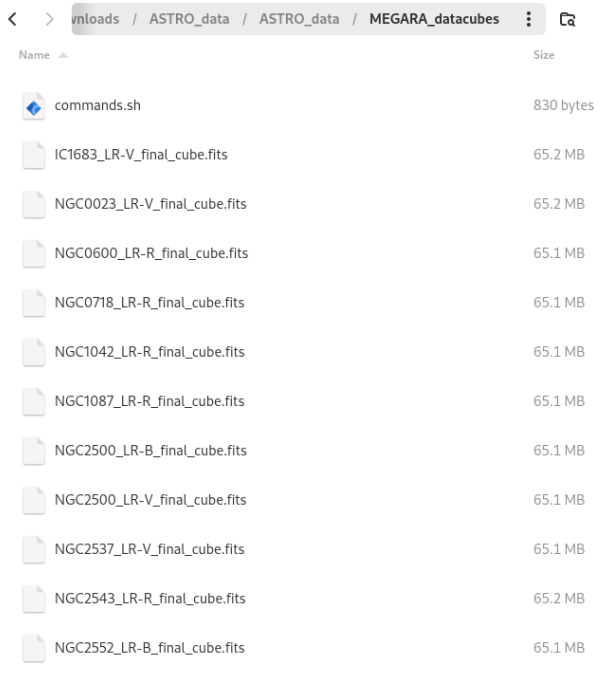

Figure 38. MEGARA Data-cubes as .fits file batches

5.4.2.3 EDC S3 Extended Connectors Developed by Ionos

To enable seamless data transfer between the UCM and the processing applications deployed on the UC3 infrastructure, UC3 leverages the EDC S3 Data Connector extensions developed by IONOS. These EDC connector extensions facilitate bi-directional data exchange between S3 buckets, ensuring efficient and secure data flows. They incorporate data governance through predefined policies and contracts, which must be negotiated prior to initiating data transfers, thereby ensuring compliance and control over data usage.

The EDC S3 extensions are designed to comply with industry-standard protocols for secure cloud storage, enhancing interoperability with diverse S3-compatible systems, and enabling integration with multiple project testbeds or external infrastructures.

Within the UC3 workflow, these connectors serve as a vital link between data ingestion and processing. They transfer astronomy data from the UCM S3 bucket to the RHT S3 bucket for batching by the Orchestrator, while their bi-directional capability supports returning processed outputs to UCM.

S3 Buckets to Store Data

S3 buckets have been created on the Ionos infrastructure by UCM and RHT to facilitate the storage, retrieval, and transferring of data by the connectors. Each bucket must be configured to allow the other parties connector read and write access by adding the Ionos user ID as a grantee in the bucket's Access Control List (ACL). There are several keys and endpoints provided by Ionos which must be included in the connectors configuration file or passed in as environment variables to ensure proper communication and access to the S3 bucket.

Deployment

The deployment of the EDC S3 Extended Connectors for UC3 is executed through a series of technical steps to ensure seamless integration with the OpenShift clusters on IONOS and Arsys infrastructures. Both the UCM provider and RHT consumer connectors are deployed as containerized applications, leveraging Docker images and OpenShift's K8s orchestration for scalability and reliability.

Each connector is deployed on an OpenShift cluster hosted on either the IONOS or Arsys testbeds. They use custom built Docker images which are pulled from a quay.io repository and deployed using K8s deployment manifests, while the connector's configuration is mounted as ConfigMaps containing the respective config.properties files, which include the IONOS S3 credentials:

```
edc.ionos.access.key=
edc.ionos.secret.key=
edc.ionos.token=
edc.ionos.endpoint.region=eu-central-2
```

These are injected as environment variables via the JAVA_TOOL_OPTIONS flag to ensure the connectors can access the corresponding S3 buckets.

```
-Dedc.fs.config=/app/resources/config.properties
```

A Hashicorp Vault server is deployed on the OpenShift testbeds alongside the connectors to manage temporary keys generated by the IonosS3Provisioner during the data transfer process and is deployed by both RHT and UCM with the details of each deployment being used in the corresponding connectors configuration:

```
edc.vault.hashicorp.url=http://vault:8200
edc.vault.hashicorp.token=myroot
edc.vault.hashicorp.timeout.seconds=30
```

Workflow

The data transfer process begins with preparing the necessary JSON payloads to define the asset, policy, contract, and transfer request. These payloads are used to interact with the RHT consumer connector management API, which communicates with the UCM provider connector to facilitate the data transfer. The RHT consumer connector exposes its management API, which is accessible for issuing curl commands. The UCM provider connector responds to these requests to provide the data from the UCM bucket, and vice versa for the results transfer.

First, an asset must be defined on the UCM provider side to represent the astronomy data in the uc3-provider S3 bucket. A JSON payload, "asset.json", is created to register this asset before being sent to the UCM provider's management API using a curl command:

© AC³ 2023

```
{
       "@context": {"edc": "https://w3id.org/edc/v0.0.1/ns/"},
       "@id": "asset-1",
       "properties": {
              "name": "astronomy-data",
              "type": "IonosS3",
              "bucketName": "uc3-provider",
              "keyName": "astronomy-data.txt"
       }
}
```

```
curl -X POST "http://provider:8282/management/v3/assets" \
-H "Content-Type: application/json" \
-H "X-API-Key: password" \
-d @asset.json
```

A policy must then be defined on the UCM provider side to govern access to the astronomy data. A JSON payload, “policy.json”, is created to define a policy that permits usage of the asset:

```
{
  "@context": {
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  },
  "@id": "policy-1",
  "policy": {
    "@type": "odrl:Set",
    "odrl:assigner": {
      "@id": "provider"
    },
    "odrl:target": {
      "@id": "asset-1"
    },
    "odrl:permission": [],
    "odrl:prohibition": [],
    "odrl:obligation": []
  }
}
```

```
curl -X POST "http://provider:8282/management/v3/policydefinitions" \
-H "Content-Type: application/json" \
-H "X-API-Key: password" \
-d @policy.json
```

Next, a contract definition is created to govern the data transfer. A JSON payload, “contract.json”, is prepared to define the contract terms, linking the asset to a policy that permits access:

```
{
   "@context": {
```

```
      "edc": "https://w3id.org/edc/v0.0.1/ns/"
    },
    "@id": "contract-1",
    "accessPolicyId": "policy-1",
    "contractPolicyId": "policy-1"

 }
```

```
curl -X POST "http://provider:8282/management/v3/contractdefinitions" \
-H "Content-Type: application/json" \
-H "X-API-Key: password" \
-d @contract.json
```

The RHT consumer then initiates a contract negotiation with the UCM provider. A JSON payload, is created and used via curl to request access to the asset:

```
{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  },
  "@type": "NegotiationInitiateRequestDto",
  "counterPartyAddress": "http://provider:8282/protocol",
  "protocol": "dataspace-protocol-http",
  "offer": {
    "offerId":
"Y29udHJhY3QtMQ==:YXNzZXQtMQ==:M2NlZTQ2OTYtNzc2Yi00Y2E0LWJlMWItM2NhMWU3OTg5NDAz"
  },
  "policy": {
    "@id":
"Y29udHJhY3QtMQ==:YXNzZXQtMQ==:M2NlZTQ2OTYtNzc2Yi00Y2E0LWJlMWItM2NhMWU3OTg5NDAz",
    "@type": "odrl:Offer",
    "odrl:assigner": {"@id": "provider"},
    "odrl:target": {"@id": "asset-1"},
    "odrl:permission": [],
    "odrl:prohibition": [],
    "odrl:obligation": []
  }
}
```

```
curl -X POST "http://consumer:9192/management/v3/contractnegotiations" \
-H "Content-Type: application/json" \
-H "X-API-Key: password" \
-d @negotiation.json
```

Once the negotiation is finalized (status FINALIZED), the response provides a contract agreement ID which is used to initiate the data transfer. A JSON payload, “transfer.json”, is prepared to define the transfer request. The transfer is initiated by sending this payload to the RHT consumer’s management API:

```
{
```

```
    "@context": {
        "edc": "https://w3id.org/edc/v0.0.1/ns/"
    },
    "@type": "TransferRequestDto",
    "connectorId": "provider",
    "counterPartyAddress": "http://provider:8282/protocol",
    "protocol": "dataspace-protocol-http",
    "contractId": "76aa024a-ab24-4d22-bce9-a1d12e8a6e2f",
    "assetId": "asset-1",
    "transferType": "IonosS3-PUSH",
    "dataDestination": {
        "type": "IonosS3",
        "storage": "s3.eu-central-2.ionoscloud.com",
        "bucketName": "uc3-consumer",
        "keyName": "asset-1.txt"
    }
}
```

```
curl -X POST "http://consumer:9192/management/v3/transferprocesses" \
-H "Content-Type: application/json" \
-H "X-API-Key: password" \

-d @transfer.json
```

The transfer process transitions through states (INITIAL, PROVISIONING, STARTED, COMPLETED), with the IonosS3Provisioner generating temporary keys (managed by Vault) to facilitate the S3 transfer. Upon completion, the astronomy data is available in the uc3-consumer bucket for processing by the RHT Orchestrator.

This workflow ensures that data transfers are executed securely and in compliance with the predefined policies, leveraging the EDC S3 Extended Connectors' capabilities within the UC3 environment.

5.4.3 OSR and Application Descriptor

The process of application onboarding begins with the developer interacting through the GUI, where they define the application's components, such as microservices, data sources, and policies, by leveraging blueprints from the Service Catalogue and Data Catalogue. The OSR enhances this abstraction by interpreting these inputs using ontologies and reasoning techniques (e.g., deduction, induction), and translating them into a machine-readable format like RDF or OWL that captures semantic relationships and dependencies. The Application Gateway then consolidates this into a structured AppD (see Annex I: OSR Application Descriptors for the full AppD), which the AI-based LCM system uses to map to specific technologies, such as K8s manifests or Docker containers.

In our specific UC, however, we are not currently utilising the Application Gateway or the full capabilities of the OSR as described, as these components have not yet been made available for use within the AC³ environment. Instead, the AppD has been manually crafted through close collaboration with the developers of the OSR. This manual process involved defining the application's components such as microservices, dependencies, and resource requirements directly with the OSR team, bypassing the automated abstraction layer typically provided by the GUI of the Application Gateway. While this approach has allowed us to align the descriptor with our needs

and the OSR's semantic reasoning capabilities, it underscores the reliance on future integration of these components to enable a more automated workflow which will be completed by the final version of UC3.

The AppD will be generated by the OSR through parameters provided by the application developers and is instrumental in facilitating the automated deployment and lifecycle management of microservices within the UC3 testbed. It delivers a structured specification of application components, ensuring reproducibility, scalability, and interoperability across the federated cloud-edge environment. This descriptor is processed by the LCM and converted within the adaptation and federation layer into K8s manifests, enabling the deployment of microservices onto the testbed clusters.

The UC3 AppD specifies each microservice essential to the UC application, alongside the resource configurations necessary for their operation on a cluster. The resources required for this UC include:

- Persistent Volumes
- Service Accounts
- Role bindings

```
Volumes_configuration:
  - VolumeName: "uc3-pv-volume"
    VolumeType: "PersistentVolume"
```

```
Security_configuration:
  - ServiceAccountName: "starlight-sa"
    ApiVersion: "v1"
    Kind: "ServiceAccount"
```

The microservices outlined in the UC3 AppD are categorized by a cluster affinity property into two deployment groups, promoting scalability and enabling the migration of certain microservices without impacting others. These microservice groups are:

Orchestrator:

- Data Connector - The EDC IONOS S3 Extended Consumer Data Connector ensures governance through contract negotiations with the provider data connector and transfers data from the UCM S3 bucket to the RHT S3 bucket.
- Orchestrator - Retrieves astronomy data from the RHT S3 bucket, batches it, and forwards it to the event queue.
- RabbitMQ - The event queue that transmits astronomy data as events to the processor's event receiver.

```
- MicroserviceName: "orchestrator"
    Version: "1.0"
    Image: "rayc/ucm-producer"
    ID: "orchestrator"
    ClusterAffinity: "orchestrator"
```

Processor:

- Event receiver - Subscribes to the event queue and directs the astronomy data to the appropriate processing application.
- Starlight - An astronomy application that processes data to provide astronomers with valuable insights into the properties of the universe.

```
- MicroserviceName: "starlight"
    Version: "1.0"
    Image: "rayc/ucm-processor"
    ID: "starlight"

    ClusterAffinity: "processor"
```

5.4.4 LCM

The application LCM functionalities are implemented with the use of MAESTRO service orchestration platform by UBITECH. The implemented functionalities include a) the deployment of application containers according to end-user-defined policies using the TMF-based standardised processes of the platform [[10],[11],[12],[13]], and b) the runtime update of the deployed services based on information received from monitoring and the decisions produced through the analytics modules. MAESTRO provides a modular framework with well-defined standardised interfaces for easily adapting to different types of external modules related to: Management of end-user application requests (i.e., OSR), monitoring, analytics, and decision engines (i.e., AC^3 monitoring and App/Resource profile engines and migration engine). Moreover, for the purpose of AC^3 integration needs, MAESTRO southbound interface is extended to integrate with RedHat's multi-cluster control plane (i.e., ACM) and multi-cluster scheduler.

The details about the integration status of MAESTRO with respect to the two core functionalities are provided in the following subsections.

5.4.4.1 Application Deployment

The Application Gateway and OSR provide a vital abstraction layer that allows the developer to define their application in a way that is agnostic to the underlying technology, by ultimately generating an AppD (based on AC^3 descriptor model) that can drive the deployment of the application. In order to anchor this to the underlying technological choice, it is essential to adopt a solution that can effectively translate the descriptor into the corresponding technology-specific implementation To this end, we have implemented an [AC^3 -OSR]-to-[MAESTRO Exposure] layer translator, that creates the K8s Manifest Files required for composing the MAESTRO service order according to TMF-641 [11]. As shown in the image below (Figure 39), the role of the application translator is to read the AC^3 AppD file generated by the OSR according to the AC^3 descriptor model and create the full set of the K8s Manifest files that essentially include all the necessary information about the containerised application components with their access permissions (service.yaml, role base(rb).yaml, service account(sa).yaml, and persistent volume(pvc).yaml files) as well as their infrastructure deployment parameters for the components (deployment.yaml files). Specifically, for UC3 and according to the UC3 AppD, there are in total 12 Manifest files that are created, 5 of which are deployment.yaml files.

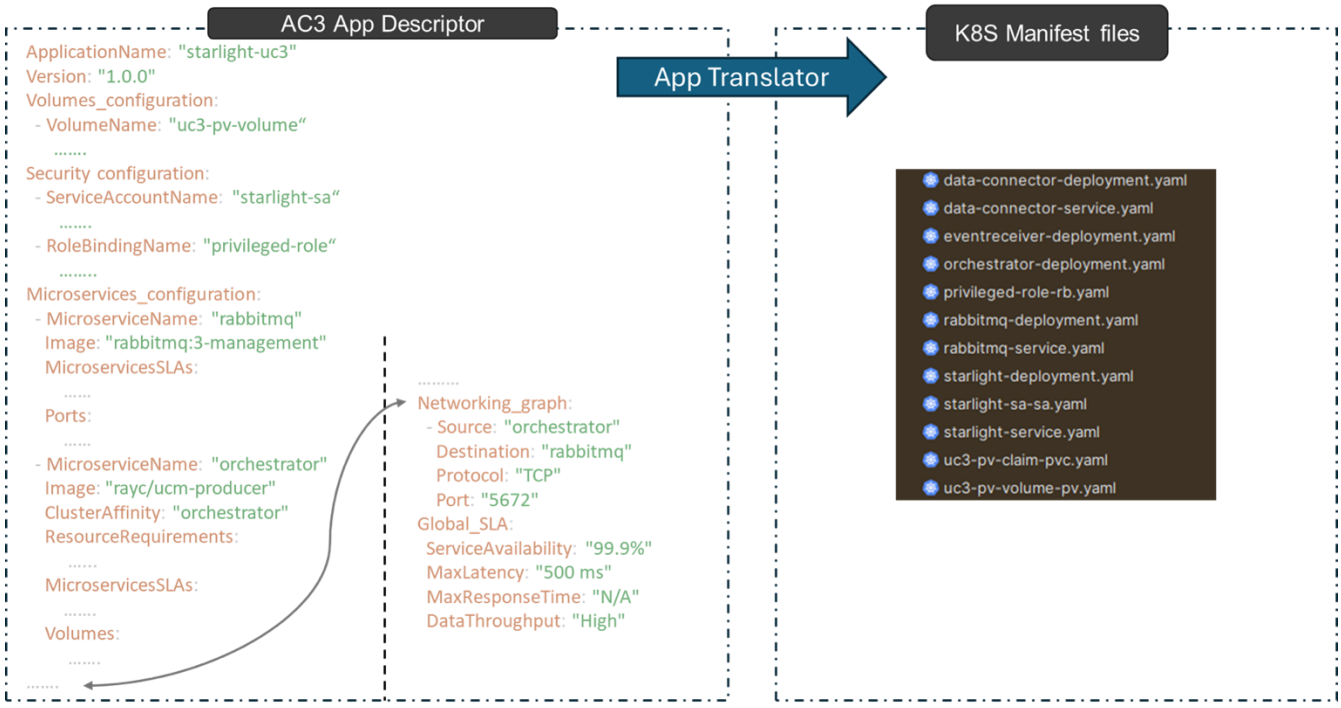

Figure 39. Schematic representation of the OSR-to-MAESTRO Exposure translation process, in which the K8S Manifests files are extracted by the AC³ App Descriptor for each micro-service.

It is noted that thanks to the clearly defined AppD model, the whole translation process can be automated, and the MAESTRO Service Ordering process can be initiated directly once a valid AppD is received from the AC³ -OSR. In this case, the creation and submission of the AppD to the MAESTRO LCM has the direct meaning of a deployment request and can be controlled by the Front-End GUI that provides the initial high-level end-user requests.

The subsequent step involves ordering the service through RedHat's multi-cluster control plane, ACM. To efficiently orchestrate application deployments, MAESTRO and ACM employ a sophisticated label-based routing mechanism to accurately determine the appropriate target cluster, such as, for instance, an OpenShift cluster, a standard K8s cluster, or a cluster with specialized resources like dedicated CPUs, among others. In essence, labels can be used to define and identify any cluster characteristic and capability. This enables MAESTRO, in conjunction with ACM, to intelligently distribute application deployments across the entire cluster continuum, based on the specific requirements and characteristics of each microservice of the application. This determination is performed by retrieving location-specific labels through metadata fields exposed by the MAESTRO TM Forum (TMF) APIs. Once the appropriate cluster target is identified, MAESTRO proceeds to transform the standard K8s resource definitions into specialized “CustomManifestWork” resources, fully compatible with RedHat’s ACM Hub. These custom resources encapsulate the detailed deployment instructions required by the control plane to effectively manage multi-cluster deployments. Deployment requests generated by this translation process are subsequently dispatched to their designated clusters via RedHat's ACM Hub, ensuring a centralized and streamlined deployment procedure.

© AC³ 2023

The overall workflow described above is depicted below in Figure 40. It begins with the end user, who provides the AppD (left side of the diagram). The Application Translator then generates the corresponding K8s manifest files and pushes them to the AC³ registry. Following this, a CI/CD pipeline is triggered to activate the Request Generator mechanism (optionally, it may also invoke the Package Generator mechanism). The Request Generator builds the complete order according to the TMF service order standard and also enriches the metadata field with labels to define the application needs (per service). These labels are subsequently used by ACM to distribute the application deployment intelligently across the clusters, as explained previously. Finally, the MAESTRO Service Engine is initiated in order to transform the K8s Manifest files to the proper resources, namely, CustomManifestWork resources, and to dispatch the request to the ACM Hub. In the last stage of the process, the ACM hub forwards the deployments to the edge site clusters. Figure 41 presents a UC3-specific example of the label-based process for creating the CustomManifestWork.

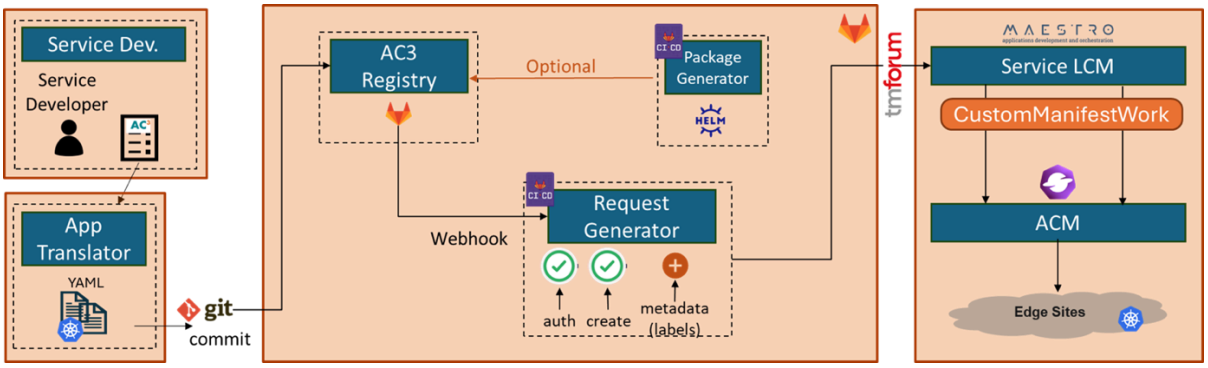

Figure 40. Process workflow for the deployment of an AC application request from the OSR translation point to the interfacing with ACM

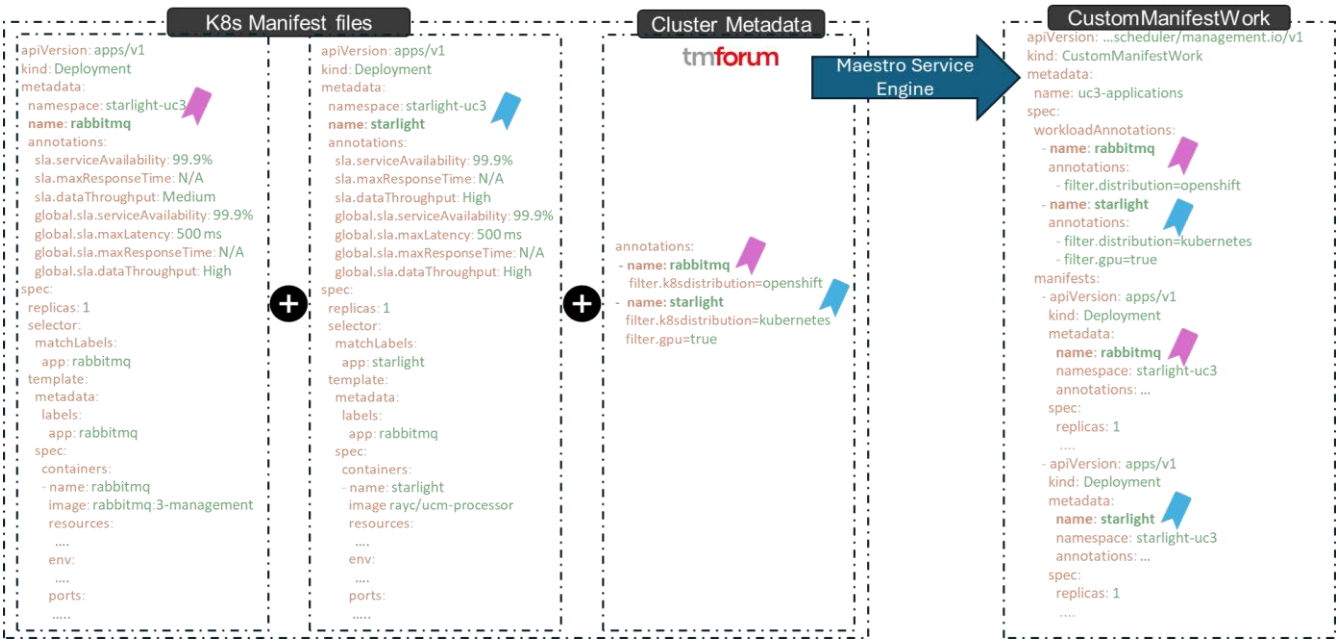

Figure 41. Schematic representation of the CustomManifestWork creation from the Manifest files and the combination of cluster metadata using a cluster labelling scheme.

5.4.4.2 Application Adaptation

MAESTRO is capable of integrating monitoring systems into its process and orchestrating feedback loops, enabling continuous LCM for each deployment. This ensures visibility, operational reliability, and efficient resource management throughout the service lifecycle.

It is noted that at the time of editing this deliverable, the interconnections between the components of the different AC^3 layers, as well as the integration steps have been defined, as shown in Figure 42 but the integration is not yet complete, since a) the MAESTRO platform runtime processes are currently updated to be able to handle custom metrics in a modular and well defined method) while offering standardised connection interfaces to the monitoring modules, and b) the AC^3 decision engines for the App and Resource profile in WP3 and WP4 require to be finalised and adjusted to the platform requirements. The work is planned for the second phase of the integration process, following the UC3 deployment phase.

Diving deeper into the LCM aspect of this specific UC, Figure 42 presents the control plane architecture, which has been designed to manage the complete lifecycle of applications across the distributed edge computing environment. Based on the three main layers of AC^3, the process begins at the upper layer (namely Application Composition and onboarding), where the end user includes in their request the SLA metrics as part of the application specifications. The App translator then will a) create the proper HPA resources and b) dispatches both the SLA metrics and the Horizontal Pod Autoscaler (HPA) resources to MAESTRO. Subsequently, MAESTRO updates the External Metrics API and Custom Metrics API of each cluster to enable the provisioning of custom metrics for any HPA resource. To support this, MAESTRO is integrated with Prometheus, which is responsible for collecting and retrieving custom metrics generated by the AC^3 AI LCM algorithms and

For each new custom metric, MAESTRO dynamically creates a ConfigMap that defines how the metric should be exposed to the K8s Custom Metrics or External Metrics APIs. As a result, HPA controllers within the clusters can retrieve these metrics and act upon them, enabling continuous and efficient intra-cluster lifecycle management. Last but not least, it is important to note that end users can continuously monitor the status of their applications along with the associated metrics, and they are able to update the initial deployment request at any time, for example, by modifying the metrics that drive autoscaling or adjusting the initial resource limit requests for each service. This architecture effectively extends the default K8s autoscaling mechanism, which is traditionally limited to CPU and memory, by introducing a dynamic, SLA-aware autoscaling strategy driven by external and application-specific metrics.

© AC^3 2023

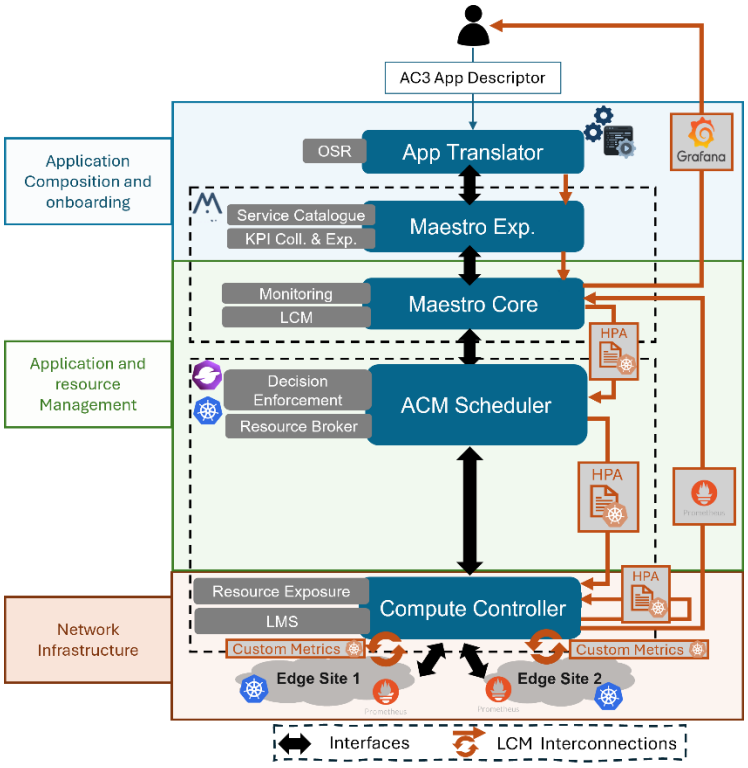

Figure 42. The MAESTRO LCM architecture adapted to the AC³ architecture

Figure 43 illustrates in detail how the custom metric exposure process works and how the AC³ AI LCM mechanism can inject its metrics into the system, enabling the creation of closed-loop adaptation systems, utilizing HPA resources, for each deployment.

In detail:

1. The user initiates a request to deploy an application, specifying the custom metric that should guide the LCM logic.

2. The MAESTRO creates a) the Prometheus adapter ConfigMaps and b) the HPA resources, that map each deployment with each custom metric. These resources are then deployed to the relevant clusters by the ACM scheduler. The Prometheus Adapter ConfigMap defines how the adapter will translate those metrics into K8s-readable custom metrics. The key config values there are:

   a. “SeriesQuery”: Specifies which metric series should the adapter query from in Prometheus

   b. “metricsQuery”: Defines how to aggregate or process the data (avg, sum etc.)

   c. “name”: Sets the name what will be exposed the custom metric via K8s Metrics or External API.

3. Prometheus collects those metrics from AI LCM mechanisms.

4. Prometheus Adapter queries Prometheus using the rules defined in the ConfigMap.

5. The Prometheus Adapter configures K8s Custom Metrics API (“/apis/custom.metrics.k8s.io”) or External Metrics API (“/apis/external.metrics.k8s.io”) to exposes the retrieved metrics.

6. Finally, HPA resources query the K8s metrics API to retrieve the custom metric values and make scaling decisions based on the thresholds defined by the user during the initial application request.

© AC³ 2023

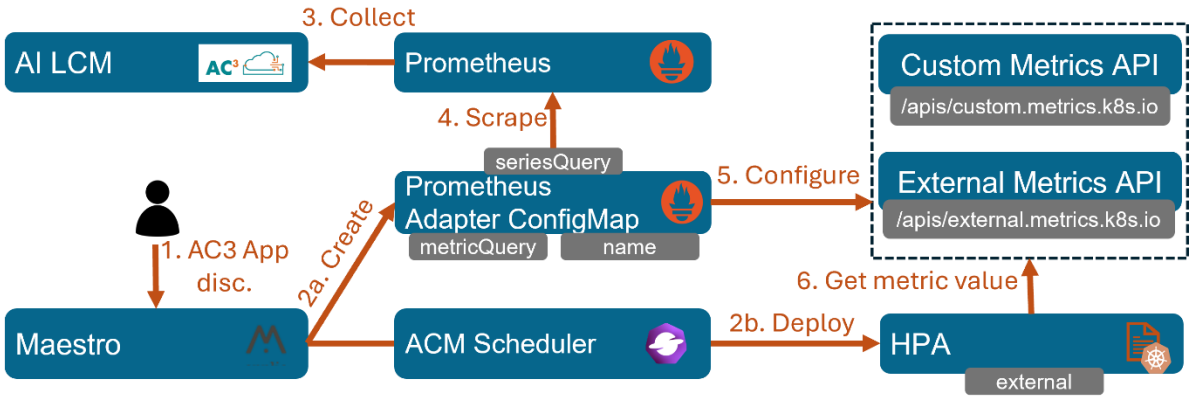

Figure 43. Custom metric exposure process and metrics injection into the system

5.4.4.3 AI LCM algorithms

In terms of AI-LCM algorithms, at this stage the core focus has been on exploring the most appropriate AI algorithms to integrate into this UC integration design. In particular, we are focusing on 2 core AI-enabled approaches under development within the project:

The XAI-enabled auto-scaler ML model, in development in WP4, can predict application QoS violations and use this to trigger corrective actions. Specifically, it monitors application CPU and RAM usage relative to application limits and uses this to predict when we need to make interventions. An XAI model also indicates the contribution of each feature to the prediction (E.g., CPU/RAM), enabling targeted mitigation of the issue. As part of this work, the Decision Engine utilises vertical autoscaling to increase the CPU or RAM for the application in question.

We are also considering, given the nature of the Astronomy processing application architecture, whether CPU and RAM provide accurate reflections of workload, given that we are using a work/event dispatching model. Instead, we are exploring using additional metrics that consider the incoming workload, which may be a better prediction of potential bottlenecks in the system. For instance, queue length and average batch processing times should provide a more accurate reflection of the workload.

We are also considering using the same algorithm for predicting QoS violations in order to mitigate the predicted performance issues via horizontal autoscaling. Using the predictions from the ML model, instead of vertically autoscaling, we can look to increase the number of Processor Pods in the system, in order to increase concurrent data processing. This work aims to utilise Maestro and the Horizontal Pod Autoscaling capability within K8s. Where specific SLA metrics are required by an application, we can convert these to HPA resources to monitor these SLA and automatically trigger scaling.

Building on this foundation, we are integrating AI-LCM algorithms into the batch processing workflow to further enhance the scalability of the astronomy data processors. While CPU and RAM usage remain under monitoring, the work/event dispatching model employed in our UC suggests these metrics may not fully reflect workload demands. To address this, we are incorporating RabbitMQ queue length as a more robust indicator of system load. Astronomy data batches are queued in RabbitMQ for processing by the RHT Orchestrator. The resulting queue length, representing unprocessed batches, is collected and analysed. This data is fed into an XGBoost-based ML model, which predicts potential performance degradation, such as elevated workload queues.

These predictions are then leveraged to inform scaling decisions through the K8s HPA. When the XGBoost model anticipates an SLA violation due to excessive queue length, the HPA responds by increasing the number of processor pods. For example, it might scale from 2 to 4 pods to distribute the workload and accelerate processing. Conversely, should the queue length fall below a lower threshold, such as 10 batches, the HPA reduces the pod count to optimise resource utilisation. This AI-driven mechanism ensures that the astronomy data processors adapt dynamically to workload fluctuations, maintaining performance efficiency while adhering to SLA commitments.

Finally, we also want to include multi-cluster replication/stateless migration in our UC execution. This would enable Processor pods with constrained resources or excessive workloads to scale not just locally (within the same cluster) but also cross-cluster, in order to exploit resource availability in other clusters. Since every event/batch is discrete, the Processors operate statelessly, so the work in WP3 focused on stateful migration may not be fully exploitable. However, we are examining whether their work on resource prediction, utilising a LSTM-based machine learning algorithm could be leveraged to trigger stateless pod migration.

5.4.5 Monitoring

5.4.5.1 Monitoring Framework Integration

To get visibility across our distributed architecture, we require a monitoring solution that can provide real-time insights, scale across our multi-cluster environment, and support long-term metric retention. This is where our integration with the monitoring framework devised under WP4, based on Prometheus and Thanos, comes into play.

In OpenShift, Prometheus is deployed and managed by default using the platform's built-in Monitoring Operator. This operator handles both the deployment and ongoing lifecycle of Prometheus, ensuring that it continuously scrapes metrics from cluster components as well as application-level metrics exposed by workloads running within the cluster. In a federated cluster environment, Prometheus instances are tailored to focus on local metric collection. These metrics are then forwarded to a central aggregation point using the remote_write feature, which allows for efficient grouping and transfer of data.

To enable centralized observability and long-term retention, Prometheus is integrated with Thanos. Each Prometheus instance is configured with remote_write to push metrics to the Thanos Receiver. This setup allows Thanos to act as a centralized storage and query layer, sending metrics from all participating sites. As a result, users gain a unified view of metrics across the entire multi-site deployment, with access to historical data and advanced querying capabilities through tools like Thanos Querier or Grafana. This architecture not only improves monitoring consistency across sites but also ensures that valuable metrics are retained and made accessible beyond the lifespan of any individual Prometheus pod or cluster.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
  uid: 186780e5-bf5d-4bed-bde5-621a5635bd71
  resourceVersion: '524193793'
```

```
  creationTimestamp: '2024-03-08T10:40:32Z'
managedFields:
  - manager: hypershift-operator-manager
    operation: Update
    apiVersion: v1
    time: '2025-03-03T12:18:46Z'
    fieldsType: FieldsV1
    fieldsV1:
      'f:data':
        .: {}
      'f:config.yaml': {}
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: http://82.223.13.241:10908/api/v1/receive
        authorization:
          credentials:
            key: token
            name: telemetry-remote-write
          type: Bearer
        queueConfig:
          batchSendDeadline: 1m
          capacity: 30000
          maxBackoff: 256s
          maxSamplesPerSend: 10000
          minBackoff: 1s
      url: https://infogw.api.openshift.com/metrics/v1/receive

      writeRelabelConfigs:
```

Figure 44. Visual representation of Our monitoring Config with our Thanos URL inserted

OpenShift inherently provides automatic service discovery for Prometheus, which detects and scrapes metrics from applications and network components, including Skupper. Prometheus collects metrics from K8s services and pods that are managed by OpenShift, making it easy to monitor the infrastructure and deployed applications.

Prometheus is currently running within OpenShift and actively monitoring metrics for both the cluster and applications. The Thanos Receiver is properly configured and is successfully aggregating metrics from multiple clusters. In addition, the RabbitMQ exporter is operational, allowing the collection and visualization of application-specific metrics. Federation between Prometheus instances is set up, enabling the central Prometheus instance to collect metrics from edge clusters, thus ensuring centralized monitoring.

5.4.5.2 Monitoring Metrics

Within UC3 there is a range of metrics that we want to leverage the monitoring framework to export. Many of these fall into the range of metrics defined in D4.1, covering standard metrics such as CPU, RAM, bandwidth, and latency. However, we are also working to export additional metrics relating to the specific architecture of our application. In Figure 45 below, we visualise the batch processing time metric and its constituent metrics. In essence, the queue length of the application is key to understanding the overall load of the entire application (not just a single processor) and to predicting the future load of each individual processor. We are exploring

integrating these metrics into the ML algorithms to give more accurate predictions of SLA violations based on this queue length.

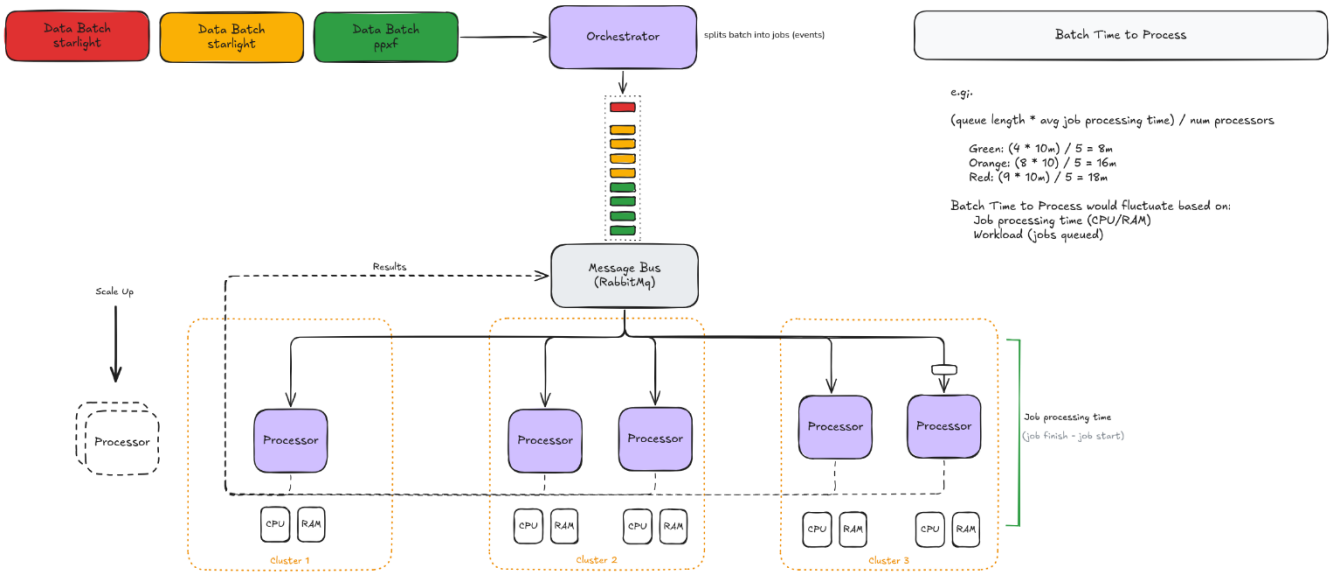

Figure 45. Batch Processing Time based on queue length

5.4.6 Compute LMS

To deliver the functionality of Compute LMS we exploit 2 core technologies, namely OpenShift (K8s) and ACM. OpenShift provides us with the capability of not just deploying and orchestrating workloads, but with a comprehensive platform to support resource provisioning such as CPU, RAM, Storage and Networking. OpenShift is a key component in supporting the scalability and resilience required for the core astronomy processing application in UC3. This allows the pre-existing processing applications (Starlight, PPXf, Steckmap) to be packaged as K8s applications, enabling them to be seamlessly and comprehensively managed (deployed, moved, removed, and scaled). It also provides an application eco-system where additional software components are available and easily deployable, such as the RabbitMQ message broker utilised as part of this UC. Another advantage of this is its ability to work in harmony with the AC^3 network Operator. Unlike relying solely on OpenShift's native operators, the AC^3 network operator is specifically designed to expose the application in a way that aligns with the unique requirements of the UC3 project. OpenShift's flexibility allows for the smooth deployment and operation of this network operator without imposing limitations on how external access is managed. This level of customization is crucial, as it allows the team to maintain control over how the application is accessed across the network, ensuring that the RabbitMQ queues and Starlight processing endpoints are reachable as needed.

In the context of UC3, we utilise multiple OpenShift clusters to represent federated infrastructures that are available to the Astronomer (via AC^3) to carry out their data processing. Within the OpenShift/K8s ecosystem, there are additional add-on components that cater to multi-cluster management. In this case, we utilise the Advanced Cluster Management add-on, based on the Cloud Native Computing Foundation Open Cluster Management project. ACM helps streamline the management of our multi-cluster setup, ensuring that all clusters are consistently monitored, maintained, and configured according to the project's needs. In particular, the workload placement capability of ACM offers us a central control point where we can deploy applications

across clusters, utilising an expressive set of rules to find the appropriate clusters or target a specific cluster. Also, the Policy construct allows for consistent and unified governance of the managed clusters, ensuring that each cluster, including newly added clusters, conforms to the required configuration.

5.4.7 Network LMS

Playing the role of Network LMS within this UC is the AC³ Network Operator developed in WP4. The goal of the network operator is to establish and manage connectivity between multiple K8s clusters in a dynamic way, leveraging a range of underlying network technologies. Leveraging the operator allows us to form a cohesive virtual application network where services and workloads can communicate across physical infrastructure boundaries.

In our UC testbed, we currently have 1 management cluster (Hub) and 2 application clusters representing individual infrastructure domains. The Network Operator is installed on the Hub cluster and is responsible for orchestrating the network configuration required to ensure inter-site connectivity. The operator automates the setup of our network by managing the lifecycle of connection tokens, distributing secrets, and creating links that connect edge sites to the central hub.

```
apiVersion: ac3.redhat.com/v1alpha1
kind: AC3Network
metadata:
name: ray-ac3network
namespace: ac3no
spec:
link:
sourceCluster: "ac3-cluster-2"
targetCluster: "ac3-cluster-1"
sourceNamespace: "sk1"
targetNamespace:
- "sk3"
applications:
- "nginx"
- "rabbitmq"
secretNamespace: "sk1"
secretName: "sk1-token"
secretName2: "sk1-token"

port: 5672
```

The Network operator establishes a secure communication between clusters, and the central site containing the network operator generates a token. This token is used by our federated sites. This will enhance the workflow by automating token distribution and secret management. Once the inter-site links are in place, the operator maintains these connections and ensures that they are kept in sync as sites scale or update over time.

Applications deployed with the Network Operator are automatically exposed to the virtual application network. This allows services running on one site to be discovered and accessed from any other site that is linked through Skupper, a visual representation of which can be seen in Figure 46. The exposure configuration is handled by the operator, which ensures consistent and reliable cross-cluster communication without requiring manual routing

rules or service mirroring configurations. This significantly reduces operational overhead and enables multi-site deployments to be managed in a declarative and scalable way.

User Interface:

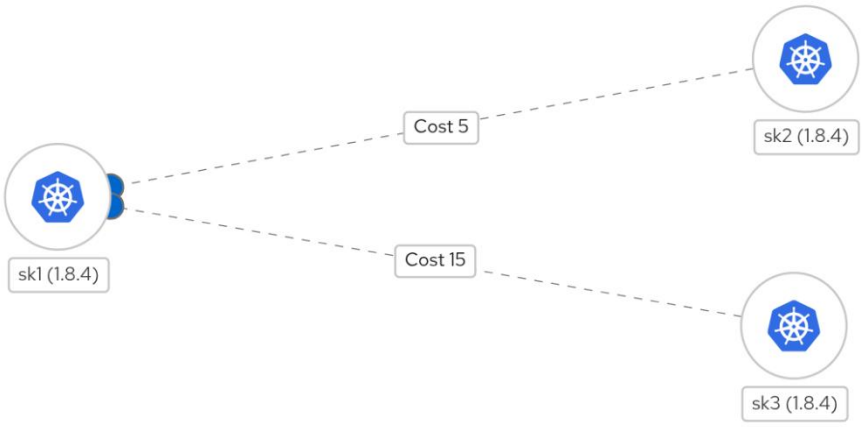

Figure 46. Visual representation of Skupper links between namespaces/clusters

One of the key technical challenges in this setup is ensuring proper synchronization between central and cross cluster sites—especially as updates and configuration changes are made centrally and need to propagate across the network. This includes both the consistency of routing information and the reliable rollout of service exposure configurations. Managing traffic flow between multiple edge sites also presents challenges around load balancing and routing efficiency. As the number of interconnected sites grows, it becomes increasingly important to monitor how traffic is distributed to avoid bottlenecks or uneven load. In addition, automated failover remains a core focus, with efforts underway to support seamless rerouting if an inter-site link fails or a node becomes temporarily unavailable.

The multi-site setup has been successfully deployed and validated, with a reliable interconnection established between the central and multi-cluster sites. Applications deployed through the Network Operator are correctly exposed and accessible across the network. Dynamic updates, such as the addition of new sites or the reconfiguration of existing links, have been tested and confirmed to propagate as expected. The system demonstrates strong resilience and flexibility, adapting to topology changes during runtime.

5.4.8 Remaining Integration

5.4.8.1 PPXF and STECKMAP Astronomy Software

While the UC3 application is designed to process astronomical data using various tools, including pPXF and STECKMAP, the complete integration of both software components into the UC3 application's processing workflow is still pending. This integration is vital to expand the UC3 application's ability to process multiple types of astronomical data effectively. It will involve ensuring the Orchestrator can correctly identify, configure, and

dispatch data specifically formatted for each tool, and that their respective outputs are seamlessly handled and stored. This integration will enable astronomers to leverage the UC3 application for a broader range of data analysis needs, maximising its utility for diverse astronomical datasets.

5.4.8.2 Finalizing Metrics for Autoscaling Model

While the monitoring framework, using Prometheus and Thanos, collects a broad range of system and application metrics, the specific metrics vital for the AI-driven autoscaling model still need to be integrated. For the UC3 application, traditional CPU and RAM usage might not fully capture the workload. As such, we are focusing on seamlessly integrating RabbitMQ queue length as an indicator of system load alongside average batch processing and wait times. This involves refining Prometheus configurations to reliably expose these specific application metrics and ensuring their smooth ingestion by the AI-LCM for both training and real-time inference. The next step is confirming that the HPA effectively consumes these custom metrics, creating an adaptation system driven by meaningful workload indicators.

5.4.8.3 AI Algorithms

The critical phase of training the AI algorithms on actual UC3 application data and their subsequent programmatic integration with the LCM system remains. This requires collecting sufficient, representative datasets from the UC3 application, particularly data correlating queue length and processing times with performance degradation. The trained XGBoost model will then need integration within the AI-LCM mechanisms, ensuring it can process real-time queue length data and generate accurate predictions of impending SLA violations. The final part of this integration involves establishing the automated pipeline for these predictions to directly inform and trigger scaling decisions via the K8s HPA, orchestrated by Maestro. This will enable the system to proactively adjust processor pod counts, ensuring dynamic adaptation to maintain performance efficiency and SLA compliance.

5.4.8.4 Maestro (LCM)

Maestro is currently running on Ubitech infrastructure and has yet to be deployed onto the AC³ testbed. Another key outstanding task is updating Maestro's runtime processes to handle custom metrics like the RabbitMQ queue length to effectively leverage the performance indicators and predictions from the monitoring framework and AI-LCM algorithms. Furthermore, Maestro must be configured to effectively utilise the outputs from the application and resource profiling decision engines from WP3 and WP4. These profiles will provide interpreted metrics and AI predictions to Maestro, driving its scaling and adaptation decisions. The completion of this phase will allow Maestro to fully orchestrate dynamic runtime updates and autoscaling of deployed services based on real-time feedback and AI insights.

5.5 UC3 Integration summary

Table 5: UC3 Integration summary

Architecture component	Sub-Component	Description	Integration summary
Application gateway (GUI)		Allow the application developer to define its application components and SLA.	Complete
OSR		Allow the generation of the AppD	Complete

Network LMS		Connects the OpenShift clusters running on the ARS cloud	Complete
Application and resource management	Monitoring	Monitors application and resource metrics	In Progress
	AI-based LCM and Decision Enforcement	1. Manage the micro-services Life Cycle 2. Horizontal autoscaling	1. In Progress 2. In Progress
		Zero Touch Configuration: Predict and describe infrastructure resources and implementation of automated corrective measures.	In Progress
	AI-Based application profile	Predicting application behavior	In Progress
	AI-Based Resource Profile	Describe the resources of the infrastructure	In Progress
Data management	1. Data Provider connector 2. Catalogue (data) 3. Data Manipulator	1. Astronomy data goes through the Data connector 2. Registers with the AC^3 Catalogues (Piveau) 3. Manipulate data for input to Orchestrator	1. Complete 2. Complete 3. Complete
Migration		1. Manages dynamic microservice migration across cloud and edge. 2. Moves microservices between clusters based on real-time data. 3. Balances loads and avoids bottlenecks.	1. Not Started 2. Not Started 3. Not Started

© AC^3 2023

6 Conclusions

The objective of deliverable D5.2 was to capture the interim status of the work done across tasks T5.1 (AC³ *components integration*) and T5.2 (*Testbed Integration*). We believe that, through the detailed account of each UC application and testbed architectures, UC integration design, as well as the thorough technical description of the implementation carried out, this intermediate report on integration successfully captures the extensive work that has been completed towards achieving our goal of a cohesive and highly effective AC³ system.

- In UC1, significant progress has been made in integration of key components such as the Data Managment and Dat Connectors, Service Catalogue, AppD, OSR and LCM.
- In UC2, significant progress has been made in the core application and application UI, as well as integration of key components such as the Application Gateway, OSR/AppD and LCM
- In UC3, work is complete/nearing completion on the core application for data orchestration and analysis, as well as integration of key components such as the Data Connectors, Monitoring Framework and AppD/OSR, as well as the Compute and Network LMS. Work is actively progressing on LCM and AI integration.

To finalize the integration across all UCs, further efforts will concentrate on the remaining components:

- For UC1, this primarily involves the full integration of the Application Gateway, LMS Edge/Cloud, the advanced AI-based LCM, AI-based profiling, Monitoring functionalities, and the initiation of Zero-touch configuration.
- UC2 will focus on completing the AI-based LCM and Decision Enforcement, Zero-touch configuration for drone availability, and the AI-based Application and Resource Profiling.
- For UC3, the key remaining tasks include finalizing the Monitoring integration, AI-based LCM, Zero Touch Configuration, and AI-Based profiles.

These final integration steps, detailed in the respective UC sections, are essential to achieve the complete vision and capabilities of the CECCM.

As a result of the extensive collaboration between the UC and component owners, we believe a clear path to realising the AC³ vision is now in place. Based on the strong foundational work described within this report, the consortium can now progress towards delivering tangible benefits of the CECCM through demonstrations, experimentation and results within the final phase of the project.

© AC³ 2023

7 References

[1] AC³ Project, "D5.1 "Initial integration and proofs of concept plan"," 03 February 2025. [Online].

[2] AC³ Project, "D2.4 "Business Analysis of CECC and Use Case Requirements"," 27 May 2024. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/07/Final-draft_AC3_Business-analysis-of-CECC-and-use-case-requirements_22.05.2024_FINAL.pdf

[3] AC³ Project, "D3.3 "Initial Report on Data Management for Applications in CECC"," 28 June 2024. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/11/AC3_D3.3_20240222_v1.0.pdf

[4] AC³ Project, "D4.1 "Initial Report on Mechanisms that Enable Green-Oriented Zero Touch Management of CECC Resources"," 30 June 2024. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/11/AC3_D4.1_to_be_submitted_annexes.pdf

[5] AC³ Project, "D3.1 "Report on the application LCM in the CEC - Initial"," 2023 December 2023. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/11/AC3_D3.1_v7.pdf

[6] AC³ Project, "D2.3 "Report on technological tools for CECC"," 31 December 2023. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/07/AC3_D2.3_v1.6_version_to_be_submitted.pdf

[7] Gil de Paz, A. et al., "MEGARA, the new intermediate-resolution optical IFU and MOS for GTC: getting ready for the telescope", in Ground-based and Airborne Instrumentation for Astronomy VI, 2016, vol. 9908, Art. no. 99081K. doi:10.1117/12.2231988.

[8] Drory, N. et al., "The MaNGA Integral Field Unit Fiber Feed System for the Sloan 2.5 m Telescope", The Astronomical Journal, vol. 149, no. 2, Art. no. 77, IOP, 2015. doi:10.1088/0004-6256/149/2/77.

[9] Bacon, R. et al., "The MUSE second-generation VLT instrument", in Ground-based and Airborne Instrumentation for Astronomy III, 2010, vol. 7735, Art. no. 773508. doi:10.1117/12.856027.

[10] TMForum, "TMF633 Service Catalog Management API v4.0.0", 2021, Available: https://www.tmforum.org/resources/standard/tmf633-service-catalog-api-user-guide-v4-0-0/

[11] TMForum, "TMF641 Service Ordering Management API v4.1.1", 2021, Available: https://www.tmforum.org/resources/specifications/tmf641-service-ordering-management-api-user-guide-v4-1-1/

[12] TMForum, "TMF638 Service Inventory Management API v4.0.1", 2020, Available: https://www.tmforum.org/resources/specification/tmf638-service-inventory-api-user-guide-v4-0-0/

[13] TMForum, "TMF674 Geographic Site Management API v4.0.1", 2020, Available: https://www.tmforum.org/resources/specification/tmf674-geographic-site-management-api-user-guide-v4-0/

© AC³ 2023

8 Annex I: OSR Application Descriptors

This Annex provides a view of the complete OSR AppD for each UC.

UC1 OSR Application Descriptor

```
ApplicationName: "UC1 IoT Data Processing"
Version: "3.0"

Microservices_configuration:
  - MicroserviceName: "edgebroker"
    Version: "0.3"
    Image: "sparkworks/ac3-edge-broker:0.3"
    ID: "edgebroker"
    EnvironmentVariables:
      - Name: "RABBITMQ_HIPE_COMPILE"
        Value: "1"

  - MicroserviceName: "logger"
    Version: "latest"
    Image: "sparkworks/ac3-amqp-http-request-logger:latest"
    ID: "logger"
    Ports:
      - "4000:4000"
    EnvironmentVariables:
      - Name: "HTTP_SERVER_PORT"
        Value: "4000"

  - MicroserviceName: "consumer"
    Version: "latest"
    Image: "sparkworks/ac3-connector-http-http-consumer:latest"
    ID: "consumer"
    Ports:
      - "28180:28180"
      - "28181:28181"
      - "28182:28182"
      - "28183:28183"
    EnvironmentVariables:
      - Name: "WEB_BASE_URL"
        Value: "http://ionos-s1.sparkworks.net"
      - Name: "WEB_HTTP_PORT"
        Value: "28180"
      - Name: "WEB_HTTP_MANAGEMENT_PORT"
```

```
      Value: "28181"
    - Name: "WEB_HTTP_PROTOCOL_PORT"
      Value: "28182"
    - Name: "WEB_HTTP_CONTROL_PORT"
      Value: "28183"
    - Name: "ASSET_NAME"
      Value: "uc1-stream"
    - Name: "PROVIDER_DOMAIN"
      Value: "http://ds.uc1.ac3.sparkworks.net:18182/protocol"

- MicroserviceName: "edgemapper"
  Version: "0.5"
  Image: "sparkworks/sw-mapper-ac3:0.5"
  ID: "edgemapper"
  Ports:
    - "5026:5026"
    - "8026:8026"
  EnvironmentVariables:
    - Name: "RABBITMQ_PORT"
      Value: "5672"
    - Name: "RABBITMQ_HOST"
      Value: "edgebroker"
    - Name: "RABBITMQ_USERNAME"
      Value: "mapperuc1"
    - Name: "RABBITMQ_PASSWORD"
      Value: "TmU5WmuikTQnDrWkRs7D"
    - Name: "QUEUE_OUT"
      Value: "mapperuc1.mapped"
    - Name: "QUEUE_IN"
      Value: "mapperuc1.data"
    - Name: "QUEUE_COMMANDS"
      Value: "mapperuc1.commands"

- MicroserviceName: "edgeapplication"
  Version: "0.4"
  Image: "sparkworks/data_manipulator_uc1:0.4"
  ID: "edgeapplication"
  ResourceRequirements:
    Cpu: "4 vCPUs"
    Memory: "8Gi"
  ReplicaCount: "1"
  Ports:
    - "5001:5001"
```

```
    EnvironmentVariables:
      - Name: "RABBITMQ_PORT"
        Value: "5672"
      - Name: "RABBITMQ_HOST"
        Value: "edgebroker"
      - Name: "RABBITMQ_USERNAME"
        Value: "ml"
      - Name: "RABBITMQ_PASSWORD"
        Value: "7Iqk7uu1Ot"
      - Name: "QUEUE_OUT"
        Value: "mapperuc1.processed.ml"
      - Name: "QUEUE_IN"
        Value: "mapperuc1.mapped.ml"

  - MicroserviceName: "edgeapplication-2"
    Version: "0.4"
    Image: "sparkworks/data_manipulator_uc1-2:0.4"
    ID: "edgeapplication-2"
    ResourceRequirements:
      Cpu: "4 vCPUs"
      Memory: "8Gi"
    ReplicaCount: "1"
    Ports:
      - "5005:5001"
    EnvironmentVariables:
      - Name: "RABBITMQ_PORT"
        Value: "5672"
      - Name: "RABBITMQ_HOST"
        Value: "edgebroker"
      - Name: "RABBITMQ_USERNAME"
        Value: "ml"
      - Name: "RABBITMQ_PASSWORD"
        Value: "7Iqk7uu1Ot"
      - Name: "QUEUE_OUT"
        Value: "mapperuc1.processed.ml"
      - Name: "QUEUE_IN"
        Value: "mapperuc1.mapped.ml-2"

Global_SLA:
  ServiceAvailability: "99.9%"
  MaxLatency: "500 ms"
  MaxResponseTime: "Low"
  DataThroughput: "High"
```

UC1 AppD

UC2 OSR Application Descriptor

```yaml
ApplicationName: "Surveillance System"
Version: "1.0.0"

Microservices_configuration:
  - MicroserviceName: "backend"
    Version: "1.0"
    Image: "capy8ra/ac3-uc2-backend:latest"
    ID: "backend"
    Dependencies:
      - "database"
    ResourceRequirements:
      Cpu: "4 vCPU"
      Memory: "8Gi"
    MicroservicesSLAs:
      ServiceAvailability: "99.9%"
      MaxResponseTime: "Low"
      DataThroughput: "High"
    ReplicaCount: "1"
    EnvironmentVariables:
      - Name: "DJANGO_DEBUG"
        Value: "False"
      - Name: "LOGGING_LEVEL"
        Value: "INFO"
      - Name: "DB_HOST"
        Value: "db"
      - Name: "DB_PORT"
        Value: "5432"
      - Name: "DB_NAME"
        Value: "ac3"
      - Name: "DB_USER"
        Value: "postgres"
      - Name: "DB_PASS"
        Value: "root"
      - Name: "RECAPTCHA_SECRET_KEY"
        Value: "6LfUgiEqAAAAADM04T7V8GNTL6VCvWtW_UJDIx0J"
    apiEndpoint: "http://backend.fingletek.com"
    apiPort: "8000"
    Protocol: "HTTP/REST"
```

```
    InternetAccess: "true"

    GeographicalArea:
      Region: "Central Cloud"
      LocationType: "cloud"

  - MicroserviceName: "frontend"
    Version: "1.0"
    Image: "capy8ra/ac3-uc2-frontend:latest"
    ID: "frontend"
    Dependencies:
      - "backend"
      - "deepstream"
    ResourceRequirements:
      Cpu: "1 vCPU"
      Memory: "2Gi"
    MicroservicesSLAs:
      ServiceAvailability: "99.9%"
      MaxResponseTime: "Low"
      DataThroughput: "High"
    ReplicaCount: "1"
    EnvironmentVariables:
      - Name: "VITE_USER_SERVICE_URL"
        Value: "http://172.21.16.156:30033/api/v1"
      - Name: "VITE_USER_SERVICE_BASE_URL"
        Value: "http://172.21.16.156:30033"
      - Name: "VITE_REACT_APP_SITE_KEY"
        Value: "6LfUgiEqAAAAAKWzfsVqrlI6YbpXgUZde85ip3z-"
    apiEndpoint: "http://frontend.fingletek.com"
    apiPort: "5173"
    Protocol: "HTTP/REST"
    InternetAccess: "true"

    GeographicalArea:
      Region: "Central Cloud"
      LocationType: "cloud"

  - MicroserviceName: "deepstream"
    Version: "1.1.0"
    Image: "capy8ra/ac3-uc2-ds:28"
    ID: "deepstream"
    Dependencies:
      - "backend"
```

```
    - "database"
  ResourceRequirements:
    Cpu: "4 vCPUs"
    Memory: "16Gi"
    Storage: "N/A"
    Gpu: "NVIDIA GPU (specific model based on throughput)"
  MicroservicesSLAs:
    ServiceAvailability: "99.9%"
    MaxResponseTime: "Low"
    DataThroughput: "High"
  ReplicaCount: "1"
  EnvironmentVariables:
    - Name: "LOG_LEVEL"
      Value: "INFO"
    - Name: "DB_HOST"
      Value: "db"
    - Name: "DB_PORT"
      Value: "5432"
    - Name: "DB_NAME"
      Value: "ac3"
    - Name: "DB_USER"
      Value: "postgres"
    - Name: "DB_PASSWORD"
      Value: "root"
    - Name: "NO_DISPLAY"
      Value: "1"
  Protocol: "TCP/RTSP"
  InternetAccess: "false"

  GeographicalArea:
    Region: "Edge"
    LocationType: "edge"

- MicroserviceName: "db"
  Version: "8.1.0"
  Image: "postgres:17"
  ID: "database"
  ResourceRequirements:
    Cpu: "0.2 vCPU"
    Memory: "256MB"
    Storage: "10GB"
  MicroservicesSLAs:
    ServiceAvailability: "99.9%"
```

```
      MaxResponseTime: "Low"
      DataThroughput: "High"
    ReplicaCount: "3"
    EnvironmentVariables:
      - Name: "POSTGRES_USER"
        Value: "postgres"
      - Name: "POSTGRES_PASSWORD"
        Value: "root"
      - Name: "POSTGRES_DB"
        Value: "ac3"
    apiEndpoint: "psql://database.fingletek.com"
    apiPort: "5432"
    Protocol: "TCP"
    InternetAccess: "false"

    GeographicalArea:
      Region: "Central Cloud"
      LocationType: "cloud"

Networking_graph:
  - Source: "backend"
    Destination: "db"
    Protocol: "TCP"
    Port: "5432"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"

  - Source: "frontend"
    Destination: "deepstream"
    Protocol: "TCP"
    Port: "8585"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"

  - Source: "frontend"
    Destination: "backend"
    Protocol: "TCP"
```

```
    Port: "8000"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"

Global_SLA:
  ServiceAvailability: "99.9%"
  MaxLatency: "500 ms"
  MaxResponseTime: "Low"
  DataThroughput: "High"
```

UC2 AppD

UC3 OSR Application Descriptor

```
ApplicationName: "starlight-uc3"
Version: "1.0.0"

Volumes_configuration:
- VolumeName: "uc3-pv-volume"
VolumeType: "PersistentVolume"
StorageClass: "standard"
Capacity: "10Gi"
AccessModes:
- "ReadWriteOnce"
HostPath:
Path: "/mnt/ucmdata"
ClaimName: "uc3-pv-claim"
ClaimSpec:
StorageClassName: "standard"
AccessModes:
- "ReadWriteOnce"
Resources:
Requests:
Storage: "3Gi"

Security_configuration:
- ServiceAccountName: "starlight-sa"
```

```
ApiVersion: "v1"
Kind: "ServiceAccount"
Metadata:
Name: "starlight-sa"
- RoleBindingName: "privileged-role"
ApiVersion: "rbac.authorization.k8s.io/v1"
Kind: "RoleBinding"
Metadata:
Name: "privileged-role"
RoleRef:
ApiGroup: "rbac.authorization.k8s.io"
Kind: "ClusterRole"
Name: "system:openshift:scc:privileged"
Subjects:
- Kind: "ServiceAccount"
Name: "starlight-sa"

Microservices_configuration:
- MicroserviceName: "rabbitmq"
Version: "3-management"
Image: "rabbitmq:3-management"
ID: "rabbitmq"
ClusterAffinity: "orchestrator"
Dependencies: []
ResourceRequirements:
Cpu: "0.5 vCPU"
Memory: "1Gi"
MicroservicesSLAs:
ServiceAvailability: "99.9%"
MaxResponseTime: "N/A"
DataThroughput: "Medium"
ReplicaCount: "1"
Ports:
- ContainerPort: "5672"
- ContainerPort: "15672"
EnvironmentVariables:
- Name: "RABBITMQ_DEFAULT_USER"
Value: "guest"
```

```
- Name: "RABBITMQ_DEFAULT_PASS"
Value: "guest"

- MicroserviceName: "orchestrator"
Version: "1.0"
Image: "rayc/ucm-producer"
ID: "orchestrator"
ClusterAffinity: "orchestrator"
Dependencies:
- "rabbitmq"
ResourceRequirements:
Cpu: "2 vCPU"
Memory: "4Gi"
MicroservicesSLAs:
ServiceAvailability: "99.9%"
MaxResponseTime: "N/A"
DataThroughput: "High"
ReplicaCount: "1"
ServiceAccountName: "starlight-sa"
SecurityContext:
privileged: true
EnvironmentVariables:
- Name: "RABBITMQ_USER"
Value: "guest"
- Name: "RABBITMQ_PASSWORD"
Value: "guest"
- Name: "RABBITMQ_HOST"
Value: "rabbitmq"
- Name: "RABBITMQ_PORT"
Value: "5672"
- Name: "INPUT_DIR"
Value: "/starlight/data/input"
- Name: "OUTPUT_DIR"
Value: "/starlight/data/output"
- Name: "BATCH_SIZE"
Value: 5
Ports:
- ContainerPort: "5672"
```

```
Volumes:
- Name: "uc3-pv-storage"
VolumeSource:
PersistentVolumeClaim:
ClaimName: "uc3-pv-claim"
VolumeMounts:
- Name: "uc3-pv-storage"
MountPath: "/starlight/"
InitContainers:
- Name: "init"
Image: "busybox:1.28"
securityContext:
privileged: true
volumeMounts:
- mountPath: "/starlight/"
name: "uc3-pv-storage"
Command:
- sh
- -c
- |
if [ ! -d /starlight/data ]; then mkdir -p /starlight/data; fi;
if [ ! -d /starlight/runtime ]; then mkdir -p /starlight/runtime; fi;
if [ ! -d /starlight/runtime/infiles ]; then mkdir -p /starlight/runtime/infiles; fi;
if [ ! -d /starlight/runtime/input ]; then mkdir -p /starlight/runtime/input; fi;
if [ ! -d /starlight/data/input ]; then mkdir -p /starlight/data/input; fi;
if [ ! -d /starlight/data/output ]; then mkdir -p /starlight/data/output; fi;
if [ ! -d /starlight/data/input/processed ]; then mkdir -p /starlight/data/input/processed; fi;
if [ ! -f /starlight/runtime/processlist.txt ]; then touch /starlight/runtime/processlist.txt; fi;

- MicroserviceName: "data-connector"
Version: "1.0"
Image: "quay.io/bcapper30/ionos-s3-consumer-env"
ID: "data-connector"
ClusterAffinity: "orchestrator"
Dependencies:
- "orchestrator"
ResourceRequirements:
Cpu: "1 vCPU"
```

```
Memory: "2Gi"
MicroservicesSLAs:
ServiceAvailability: "99.9%"
MaxResponseTime: "N/A"
DataThroughput: "Medium"
ReplicaCount: "1"
EnvironmentVariables:
- Name: "JAVA_TOOL_OPTIONS"
Value: "-Dedc.fs.config=/app/resources/config.properties"
- Name: "EDC_PARTICIPANT_ID"
Value: "consumer"
- Name: "WEB_HTTP_PORT"
Value: "9191"
- Name: "WEB_HTTP_PATH"
Value: "/api"
- Name: "WEB_HTTP_MANAGEMENT_PORT"
Value: "9192"
- Name: "WEB_HTTP_MANAGEMENT_PATH"
Value: "/management"
- Name: "WEB_HTTP_PROTOCOL_PORT"
Value: "9292"
- Name: "WEB_HTTP_PROTOCOL_PATH"
Value: "/protocol"
- Name: "WEB_HTTP_CONTROL_PORT"
Value: "9293"
- Name: "WEB_HTTP_CONTROL_PATH"
Value: "/control"
- Name: "WEB_HTTP_PUBLIC_PORT"
Value: "9393"
- Name: "WEB_HTTP_PUBLIC_PATH"
Value: "/public"
- Name: "EDC_DSP_CALLBACK_ADDRESS"
Value: "http://consumer:9292/protocol"
- Name: "EDC_DATAPLANE_TOKEN_VALIDATION_ENDPOINT"
Value: "http://localhost:9293/control/token"
- Name: "EDC_DATAPLANE_API_PUBLIC_BASEURL"
Value: "http://localhost:9393/public"
- Name: "EDC_DSP_HTTP_ENABLED"
```

```
Value: "true"
- Name: "EDC_API_AUTH_KEY"
Value: "password"
- Name: "EDC_TRANSFER_PROXY_TOKEN_SIGNER_PRIVATEKEY_ALIAS"
Value: "edc.connector.private.key"
- Name: "EDC_TRANSFER_PROXY_TOKEN_VERIFIER_PUBLICKEY_ALIAS"
Value: "edc.connector.public.key"
- Name: "EDC_VAULT_HASHICORP_URL"
Value: "http://vault:8200"
- Name: "EDC_VAULT_HASHICORP_TOKEN"
Value: "test-token"
- Name: "EDC_VAULT_HASHICORP_TIMEOUT_SECONDS"
Value: "30"
- Name: "EDC_IONOS_ACCESS_KEY"
Value: "xxx"
- Name: "EDC_IONOS_SECRET_KEY"
Value: "xxx"
- Name: "EDC_IONOS_ENDPOINT_REGION"
Value: "eu-central-2"
- Name: "EDC_IONOS_TOKEN"
Value: "xxx"
Ports:
- ContainerPort: 9191
- ContainerPort: 9192
- ContainerPort: 9292
- ContainerPort: 9293
- ContainerPort: 9393

- MicroserviceName: "starlight"
Version: "1.0"
Image: "rayc/ucm-processor"
ID: "starlight"
ClusterAffinity: "processor"
Dependencies:
- "eventreceiver"
ResourceRequirements:
Cpu: "4 vCPU"
Memory: "8Gi"
```

```
MicroservicesSLAs:
ServiceAvailability: "99.9%"
MaxResponseTime: "N/A"
DataThroughput: "High"
ReplicaCount: "1"
ServiceAccountName: "starlight-sa"
SecurityContext:
privileged: true
WorkingDirectory: "/docker/starlight/STARLIGHTv04"
Command:
- ./bash_script2.sh
Volumes:
- Name: "uc3-pv-storage"
VolumeSource:
PersistentVolumeClaim:
ClaimName: "uc3-pv-claim"
VolumeMounts:
- Name: "uc3-pv-storage"
MountPath: "/starlight/"
Ports:
- ContainerPort: 8080

- MicroserviceName: "eventreceiver"
Version: "1.0"
Image: "rayc/ucm-receiver"
ID: "eventreceiver"
ClusterAffinity: "processor"
Dependencies:
- "rabbitmq"
ResourceRequirements:
Cpu: "1 vCPU"
Memory: "2Gi"
MicroservicesSLAs:
ServiceAvailability: "99.9%"
MaxResponseTime: "N/A"
DataThroughput: "High"
ReplicaCount: "1"
EnvironmentVariables:
```

```
- Name: "RABBITMQ_USER"
Value: "guest"
- Name: "RABBITMQ_PASSWORD"
Value: "guest"
Volumes:
- Name: "uc3-pv-storage"
VolumeSource:
PersistentVolumeClaim:
ClaimName: "uc3-pv-claim"
VolumeMounts:
- Name: "uc3-pv-storage"
MountPath: "/starlight/"

Networking_graph:
- Source: "orchestrator"
Destination: "rabbitmq"
Protocol: "TCP"
Port: "5672"
- Source: "data-connector"
Destination: "orchestrator"
Protocol: "HTTP"
Port: "9192"
- Source: "eventreceiver"
Destination: "rabbitmq"
Protocol: "TCP"
Port: "5672"
- Source: "eventreceiver"
Destination: "starlight"
Protocol: "HTTP"
Port: "8080"

Global_SLA:
ServiceAvailability: "99.9%"
MaxLatency: "500 ms"
MaxResponseTime: "N/A"
DataThroughput: "High"
```

UC3 AppD