D.5.3 Results and Achievements

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	D5.3 Results and Achievements		
Work Package	WP5		
Contractual due date	M38: 31st December 2025	**Actual submission date**	M39: March 2026
Type	(Report, Document)	**Dissemination Level**	PU (Public)
Lead Beneficiary	ARS		
Responsible Author	David Ruiz (ARS)		
Contributors	Abd Elghani Meliani (EUR), Ayoub Mokhtari (EUR), Adlen Ksentini (EUR), Ben Capper (RHT), Ray Carroll (RHT), Mario Chamorro (UCM), Eduardo Ojeda (IQU), Jeffrey Redondo (IQU) , Dimitrios Amaxilatis, Themistoklis Sarantakos (SPA), Kadouma Abdelhak (FIN), Qing Li (FIN), Hammed Ahmed (FIN), Ibrahim Afolabi (FIN), Rasheed Alabi (FIN), Wassim Kribaa (FIN) Andronikou Elias, Ioannis Lakoumentas, Dionysios Chachampis, Vasilis Avgerinos, Alaa AlZailaa (ISI/ATH),Vrettos Moulos (UPRC)		
Peer reviewer(s)	Abd Elghani Meliani (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.

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
V0.1	24/09/2025	5%	Initial Deliverable Structure	David Ruiz (ARS)
V0.2	10/01/2026	10%	UC1 Description Scenarios and Results	Eduardo Ojeda (IQU) Jeffrey Redondo (IQU) Dimitrios Amaxilatis (SPA) Themistoklis Sarantakos (SPA)
V0.3	04/02/2026	20%	UC2 Description Scenarios and Results	Abdelhak Kadouma (FGTEK)
V0.4	18/02/2026	30%	UC3 Description Scenarios and Results	Ben Capper (RHT)
V0.5	15/03/2026	40%	Review, format and conclusion	David Ruiz (ARS)
V0.7	23/03/2026	75%	Review, format and conclusion	David Ruiz (ARS)
V0.8		80%	Peer review	
V0.9		90%	TM review	Adlen Ksentini (EUR)
V1.0		100%	Addressing TM Reviews	

Disclaimer

Copyright message

Table of Content

1 Executive Summary 10
2 Introduction 12
2.1 Purpose and objectives 12
2.2 Link with other project activities 12
2.3 Mapping AC3 Outputs 12
2.4 Deliverable Overview and Report Structure 13
3 Use Case 1 - IoT smart sensing and monitoring 15
3.1 Use Case Description 15
3.2 Experimental Scenarios 16
3.2.1 Scenario 1: Zero-touch configuration, application and data management 17
3.2.2 Scenario 2: Application Descriptor Composition and OSR Integration 17
3.2.3 Scenario 3: Time to process and react to sensor data 17
3.2.4 Scenario 4: Seamless Microservice Deployment and Migration 18
3.2.5 Scenario 5: AI-powered Infrastructure Monitoring & Control Service at the Edge 18
3.3 Key Results and Achievements 18
3.3.1 Scenario 1: Zero-touch Configuration and Management 18
3.3.2 Scenario 2: Semantic-aware Application Management 21
3.3.3 Scenario 3: Edge Processing Performance Gains 22
3.3.4 Scenario 4: Dynamic Microservice Migration 24
3.3.5 Scenario 5: AI-powered Edge Intelligence 25
3.4 Lessons Learned and Recommendations 27
3.4.1 Scenario 1 – Zero-touch Configuration and Management 27
3.4.2 Scenario 2 – Application Descriptor Composition and OSR Integration 27
3.4.3 Scenario 3 – Edge Processing and Reaction Time 28
3.4.4 Scenario 4 – Seamless Deployment and Migration 28
3.4.5 Scenario 5 – AI-powered Edge Monitoring 28
4 Use Case 2 - Smart Monitoring using UAVs 30
4.1 Use Case Description 30
4.2 Experimental Scenarios 32
4.2.1 Scenario 1: Zero-touch definition and deployment 33
4.2.2 Scenario 2: Reducing the Load generated by video traffic 34
4.2.3 Scenario 3: Service continuity through migration 34
4.2.4 Scenario 4: SD-WAN-based multi-site connectivity and secure service exposure 35
4.2.5 Scenario 5: Monitoring-driven autonomous adaptation (zero-touch operations) 35
4.3 Key Results and Achievements 35
4.3.1 Scenario 1: Zero-touch definition & deployment 36
4.3.2 Scenario 2: Reducing the load generated by video traffic 38
4.3.3 Scenario 3: Service continuity through migration 40
4.3.4 Scenario 4: SD-WAN multi-site networking 43
4.4 Lessons Learned and Recommendations 44
5 Use Case 3 - Distributed computing large-scale data 46
5.1 Introduction 46
5.2 Use Case Description 46
5.2.1 KPI Validation 47
5.3 Experimental Scenarios 48

5.3.1 Scenario 1: Zero-Touch Application Lifecycle Management 48
5.3.2 Scenario 2: Federated Cloud-Edge Networking 48
5.3.3 Scenario 3: Dynamic Microservice Scaling 49
5.3.4 Scenario 4: Monitoring and Performance Tracking 49
5.3.5 Scenario 5: Data Management 49
5.4 Key Results and Achievements 49
5.4.1 Scenario 1: Zero-Touch Application Lifecycle Management 50
5.4.2 Scenario 2: Federated Cloud-Edge Networking 52
5.4.3 Scenario 3: Dynamic Microservice Scaling 54
5.4.4 Scenario 4: Monitoring and Performance Tracking 55
5.4.5 Scenario 5: Data Management 56
5.5 Lessons Learned and Recommendations 58
5.5.1 Scenario 1 - Zero Touch Configuration 58
5.5.2 Scenario 2 - Federated Cloud-Edge Networking 58
5.5.3 Scenario 3 - Dynamic Microservice Scaling 59
5.5.4 Scenario 4 - Monitoring and Performance Tracking 59
5.5.5 Scenario 5 - Data Management 60
6 Conclusions 61

Figure 1: Automatically generated UC1 deployment descriptor and successful LiSO deployment notification. . 19

Figure 2: UC1 deployed application view showing completed deployment metadata and the instantiated UC1 microservices. 20

Figure 3: Runtime validation of the UC1 deployment in Kubernetes, showing all deployed pods in the Running state. 20

Figure 4: OSR output for UC1, showing the structured application descriptor generated from the developer-defined application composition. 21

Figure 5: Confirmation of the OSR-to-LiSO handoff, showing that the generated descriptor was successfully translated and transferred to the lifecycle-management layer. 22

Figure 6: UC1 migration evidence showing source-side edge pods in Terminating state while destination-side cloud pods are already Running and Ready. 24

Figure 7: UC1 Grafana dashboard for the Sensirion temperature sensor, showing runtime anomaly-detection and forecasting outputs together with uncertainty and latency histories. 25

Figure 8: UC1 Grafana dashboard for the Shelly Motion 2 sensor, showing runtime anomaly-detection and forecasting outputs together with uncertainty and latency histories. 26

Figure 9. UAV far edge node 32

Figure 10. The covered area of the drone 33

Figure 11. Screenshot of the application (front-end microservice) 33

Figure 12. OSR Output – NSD 37

Figure 13. Deployed UC2 37

Figure 14 UC2 – Services 38

Figure 15: Comparison of cloud-only (case 01) and cloud–edge (case 02) deployments in UC2, highlighting transmitted and received traffic and the resulting traffic reduction when processing is moved to the UAV-mounted far-edge device. 39

Figure 16: Comparison of network traffic between cloud-based and far-edge deployment of the detection model, highlighting the reduction in transmitted and received data when leveraging CECCM-enabled edge capabilities. 39

Figure 17: Fine-tuning pipeline for microservice migration: initial model training on a general dataset, followed by fine-tuning on real-time monitoring data for infrastructure-specific predictions. 40

Figure 18: Illustration of the fine-tuned model's predictions versus actual resource usage, highlighting difficulties in accurately capturing peak values. 41

Figure 19: Migration error rates as a function of alpha. Lower alpha values reduce missed migrations (protecting performance), while higher alpha values reduce unnecessary migrations (minimizing resource waste). 42

Figure 20: Downtime and total migration time for each scenario. 42

Figure 21: Jitter and RTT CDF Overlay A vs Overlay B. 43

Figure 22: FPS comparing with and without SD-WAN enabled traffic redirection. 44

Figure 23: OSR Output - YAML Format 50

Figure 24: Deployed Applications 51
Figure 25: Use Case 3 – Applications. 51
Figure 26: Network topology of Cluster 1 and Cluster 4. 52
Figure 27: Skupper Router / Network Link Visualisation. 53
Figure 28: Open Connections Graph 53
Figure 29: Processing Time vs Number of Processing Pods 54
Figure 30: EDC Transfer Duration per File 56
Figure 31: Average Time per Lifecycle Stage 57

Table 1: Adherence to AC³ GA Deliverable & Tasks Descriptions 13

Table 2. UC1 tested components 15

Table 3: KPIs of UC1 16

Table 4: Edge deployment reduces network latency by up to 74% compared to cloud, with significantly improved consistency and overall ~65% lower end-to-end response time despite minor compute overhead. 23

Table 5. Local edge filtering cuts cloud egress traffic by 96.9%, dramatically reducing bandwidth usage and enabling scalable, cost-efficient IoT deployments. 23

Table 6. UC2 tested components 31

Table 7: KPIs of UC2 31

Table 8. UC3 tested components 46

Table 9: KPIs of UC3 47

Table 10. UC3 Horizontal Scaling Results. Processing time decreases significantly as the number of pods increases across all dataset sizes, demonstrating effective dynamic microservice scaling. 54

Table 11. UC3 Data Transfer Performance via EDC. Inbound and outbound S3 transfers achieved 100% reliability with consistent average durations (~16–17 seconds), confirming robust and policy-compliant data exchange performance. 56

Table 12. UC3 EDC Transfer Lifecycle Breakdown. End-to-end transfer time (~17–18s) is primarily dominated by provisioning and data movement phases, with minor directional differences arising from extended outbound contract negotiation steps. 57

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
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

1 Executive Summary

The AC^3 project aims to design, develop, and evaluate a Cloud Edge Continuum Computing Manager (CECCM) enhanced with AI/ML and explainable AI (XAI). Its primary goal is to manage the lifecycle of microservice-based applications by redefining service-level agreements (SLAs), predicting application behavior, and semantically describing microservices, all while ensuring SLA compliance. In parallel, AC^3 focuses on efficient management of the Cloud Edge Continuum (CEC) infrastructure, including far-edge resources, through advanced mechanisms such as stateful service migration, dynamic resource scaling, and energy optimization, maintaining a balance between infrastructure capabilities and application requirements. Further, the project leverages programmable networking to guarantee Quality of Service (QoS) for communication between microservices across distributed computing nodes using SD-WAN. Beyond these capabilities, AC3 integrates data management procedures at the platform level (PaaS) within the CECCM, ensuring compliance with Gaia-X principles.

The innovative AC^3 architectural design, developed to meet the dynamic and evolving needs of the CECC, was introduced in WP2. This architecture is organized into three distinct planes: the User plane, the Management plane, and the CECC plane. Together, the User and Management planes form the core of the CECCM, providing essential components that enable seamless and intuitive interactions for application developers.

The User Plane is meticulously designed to prioritize user-friendly interfaces, serving as the central hub for developers to interact with the CECCM. This plane is equipped with robust tools and components that streamline development processes and enhance usability. Key among these is the Application Gateway, which acts as a critical bridge, enabling developers to seamlessly access and engage with the CECCM. The Catalogue offers a comprehensive repository of application descriptors and crucial data source information, empowering developers with the resources needed to build and deploy applications effectively. Additionally, the Ontology and Semantic-Aware Reasoner (OSR) is a standout feature, providing advanced capabilities to interpret and manage complex policies defined by various CECCM stakeholders. Complementing these tools are versatile interfaces that facilitate essential operations related to computing, networking, and data management tasks. Together, these components make the User Plane an indispensable part of the AC^3 framework, enhancing accessibility, functionality, and the overall developer experience. The Management plane serves as the cornerstone of the CECCM, embodying operational excellence and robust administrative capabilities. It is meticulously designed to integrate key functionalities, including Application Lifecycle Management (LCM) and Resource Orchestration, ensuring the seamless operation of applications while efficiently overseeing the CECC infrastructure.

The AC³ data management platform enables application developers to incorporate data lifecycle capabilities directly into their deployments. The finalized framework now fully supports efficient data integration, semantic interoperability, advanced data discovery, secure and compliant data sharing, real-time streaming, cataloguing of both data and services, and dynamic deployment—all underpinned by strong privacy and scalability features.

WP3 and WP4 delivered the essential AI-based algorithms and mechanisms needed to develop the User Plane components, specifically the Application Gateway and OSR. They also contributed to the development of management plane components, including LCM, monitoring, resource exposure, and others. Additionally, the design and implementation of a PaaS solution were carried out to integrate data management with the application lifecycle, supporting both hot and cold data.

In this deliverable, D5.3 Results and Achievements, we evaluate the AC^3 concepts developed in WP2, as well as the performance of the components designed in WP3 and WP4. This evaluation is conducted using three representative use cases:

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. In this UC, the objective is to demonstrate the CECCM's components capabilities to deploy and run microservices at the edges of the CEC infrastructure, enabling low-latency processing close to data sources. The use case also demonstrates zero-touch configuration and automated application management, including advanced data handling through microservice composition for processing hot data. By ensuring minimal processing and reaction time to sensor inputs, the platform supports real-time or near-real-time responsiveness, which is essential for delivering efficient and adaptive smart building services.

UC2, Smart Monitoring System using UAVs, leverages UAVs in combination with AI/ML technologies and edge computing to enhance video surveillance and environmental monitoring capabilities. It demonstrates zero-touch configuration and automated application management, enabling efficient deployment and operation of services. Machine learning workloads are executed directly at the far-edge on NVIDIA Jetson platforms embedded in UAVs, allowing real-time data processing close to the source. Additionally, the system supports seamless migration of microservices between far-edge and edge environments, ensuring continuity, adaptability, and optimal resource utilization across the computing continuum.

UC3 focuses on deploying and executing astronomical software capable of processing hundreds of terabytes of data cubes, addressing the demands of large-scale scientific workloads. It integrates advanced scientific applications within hybrid cloud-native infrastructures, leveraging AI-driven algorithms to optimize computation and resource utilization. The system supports zero-touch configuration and automated application management, alongside efficient data handling through microservice composition tailored for cold data processing. Additionally, it relies on a federated cloud and edge virtualized platform to orchestrate and run microservices, ensuring scalability, flexibility, and efficient distribution of computational tasks across the infrastructure.

Each UC evaluates the performance of various AC^3 components and verifies the set of KPIs defined in the of DoA. The results confirm that the AC^3 approach efficiently manages the application lifecycle, including definition, deployment, and SLA assurance, across the Cloud Edge Continuum. This is achieved by leveraging AI and ML-based methods, while also supporting data management that efficiently handles both cold and hot data sources.

2 Introduction

This section defines the context of the work presented in this deliverable. It outlines its main purpose, key objectives, and its relationship with the overall project framework as well as with other related deliverables. Furthermore, it describes the expected outcomes and their alignment with the commitments specified in the Grant Agreement. The section concludes with an overview of the deliverable's structure.

2.1 Purpose and objectives

The Cloud-Edge Continuum Computing (CECC) paradigm unifies cloud and edge resources into a cohesive and collaborative computing environment. Within this context, the Agile and Cognitive Cloud-Edge Continuum Management (CECCM) architecture developed in AC^3 enables seamless convergence, unlocking the full potential of modern computing infrastructures. It allows microservice-based applications to dynamically adapt to changing conditions across the continuum. At the core of this architecture, the CECC Manager intelligently orchestrates the placement and migration of microservices to optimize performance while minimizing energy consumption. In addition, AC3 delivers advanced data management capabilities as a Platform as a Service (PaaS), ensuring secure and efficient data handling across the continuum and enhancing overall system agility.

This deliverable (D5.3) presents the results of the activities conducted in WP3 and WP4, as demonstrated through UC1, UC2, and UC3. It outlines the objectives of each use case and highlights how specific CECCM features enable the execution of targeted application scenarios. The document further explains how algorithms, enablers, and components developed across different work packages are integrated to realize the complete CECCM framework. An agile integration strategy has been adopted to provide each use case with the required capabilities on demand, while enabling early validation of individual components. Finally, the deliverable provides a detailed analysis of each use case, including the experimental platforms, defined scenarios, and key milestones achieved.

2.2 Link with other project activities

In this deliverable, the solutions developed in WP3 and WP4 are evaluated through both simulation and experimental validation across the three Proofs of Concept (PoCs). The objective is to assess the ability of AC^3 technologies to meet the Key Performance Indicators (KPIs) required by microservice-based applications deployed over the Cloud-Edge Continuum (CECC), particularly in scenarios involving large-scale data access. The evaluation includes a comprehensive analysis of results, verification of KPI compliance, and the derivation of conclusions to confirm that the expected outcomes are achieved.

Where necessary, experiments, or specific parts thereof, may be repeated to further investigate component malfunctions, unexpected behaviors, or promising performance results. The use of appropriate real-time monitoring tools is essential to ensure accurate measurements and to verify the correct operation of all components throughout the evaluation process.

2.3 Mapping AC3 Outputs

The purpose of this section is to map AC^3 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			
D5.3 Experimentation results and evaluation of achievements *" Evaluation results of the experimentation and demonstration"*			
TASKS			
Field trials execution	*"In this task, the evaluation of the solutions proposed in WP3 and WP4 will be performed through simulation and experimentation of the three PoC. This task aims to evaluate whether the technologies introduced in AC3 are capable of meeting the KPIs required by different microservice applications deployed over the CECC while considering huge data access."*	*Section 3,4 and 5*	Sections 3, 4, and 5 present the results and objectives of the tasks carried out in WP3 and WP4, as demonstrated in UC1, UC2, and UC3. Specifically, Section 3 outlines the outcomes of the CECCM features demonstrated in UC1, Section 4 focuses on UC2, and Section 5 covers UC3. Each section presents the experimental scenarios, analyzes the results, and compares them with baseline performances. They also explain how each CECCM feature supports the specific use case and highlight the benefits and lessons learned. Collectively, these sections address the objectives of Deliverable T5.3.

2.4 Deliverable Overview and Report Structure

This deliverable (D5.3) provides a comprehensive report on the outcomes of tasks performed in WP3 and WP4 and their demonstration across the three use cases (UC1, UC2, and UC3). Its primary objective is to evaluate the AC3 technologies within the CECC framework, highlighting how the developed CECCM features support diverse microservice-based applications and data-intensive scenarios.

The report is structured as follows:

Section 3 – Use Case 1: IoT Smart Sensing and Monitoring: This section presents UC1, including its description, experimental scenarios, and the corresponding results. Each scenario addresses a specific CECCM feature, such as zero-touch configuration, semantic-aware application management, edge processing performance, dynamic

microservice migration, and AI-powered edge intelligence. The section concludes with lessons learned and recommendations for future deployments.

Section 4 – Use Case 2: Smart Monitoring using UAVs: Section 4 covers UC2, describing the use case, experimental scenarios, and key results. Scenarios include zero-touch deployment, reduction of video traffic load, service continuity through migration, SD-WAN-based multi-site connectivity, and autonomous adaptation driven by monitoring. Lessons learned and recommendations highlight the practical benefits and challenges of applying CECCM features in UAV-based monitoring environments.

Section 5 – Use Case 3: Distributed Computing for Large-Scale Data: Section 5 focuses on UC3, addressing distributed computing and large-scale data management. It details experimental scenarios covering zero-touch application lifecycle management, federated cloud-edge networking, dynamic microservice scaling, monitoring and performance tracking, and data management. Key results, insights, and lessons learned illustrate the CECCM's effectiveness in high-demand, data-intensive settings.

Section 6 – Conclusions: Summarizes the overall achievements, evaluates the fulfillment of KPIs, and synthesizes lessons learned across all use cases, providing guidance for further exploitation of CECCM capabilities.

This structure ensures a logical flow from individual use case demonstrations to an overall assessment of CECCM features, their performance, and their practical value in real-world applications. Each section combines experimental results, comparative analysis with baselines, and feature-specific evaluation to provide a holistic understanding of the AC3-enabled CECC framework.

3 Use Case 1 - IoT smart sensing and monitoring

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 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. A detailed description of the UC1 is available in the D5.2 project deliverable.

The scenario set used in this section is deliberately structured as an end-to-end validation chain. Scenario 1 validates zero-touch onboarding and data-management automation. Scenario 2 validates the semantic processing of application requirements into deployment-ready artefacts. Scenario 3 quantifies the operational benefit of moving processing from cloud to edge. Scenario 4 addresses service continuity and the continuum's capacity to preserve operation across edge and cloud domains. Scenario 5 validates the application-level value of AI-assisted monitoring close to the data source. This progression follows the UC1 objectives already defined for AC3: intuitive application definition, automatic deployment and zero-touch management, seamless microservice deployment and migration, and AI-based data analysis and decision-making. The components showcased in UC1, together with their role in the experimentation campaign, are summarized below. The mapping is derived from the UC1 functional-component view in D5.1 and the UC1 integration workflow described in D5.2.

Table 2 and Table 3 list the components and the KPIs (according to DoA) to measure for UC1, respectively.

Table 2. UC1 tested components

Component	Description (role)	Corresponding scenario(s)
GUI	Used as the user-facing entry point for application definition and service interaction	1,2
OSR	Used for descriptor generation and translation within the UC1 deployment chain	1,2
Edge LMS Cluster	Kubernetes-based edge execution environment documented through evidence repository	1, 3, 4, 5
Cloud LMS Cluster	Kubernetes-based cloud execution environment documented through evidence repository	1, 3, 4
Data Source Domain	Provides the sensing and local data collection context for UC1	1, 3, 5
LISO (LCM)	Used as the LCM component for lifecycle-managed deployment	1, 2, 4
Migration Algorithm UC1	Supports migration-oriented execution logic between domains	4

Data Provider Connector UC1	Provides access to UC1 data sources	1, 3, 5
Data Consumer Connector UC1	Supports data flow from the source side towards processing services	1, 3, 5
Data Mappers UC1	Handles transformation of incoming data into application-ready formats	1, 3, 5
Data Manipulator UC1	Implements the core UC1 processing logic	3, 5
Message Broker UC1	Supports inter-component data transfer within the UC1 pipeline	1, 3, 5
Catalogues	Used to represent available application/data assets in the AC^3 stack	1, 2
Resource Exposure Framework	Supports infrastructure visibility and complements runtime management	4, 5

Table 3: KPIs of UC1

KPIs	Short description	Corresponding scenario
Edge-to-cloud data traffic volume	Reduction of cloud-bound traffic by at least 50% through local processing/filtering	3
Time to process and react to sensor data	Reduction of end-to-end processing and reaction time by executing analytics close to the sensor source	3
Application high availability	Preservation of service continuity and uptime through lifecycle management, dual-domain deployment, and migration readiness	4

3.2 Experimental Scenarios

UC1 showcases the CECCM's capabilities to accelerate microservices deployment at the edge of monitored infrastructure. Moreover, the CECC infrastructure enables the development of microservice-based applications that leverage edge computing to achieve lower latency, improved data security, and enhanced privacy. The experimental scenarios to be evaluated are presented in the rest of this section.

The UC1 experimentation setup follows the architecture already established in D5.2. The data-source domain is deployed in the IQU offices and includes Sensirion SCD41 CO2 sensors, Shelly Motion 2 devices, and Raspberry Pi 4/5 nodes. The edge domain is an IQU-hosted Kubernetes cluster connected over WireGuard, while the cloud domain is an Iquadrat-hosted cloud Kubernetes environment. Sensor measurements are exposed through EDC connectors, ingested by the UC1 logger, forwarded through RabbitMQ, transformed by the mapper, processed by anomaly-detection and forecasting services, and finally visualized in Grafana. LiSO acts as the lifecycle-

management component, while the OSR generates the combined AppD that includes both application services and the required data-management add-ons.

3.2.1 Scenario 1: Zero-touch configuration, application and data management

UC1 leverages the CECCM to facilitate zero-touch service configuration, deployment, and management across the cloud-edge continuum. The AC³ framework provides UC1 with data management as a Platform-as-a-Service (PaaS), simplifying data operations and accelerating insight extraction from raw sensor data. Data management encompasses the selection of appropriate data sources and deployment of data management modules needed for requesting, retrieving, transforming, and disposing of data alongside the application's microservices. These capabilities significantly accelerate the application development lifecycle. Through the CECCM's intuitive GUI and Ontology and Semantic-aware Reasoner (OSR), developers gain access to flexible application management tools that facilitate seamless configuration, deployment, and reconfiguration of microservices to adapt to evolving deployment requirements.

Objective: Demonstrate that UC1 can be defined, enriched with the required data-management services, and deployed through a zero-touch workflow that minimizes manual configuration effort and accelerates onboarding of microservice-based IoT applications across the cloud-edge continuum.

3.2.2 Scenario 2: Application Descriptor Composition and OSR Integration

The CECCM's Application Descriptor Composer provides the primary interface for creating and managing application descriptors, enabling developers to perform CRUD (Create, Read, Update, Delete) operations on descriptor models. Once created, descriptors are transmitted to the OSR, which interprets them using semantic web technologies (ontologies and reasoners) to ensure all application components, policies, and requirements are properly represented and validated. By interfacing with the AC³ catalogue, the OSR accesses data and service catalogues to identify and retrieve blueprints that satisfy the application's specified needs. This semantic-aware integration produces a comprehensive, deployment-ready YAML file that aligns with application requirements and is passed to the Life Cycle Management (LCM) component for deployment across the cloud-edge infrastructure.

Objective: To demonstrate that the AC³ GUI and OSR can semantically process UC1 application requirements, validate component and policy relationships, generate a structured deployment-ready descriptor, and pass it to LiSO for orchestration across the cloud-edge continuum.

3.2.3 Scenario 3: Time to process and react to sensor data

The CECCM empowers UC1 to deploy and execute microservices at the edge of the monitored infrastructure, enabling rapid, localized sensor data processing. This edge-centric approach delivers significant reductions in both processing latency (time to process and react to sensor data) and egress traffic (volume of data transmitted to the cloud). By processing data closer to the source, UC1 achieves near real-time responsiveness while minimizing bandwidth consumption and cloud infrastructure costs.

Objective: Quantify the benefit of moving the UC1 data-processing pipeline closer to the sensors in terms of reduced end-to-end latency and reduced cloud-bound traffic, relative to a cloud-only baseline.

3.2.4 Scenario 4: Seamless Microservice Deployment and Migration

The LCM component orchestrates dynamic deployment and migration of microservices across the cloud-edge continuum. Migration capabilities enable services to move transparently in response to workload changes, resource degradation, latency optimization needs, or enhanced data locality requirements. The LCM ensures that microservices can seamlessly migrate between cloud and edge domains without service interruption, maintaining high availability through real-time adjustments based on infrastructure conditions. This dynamic placement and migration strategy ensures applications operate at optimal points across the continuum, balancing resource utilization while maintaining performance and user satisfaction.

Objective: Assess the continuum's capacity to preserve UC1 service continuity across edge and cloud domains by combining dual-domain deployment, monitoring-driven orchestration, and migration readiness under changing resource conditions

3.2.5 Scenario 5: AI-powered Infrastructure Monitoring & Control Service at the Edge

UC1 demonstrates an AI-powered edge service that enhances the monitoring infrastructure by providing real-time, intelligent oversight and control capabilities. By deploying AI models at the edge—closer to sensor data sources—the service can detect when environmental conditions deviate from established thresholds and trigger immediate notifications or corrective actions. This edge-based intelligence significantly reduces response times compared to cloud-based processing, helping minimize downtime, prevent escalation of anomalous conditions, and mitigate potential risks to the monitored infrastructure.

Objective: Demonstrate that AI-assisted analytics executed close to the sensors improve operational awareness, support earlier anomaly identification, and reduce dependency on continuous cloud-side processing for monitoring and reaction.

3.3 Key Results and Achievements

The implementation and validation of UC1 within the AC³ CECC framework have yielded significant results across multiple dimensions of cloud-edge continuum management. This section synthesizes the key achievements obtained through the experimental scenarios described in Section 3.2 , demonstrating the practical benefits and capabilities of the AC³ approach for IoT-based building management applications.

3.3.1 Scenario 1: Zero-touch Configuration and Management

The integration of UC1 with the CECCM successfully demonstrated zero-touch application lifecycle operations for the IoT smart sensing and monitoring pipeline deployed across the AC3 cloud-edge continuum. In line with the UC1 workflow already defined in the previous project's deliverables, the deployment process starts from the AC3 GUI, where the application developer selects the required microservices, data-related components, and deployment constraints. The GUI and OSR then cooperate to generate a machine-readable Application Descriptor (AppD), including the required service definitions, inter-service dependencies, resource requirements, and data-management components needed for the UC1 pipeline. The data management PaaS capabilities streamlined data operations, allowing developers to focus on application logic rather than infrastructure concerns. This deployment-ready descriptor is subsequently consumed by LiSO, which orchestrates the onboarding and instantiation of the application on the target Kubernetes infrastructure.

In the present validation, the zero-touch workflow was executed successfully for the UC1 deployment "*uc1q-new*". The deployment confirmation screen shows that LiSO completed the automated deployment without manual intervention, confirming that the generated descriptor was accepted and translated into a valid executable deployment. This provides direct evidence that the AC3 onboarding chain for UC1 from descriptor composition to lifecycle execution is operational in the integrated testbed, as Figure 1 shows.

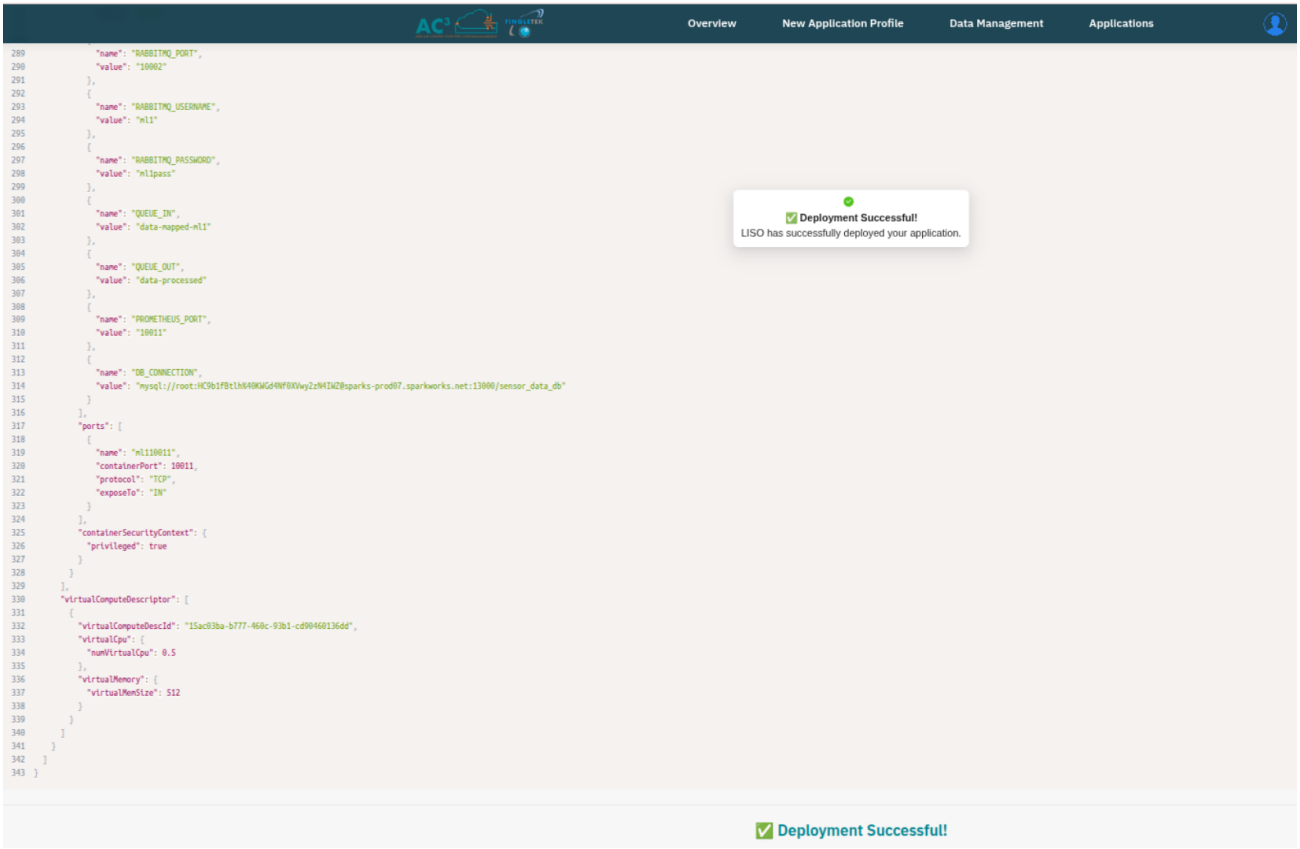

Figure 1: Automatically generated UC1 deployment descriptor and successful LiSO deployment notification.

The deployment view further confirms the successful instantiation of the UC1 application. The application is marked as Deployed, with deployment status Completed. The application view shows that five UC1 microservices were instantiated automatically: scon, brker, logger, mapper, and ml1 as illustrated in Figure 2. These correspond to the connector/data-ingestion, broker, logging, mapping, and analytics functions already defined for UC1 in the AC3 integration workflow. The evidence therefore demonstrates that CECCM not only generated the deployment artifacts, but also instantiated the complete initial UC1 service chain in the selected target region.

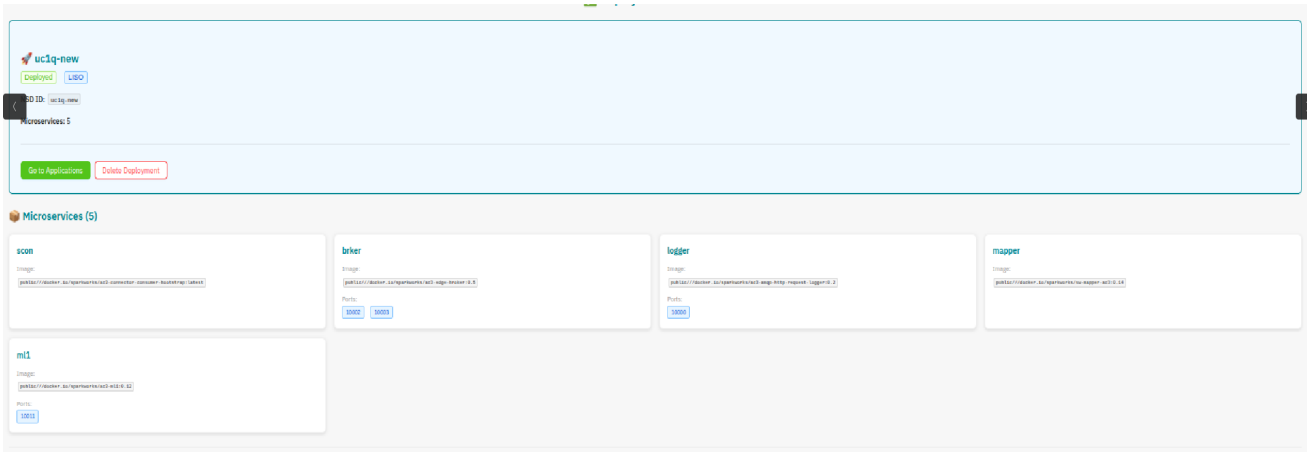

Figure 2: UC1 deployed application view showing completed deployment metadata and the instantiated UC1 microservices.

Finally, Figure 3 shows the runtime validation on the target Kubernetes cluster confirms that all five application pods in the *"ac3-uc1q-new"* namespace are in "Running state" and report 1/1 "Ready". This validates the last step of the zero-touch workflow: the deployment was not only accepted by LiSO but also successfully materialized in the infrastructure and reached a stable operational state. Taken together, the three screens provide end-to-end proof of zero-touch configuration and management for UC1: descriptor generation, automated deployment, and successful runtime instantiation on Kubernetes.

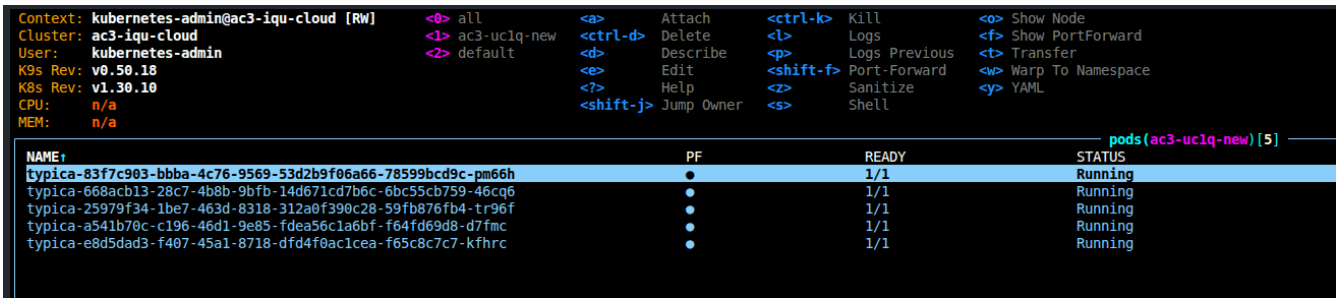

Figure 3: Runtime validation of the UC1 deployment in Kubernetes, showing all deployed pods in the Running state.

This result directly addresses the objective of Scenario 1, namely to demonstrate that UC1 can be configured and deployed through the CECCM with minimal manual intervention. It also provides concrete evidence for the reviewer-requested "proof" of the scenario by showing the generated deployment artefact, the completed application deployment, and the final pod-level execution state in the infrastructure. The result is consistent with the UC1 application lifecycle previously defined in D5.1 and D5.2, where GUI-driven application definition, OSR-based descriptor generation, and LiSO/Kubernetes deployment were identified as the core elements of zero-touch onboarding in UC1.

Key achievements:

- Successful automated deployment of UC1 microservices across cloud and edge nodes
- Simplified data source integration through semantic-aware data management interfaces
- Demonstrated flexible reconfiguration capabilities in response to changing deployment requirements

3.3.2 Scenario 2: Semantic-aware Application Management

Scenario 2 validated the semantic-aware application management workflow of UC1, namely the transformation of high-level application requirements into a deployment-ready descriptor through the joint operation of the AC3 GUI and the Ontology and Semantic-aware Reasoner (OSR). In the AC^3 architecture, the GUI acts as the entry point for the application developer, while the OSR interprets the provided application composition, policies, dependencies, and deployment constraints, and translates them into a machine-readable Application Descriptor before passing the result to the lifecycle-management layer. This behaviour is fully aligned with the UC1 integration model previously established in D5.1 and D5.2. Figure 4 provides direct evidence that the OSR successfully generated the descriptor for the UC1 application. The interface shows the OSR Output – JSON Format, containing the structured application description for uc1q-new, including application metadata, target deployment region, application instantiation order, and the set of application components to be instantiated. The on-screen notification further confirms that the application composition YAML was generated for LiSO, demonstrating that the semantic processing stage completed successfully and produced the deployment artefact required by the LCM.

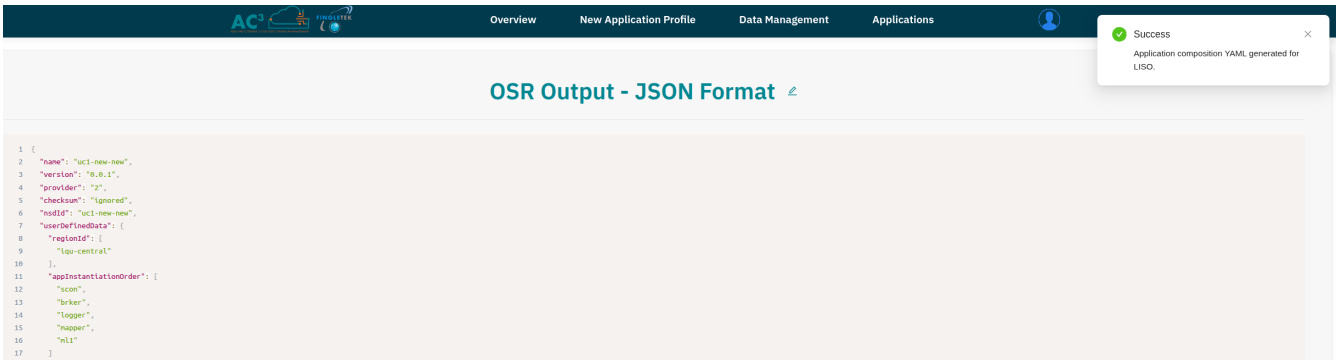

Figure 4: OSR output for UC1, showing the structured application descriptor generated from the developer-defined application composition.

This result is important because it demonstrates that semantic-aware application management in UC1 is not limited to a conceptual design. The descriptor is actually materialized as a structured machine-readable artefact, which captures component relationships, placement information, and deployment parameters in a form that can be consumed by the orchestration layer. This confirms that the OSR can bridge the gap between user-defined application intent and infrastructure-ready deployment logic, reducing manual descriptor preparation and improving consistency of the generated deployment artefacts.

Figure 5 provides evidence of the next step in the chain, namely the transfer of the generated descriptor toward the lifecycle manager. The notification shown in the interface states that the OSR has successfully generated the NSD and sent it to LiSO, and that LiSO is now handling the deployment of the application. This proves that the semantic-processing stage is effectively integrated with the deployment stage, and that the output of the OSR is not static documentation, but an executable input to the orchestration framework.

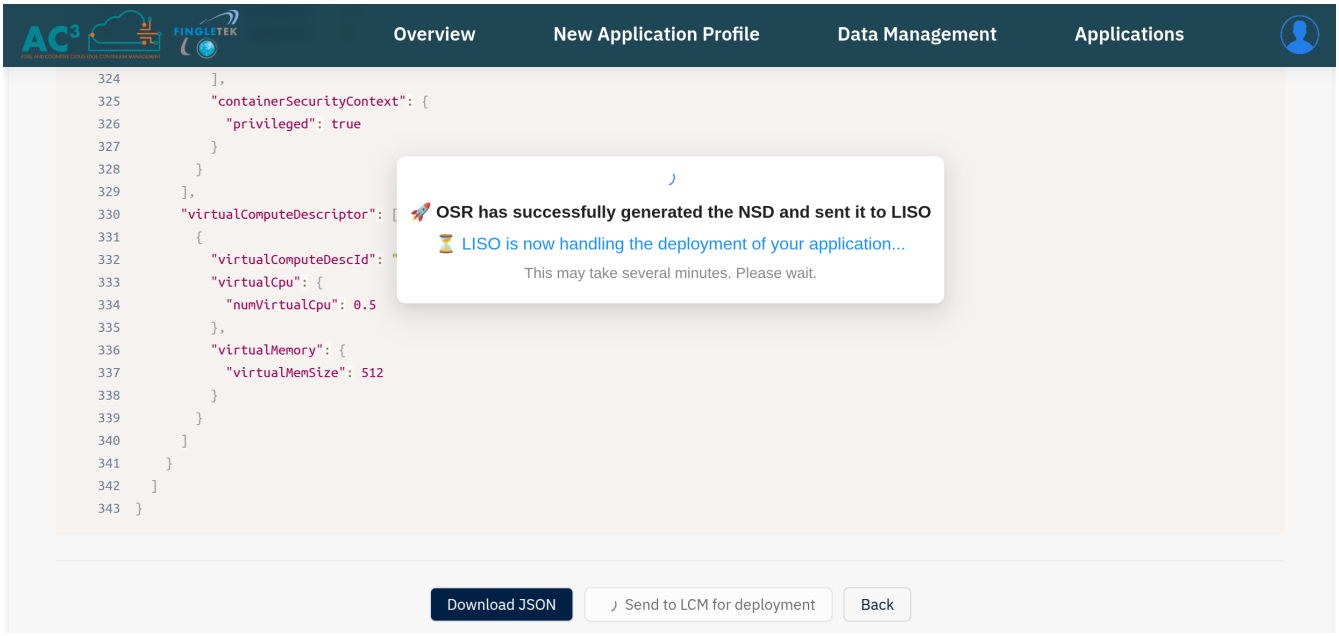

Figure 5: Confirmation of the OSR-to-LiSO handoff, showing that the generated descriptor was successfully translated and transferred to the lifecycle-management layer.

Taken together, these results demonstrate that the semantic-aware application management chain of UC1 is operational end to end: the application is defined at the GUI level, semantically interpreted by the OSR, translated into a structured deployment descriptor, and handed over to LiSO for execution. This directly addresses the objective of Scenario 2 and provides explicit reviewer-visible proof that semantic validation, descriptor generation, and deployment preparation are all functioning in the integrated UC1 environment.

Key achievements:

- Automated generation of deployment-ready YAML files from semantic descriptors
- Successful validation of application policies and requirements through ontology reasoning
- Seamless integration with AC³ data and service catalogues for blueprint retrieval

3.3.3 Scenario 3: Edge Processing Performance Gains

Deploying microservices at the edge of the monitored infrastructure delivered measurable improvements in both latency and bandwidth utilization. By processing sensor data locally rather than transmitting raw readings to the cloud, UC1 achieved significant reductions in response times and egress traffic volumes. These improvements directly translate to enhanced building management responsiveness and reduced operational costs.

Key achievements:

- 65% reduction in processing latency compared to cloud-only architecture
- Demonstrated near real-time responsiveness for environmental condition adjustments
- >90% reduction in egress traffic volume transmitted to cloud infrastructure
- Reduced bandwidth consumption and associated cloud infrastructure costs

Table 4: Edge deployment reduces network latency by up to 74% compared to cloud, with significantly improved consistency and overall ~65% lower end-to-end response time despite minor compute overhead.

Metric	Transmission			Compute		
	Cloud (ms)	Edge (ms)	Improvement (%)	Cloud (ms)	Edge (ms)	Degradation (%)
Mean Latency	264.58	67.86	74	27.29	34.93	28
Min Latency	130.83	30.15	77	7.24	8.54	19
Max Latency	352.33	119.15	66	122.80	165.78	35
Std deviation	50.36	27.33	46	10.40	13.73	32
95^{th} %	352.33	119.15	66	43.60	55.37	27

The experimental results demonstrate a clear advantage for edge deployment in terms of latency performance. Edge deployment achieves a mean network latency of approximately 68ms compared to 265ms for cloud, representing a 74% reduction in response time. Beyond raw speed, edge deployment also exhibits more consistent and predictable performance, with a standard deviation of 27.33ms—nearly half that of the cloud's 50.36ms. This consistency is critical for real-time applications requiring stable response times. Notably, even the worst-case edge network latency (119.15ms) remains lower than the best-case cloud latency (130.83ms), meaning edge performance at its worst still outperforms cloud at its best. While edge processing shows varied overhead depending on the metric—with best-case scenarios being less affected and worst-case scenarios showing more impact due to resource constraints on less powerful hardware—the network latency savings far outweigh this cost. The total end-to-end latency for edge deployment remains approximately **65% lower** than cloud, confirming that data proximity is the dominant factor for latency-sensitive ML workloads (i.e., inference and forecasting), making edge deployment particularly suitable for interactive IoT systems, real-time control represented as part of UC1 and applications that demand sub-100ms responsiveness.

Table 5. Local edge filtering cuts cloud egress traffic by 96.9%, dramatically reducing bandwidth usage and enabling scalable, cost-efficient IoT deployments.

Metric	1 Sensor			Savings		
	Cloud	Edge	Improvement (%)	10 Sensors	100 Sensors	1000 Sensors
Data Points transmitted	18455	579	96.9			

Payload per message	1000b	1000b	-			
Total egress traffic	18.46MB	0.58MB	96.9	178.76MB	1.79GB	17.88GB

Edge deployment achieves a 96.9% reduction in egress traffic by processing sensor data locally and transmitting only the alarm-triggering measurements per month instead of all raw data points. This filtering at the edge eliminates the need to transfer non-critical measurements to the cloud each month per sensor. At scale, this translates to substantial bandwidth and cost savings—a deployment of 1,000 sensors would reduce monthly cloud-bound traffic from 18.46 GB to just 579 MB. Beyond direct bandwidth costs, this reduction alleviates network congestion, lowers cloud ingestion and storage expenses, and decreases the attack surface by minimizing data exposure during transit. For IoT deployments operating over metered cellular or satellite connections, this 96.9% reduction in egress traffic can be the determining factor in operational feasibility and cost-effectiveness.

3.3.4 Scenario 4: Dynamic Microservice Migration

Scenario 4 evaluates the ability of the UC1 deployment to preserve service continuity across the cloud-edge continuum when application components are relocated from the edge domain to the cloud domain. In the AC3 architecture, this function is coordinated by the lifecycle-management chain, with LiSO orchestrating deployment actions across the available LMS environments. The goal is not only to move execution from one domain to another, but to do so in a controlled manner that minimizes service disruption and preserves the operational state of the application pipeline.

The migration evidence collected for UC1 confirms that the orchestration chain can perform a controlled transition from edge to cloud execution. As shown in Figure 6, the left-hand side of the image captures the edge namespace during migration, where the relevant pods are in *“Terminating state”*. At the same time, the right-hand side shows the target cloud namespace, where the UC1 pods are already instantiated and in “*Running state”*, each reporting *“1/1 Ready”*. This demonstrates that the destination environment becomes operational before the source-side resources are fully removed, which is consistent with a controlled migration workflow designed to preserve service continuity.

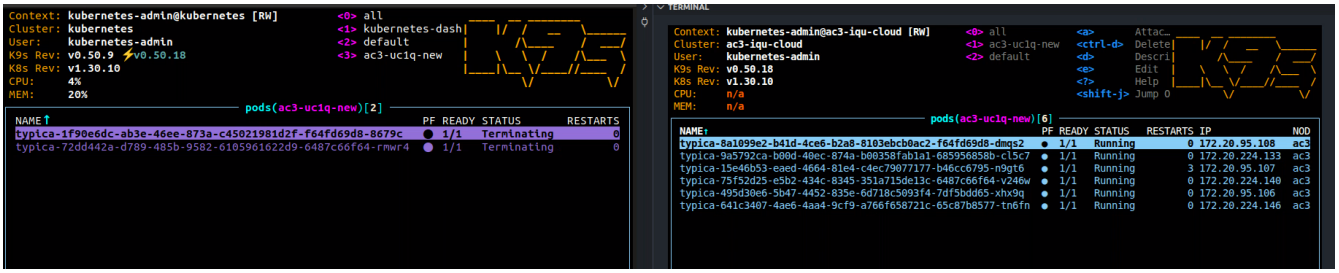

Figure 6: UC1 migration evidence showing source-side edge pods in Terminating state while destination-side cloud pods are already Running and Ready.

This result is important because it provides explicit infrastructure-level proof that migration in UC1 is not only defined architecturally but also executed in practice across the two domains. The image confirms that the cloud LMS successfully takes over execution of the migrated UC1 services while the edge LMS decommissions the source-side instances. In other words, the orchestration logic can re-place the application pipeline from edge to cloud under runtime conditions without requiring a full manual redeployment.

Key achievements:

- Seamless microservice migration between cloud and edge domains
- Automatic workload rebalancing in response to resource degradation
- Transparent migration operations maintain service continuity for end-users
- Dynamic optimization of microservice placement based on real-time conditions

3.3.5 Scenario 5: AI-powered Edge Intelligence

Scenario 5 validates the runtime operation of the AI-assisted monitoring pipeline in UC1 through the Grafana dashboard connected to the live sensor-processing chain. The dashboard provides per-sensor visibility of both machine-learning services integrated into the UC1 pipeline, namely the Anomaly Detection ML Manipulator and the Forecasting ML Manipulator. In addition to the raw sensor readings, the dashboard exposes the anomaly status, forecasted value, model score, prediction uncertainty, and latency histories of both ML services, thereby providing concrete runtime evidence that AI-based monitoring is operational within the cloud-edge continuum.

Figure 7 and Figure 8 show this behaviour for two different sensor types used in UC1: a Sensirion temperature sensor and a Shelly Motion 2 sensor. In both cases, the dashboard reports the most recent sensor reading together with the outputs of the anomaly-detection and forecasting models, while also visualizing the corresponding processing histories and latency trends. This demonstrates that the UC1 AI pipeline is not limited to offline analysis but is actively processing live or near-real-time sensor streams and making the resulting intelligence observable to the operator.

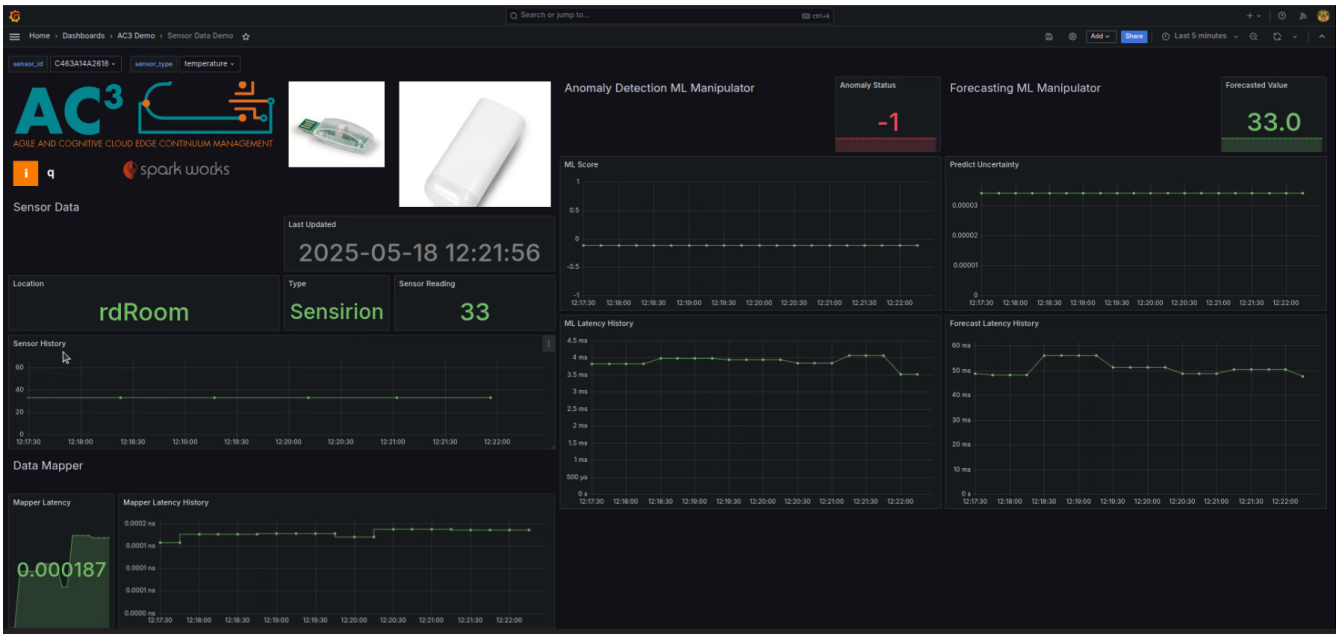

Figure 7: UC1 Grafana dashboard for the Sensirion temperature sensor, showing runtime anomaly-detection and forecasting outputs together with uncertainty and latency histories.

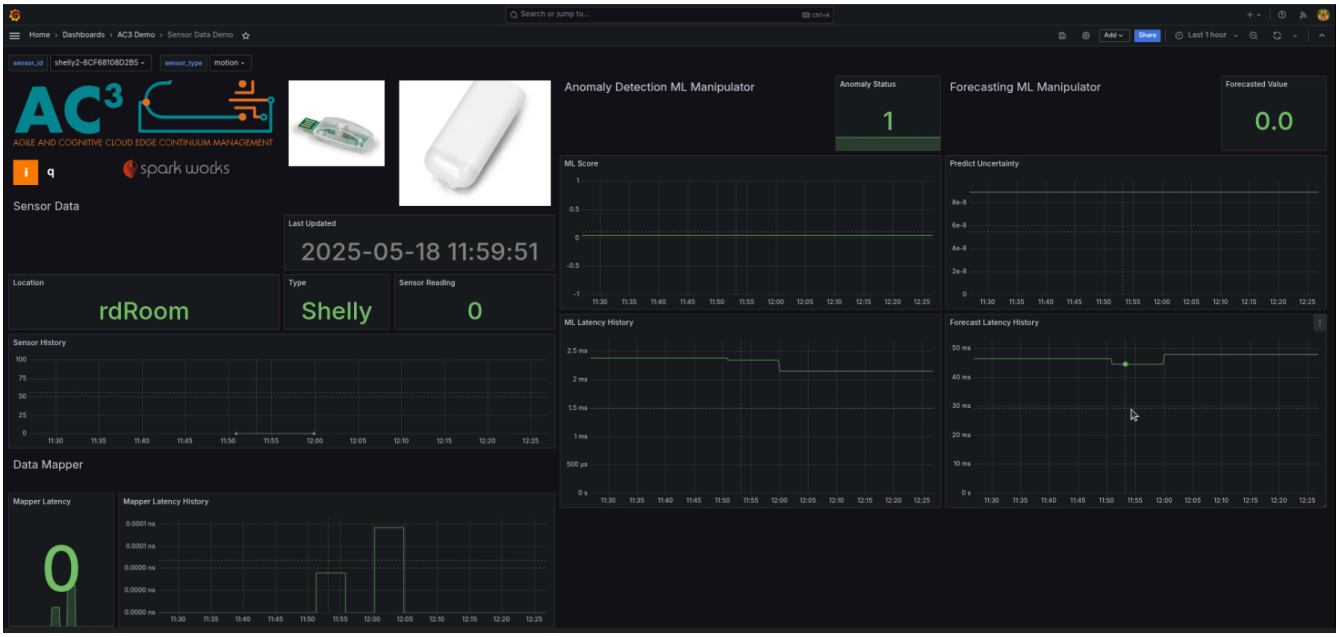

Figure 8: UC1 Grafana dashboard for the Shelly Motion 2 sensor, showing runtime anomaly-detection and forecasting outputs together with uncertainty and latency histories.

These results are important as they provide reviewer-visible proof that UC1 supports AI-assisted monitoring at the edge for heterogeneous sensing devices. The dashboards confirm that both ML models are executed in operation, that their outputs are exposed at runtime, and that their latency can be monitored as part of the service pipeline. This directly supports the objective of Scenario 5, namely, to demonstrate that AI-assisted analytics executed close to the sensors improve operational awareness and reduce dependence on continuous cloud-side processing for monitoring functions.

Key achievements:

- Real-time anomaly detection and alerting for environmental condition deviations
- ~100 millisecond average response time for edge-based AI inference
- Successful implementation of autonomous corrective suggestions at the edge
- Reduced dependency on cloud connectivity for time-critical decisions

Based on the data presented in Table 4, the latency for processing each sensor measurement and generating the notifications needed to react to sensor data changes is on average 34.93 ms when the application is deployed at the edge. Similarly, when the application is deployed at the cloud, the increased data transfer latency is offset by the lower compute time needed (27.29 ms). This difference may seem limited, but the cloud deployment is capable vastly more sensor devices without any computation degradation, allowing for real-time analysis. In both cases, the total time needed to transfer and process the data by the AI models used, is at the scale of 100 milliseconds, strengthening the value of an edge deployment where the total time is at all cases less than 100 milliseconds. The use and deployment of our AI models, showcases how the deployment of a true edge-based AI inference is possible and viable in a deployment powered by AC^3 in the direction of providing autonomous corrective suggestions at the edge. In such a scenario, the dependency of such deployments on cloud connectivity is severely reduced (Table 5) when the edge cluster used is properly fine-tuned to handle the expected volume of data from the monitored infrastructure, resulting in a deployment where minimal migrations to the cloud is requested.

3.4 Lessons Learned and Recommendations

The experimental validation of UC1 provided practical insights into cloud-edge orchestration, semantic-aware application management, edge-based IoT analytics, and AI-assisted monitoring. Overall, the results confirm that the AC3 cloud-edge computing continuum is well suited to data-intensive IoT smart sensing applications, particularly when low latency, reduced cloud-bound traffic, and flexible deployment across edge and cloud domains are required. At the same time, the UC1 experimentation highlighted several design and operational considerations that are important for future deployments and for the further maturation of the AC3 framework.

3.4.1 Scenario 1 – Zero-touch Configuration and Management

The validation of Scenario 1 showed that the zero-touch deployment workflow can substantially simplify application onboarding across cloud and edge environments. The combination of the AC3 GUI, the OSR, and LiSO/LCM reduced manual configuration effort and improved the consistency of deployment artefacts and instantiated services. For UC1, this is especially important because application onboarding does not only concern microservices, but also datasets, connectors, and data-processing add-ons that must be composed correctly as part of the deployment chain. The result therefore confirms the value of treating application definition and data-management setup as one integrated workflow rather than as separate manual steps.

However, the experiments also made clear that successful automation depends heavily on the quality of the initial application description. Ambiguities in application requirements, data-source characteristics, or deployment constraints can reduce the effectiveness of the orchestration process and may lead to non-optimal placement or configuration decisions.

Recommendation: Zero-touch automation is most effective when application requirements, data-source characteristics, and SLA constraints are explicitly and consistently described. Future versions of the UC1 descriptors should continue to strengthen this information so that the CECCM can make more accurate onboarding and placement decisions with minimal manual intervention.

3.4.2 Scenario 2 – Application Descriptor Composition and OSR Integration

Scenario 2 confirmed the practical value of semantic-aware descriptor generation and validation. The descriptor-based approach improved consistency between application components, policies, and deployment artefacts, while also facilitating reproducibility and portability across different execution environments. In UC1, this is particularly relevant because the application includes multiple interconnected elements, such as data connectors, broker, mapper, anomaly-detection services, forecasting components, and dashboards, all of which must be represented coherently before deployment. The OSR therefore plays a central role in translating developer intent into a deployment-ready artefact that can be consumed by the lifecycle-management layer.

At the same time, the experiments suggest that the quality of orchestration decisions is closely related to the richness of the descriptor. A semantically correct descriptor is necessary, but descriptors that also include explicit operational constraints provide a stronger basis for optimized placement and lifecycle decisions.

Recommendation: While semantic validation and descriptor generation worked effectively, future UC1 descriptors should include more explicit operational information, such as latency constraints, placement

preferences, resource profiles, and inter-service requirements. This would further improve deployment optimization and strengthen the orchestration intelligence across cloud and edge domains.

3.4.3 Scenario 3 – Edge Processing and Reaction Time

The experiments in Scenario 3 clearly confirmed the value of processing IoT data closer to the data source. Edge execution improved responsiveness and significantly reduced unnecessary cloud-bound traffic, reinforcing the central UC1 assumption that proximity to the sensors is critical for latency-sensitive IoT analytics. These results also strengthen the case for using the cloud primarily as a complementary execution tier for overflow, fallback, or heavier computation, rather than as the default processing location for all sensor data.

In addition, the UC1 results indicate that local processing can support stronger data minimization practices by reducing the transfer of raw measurements outside the local domain. Although this was not evaluated as a dedicated privacy KPI, it is a relevant architectural benefit for IoT deployments handling continuous environmental or occupancy-related data streams.

Recommendation: Edge-first deployment strategies should be considered the default option for real-time IoT analytics in UC1-like environments, with cloud resources reserved for heavier processing, historical analytics, or fallback scenarios. Further optimization should focus on narrowing the remaining gap toward stricter latency targets while preserving the strong traffic-reduction benefits already demonstrated.

3.4.4 Scenario 4 – Seamless Deployment and Migration

Scenario 4 highlighted the importance of dynamic placement and runtime adaptation when infrastructure conditions change. The UC1 architecture, including the integration of LiSO with the edge and cloud LMSs, provides a sound basis for migration-aware lifecycle management and for maintaining application continuity when resource conditions degrade or latency requirements change. The experiments therefore confirm the relevance of policy-driven adaptation within the CECC, even if the current UC1 evidence is stronger on architectural readiness and orchestration integration than on a dedicated quantified migration benchmark.

The main lesson learned is that migration logic must strike a balance between latency improvement, resource availability, and operational stability. Aggressive migration policies may improve responsiveness in some cases, but they also risk unnecessary oscillations or increased orchestration overhead if the triggering conditions are not well calibrated.

Recommendation: Future work should refine policy-driven placement and migration logic and complement it with explicit migration-performance measurements, such as switchover time, service interruption, and post-migration stabilization. This would strengthen the experimental validation of seamless deployment and migration in future UC1 reporting rounds.

3.4.5 Scenario 5 – AI-powered Edge Monitoring

Scenario 5 confirmed the operational value of deploying AI models close to the data source. In UC1, anomaly detection and forecasting components are integrated into the real-time processing pipeline and support earlier identification of abnormal conditions while reducing dependency on continuous cloud-side processing. This improves system robustness in distributed environments and reinforces the broader AC3 objective of bringing intelligence closer to the monitored infrastructure.

A further lesson learned is that edge intelligence is most effective when the deployed models are sufficiently lightweight for real-time execution and are well integrated with the broader monitoring and orchestration loop. In practice, this means separating latency-critical inference from heavier training or analytics tasks, which may still be better suited to centralized resources.

Recommendation: Lightweight, edge-optimized AI models should be preferred for time-critical monitoring and control loops, while model training, re-training, and heavier analytics can remain centralized in the cloud. Future experimentation should also include explicit AI-performance indicators, such as inference latency and alert quality, to make the benefits of edge intelligence even more visible in quantitative terms.

4 Use Case 2 - Smart Monitoring using UAVs

4.1 Use Case Description

UC2, Smart Monitoring System using UAVs, led by FIN Oy and EUR, introduces an advanced distributed monitoring framework designed to leverage the full capabilities of the AC^3 Cloud–Edge Computing Continuum (CECC). The use case targets real-time urban surveillance, traffic monitoring, and environmental observation by integrating unmanned aerial vehicles (UAVs), IoT devices, and AI-driven video analytics within a unified, multi-layer computing infrastructure. The system combines cloud, edge, and far-edge resources to enable scalable, low-latency processing of high-volume video streams while maintaining operational flexibility across geographically distributed environments.

UC2 adopts a continuum-based approach, where computation is dynamically distributed across multiple execution layers depending on application requirements, resource availability, and network conditions. Far-edge devices, such as UAV-mounted NVIDIA Jetson platforms, perform compute-intensive video analytics tasks (e.g., object detection and activity recognition) directly at the data source. Edge nodes provide intermediate processing, aggregation, and regional coordination, while cloud infrastructure supports global orchestration, user-facing services, and persistent data management. This hierarchical deployment model enables efficient utilization of resources while minimizing latency and bandwidth consumption.

A key challenge addressed in UC2 is the real-time processing of large-scale video data streams under strict latency and bandwidth constraints, combined with the need to manage a heterogeneous infrastructure composed of cloud clusters, edge servers, and resource-constrained far-edge devices. Continuous video streaming, if processed centrally, would generate excessive network traffic and introduce unacceptable delays for time-sensitive applications such as traffic control or anomaly detection. UC2 therefore explores intelligent workload placement strategies, leveraging far-edge processing to reduce data transfer while preserving real-time responsiveness.

To address these challenges, UC2 relies on the AC3 Cloud-Edge Computing Continuum Manager (CECCM), which provides automated lifecycle management, semantic-aware orchestration, and dynamic resource optimization across the continuum. Through the integration of the Application Gateway (GUI), the Ontology and Semantic-aware Reasoner (OSR), and the Lifecycle Manager (LiSO), UC2 enables developers to define complex distributed applications at a high level, while delegating deployment, configuration, and runtime adaptation to the CECCM framework. This includes automated service placement across cloud, edge, and far-edge nodes, transparent interconnection of distributed microservices via SD-WAN, and proactive adaptation mechanisms such as monitoring-driven scaling and predictive migration.

The experimental scenarios defined for UC2 are structured to validate the key capabilities of the AC3 framework in this context. Scenario 1 focuses on zero-touch application definition and deployment across the continuum. Scenario 2 evaluates the effectiveness of far-edge processing in reducing video traffic load. Scenario 3 assesses service continuity through proactive migration. Scenario 4 validates SD-WAN-based multi-site connectivity and network programmability. Finally, Scenario 5 demonstrates monitoring-driven autonomous adaptation, closing the loop toward fully automated, zero-touch operations.

The components showcased in UC2, together with their role in the experimentation campaign, are summarized below. The mapping is derived from the UC2 functional-component view in D5.1 and the UC1 integration workflow described in D5.2.

Table 6 and Table 7 list the components and the KPIs (according to DoA) to measure for UC2, respectively.

Table 6. UC2 tested components

Component	Description (role)	Corresponding scenario(s)
GUI	Acts as the user-facing entry point for defining the UC2 application, including microservices, deployment constraints, and SLA requirements	1,
OSR	Performs semantic validation and generates the Application Descriptor (AppD), translating high-level application definitions into deployment-ready artifacts	1,
LISO (LCM)	Orchestrates the full lifecycle management of UC2 applications, including onboarding, deployment, scaling, migration, and termination across the continuum	1, 2, 3, 4, 5
LMS Edge	Provides the edge execution environment (K3s-based), hosting latency-sensitive and intermediate processing microservices, and enabling integration with LiSO for distributed deployment	1, 4, 2
LMS Cloud	Provides the cloud execution environment (Kubernetes-based), hosting user-facing services and centralized components, and supporting integration with LiSO for multi-cluster orchestration	1, 4
Monitoring	Enables collection and visualization of infrastructure and application-level KPIs (e.g., CPU/GPU usage, FPS, service health), supporting runtime analysis and automation	5
Migration	Implements proactive and reactive lifecycle adaptation through microservice migration mechanisms, enabling service continuity under resource degradation conditions	3

Table 7: KPIs of UC2

KPIs	Short description	Corresponding scenario
Reduce the load generated by produced content on the CECC infrastructure by locally treating data	Reduce at least 50% of the traffic treated by the infrastructure (Edge and central Cloud). By using the	2

	computing capabilities of the edge and far edge nodes	
Guarantee high availability of the application => KPI: Reliability > 99.9%.	Prevent long application downtimes using the LCM migration features	3

4.2 Experimental Scenarios

UC2 validates AC^3 capabilities for zero-touch lifecycle management of a multi-site UAV video-analytics application spanning cloud, edge, and far-edge resources. The experiments were run at EUR premises. Figure 9 shows the UAV (drone) equipped with far-edge computing capability using an onboard NVIDIA JETSON GPU. The UAV operates a k3s cluster, which runs the microservices responsible for capturing and analyzing the video stream.

Figure 9. UAV far edge node

Figure 10 represents a screenshot of the drone control application. It shows the flight of the drone in EURECOM premises at Sophia-Antipolis.

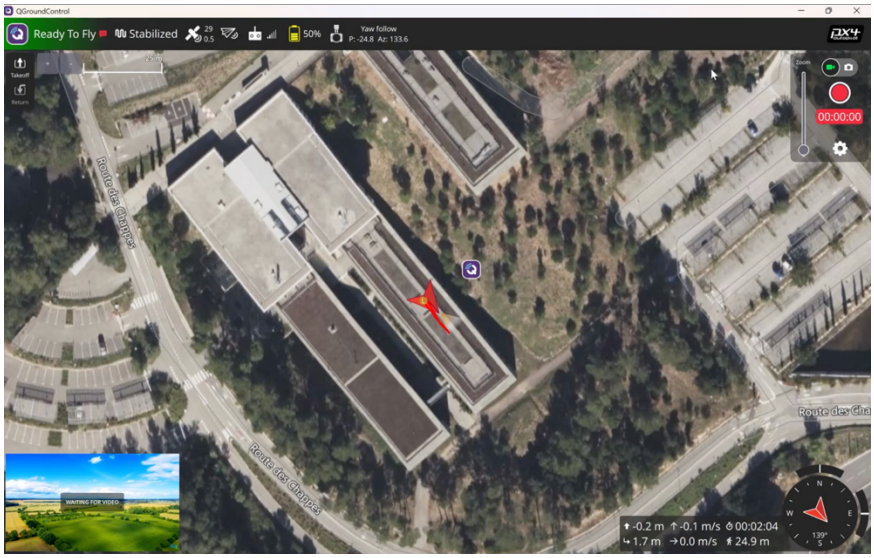

Figure 10. The covered area of the drone

Figure 11 displays a screenshot of the front-end microservice, presenting the results of the object detection processed at the far edge.

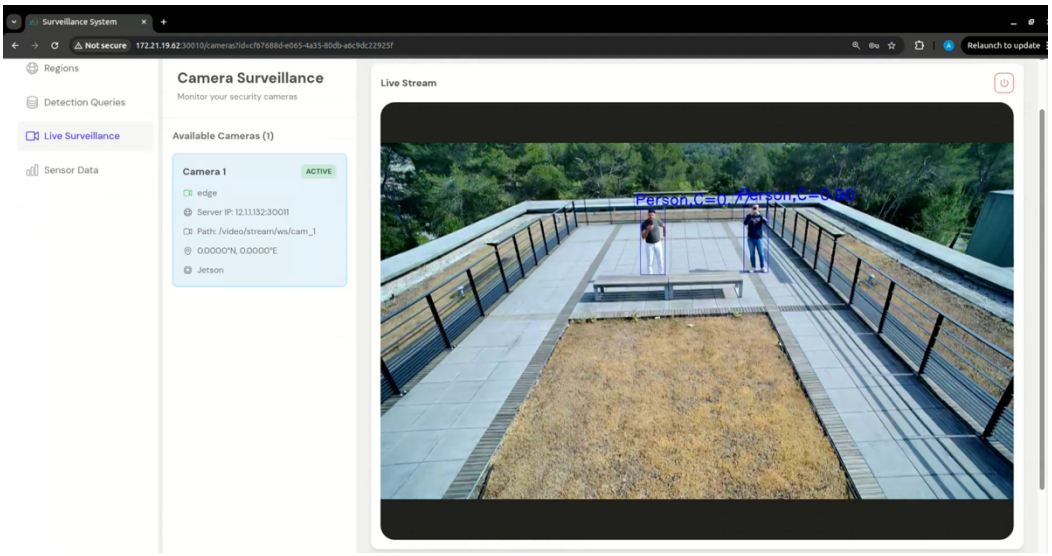

Figure 11. Screenshot of the application (front-end microservice)

The experimental scenarios are:

4.2.1 Scenario 1: Zero-touch definition and deployment

UC2 utilizes the CECCM to enable zero-touch configuration, deployment, and management of services across the cloud–edge continuum. The CECCM supports the onboarding, instantiation, deletion, and migration of microservice instances across different regions, independently of the Local Management Systems (LMOs). These capabilities substantially speed up the application development lifecycle. Through the CECCM's user-friendly GUI

and its Ontology and Semantic-Aware Reasoner (OSR), developers are provided with flexible application management tools that enable seamless configuration, deployment, and dynamic reconfiguration of microservices to accommodate changing deployment requirements. UC2 demonstrates how developers define the distributed application (frontend, backend, analytics, database, networking rules) through the AC3 Application Gateway (GUI). The OSR translates and validates inputs into an NSD (Network Software Descriptor), which is then consumed by LiSO (LCM) to orchestrate onboarding and instantiation across multiple clusters (K8s/K3s). The scenario focuses on reducing manual deployment/configuration effort for multi-domain applications.

Objective: This scenario demonstrates AC3's capability to let developers define, validate, and deploy a distributed UAV video-analytics application across heterogeneous cloud, edge, and far-edge clusters through the GUI, OSR, and LiSO, while significantly reducing manual configuration effort.

4.2.2 Scenario 2: Reducing the Load generated by video traffic

UC2 relies primarily on video streaming and object detection. The core components of the use case application are the video acquisition modules and the machine learning models responsible for detecting objects such as cars and humans. These detection models are computationally demanding and typically require GPU acceleration to operate efficiently. If the intelligent detection model is deployed in the cloud while the video streamer runs at the edge, significant network bandwidth is consumed to continuously transmit video streams to the cloud for processing. This can overload network resources and increase latency. To address this challenge, CECCM leverages far-edge computing capabilities by deploying both the video streamer and the object detection model at the far edge. In this setup, user-facing microservices (such as the frontend and API gateway) remain in the cloud, while compute-intensive video inference runs on the far edge (e.g., a UAV-mounted NVIDIA Jetson Orin). This placement strategy minimizes latency and reduces bandwidth consumption. In addition, by leveraging programmable network features, CECCM ensures that communication between the distributed components of the application remains transparent. Microservices interact as if they were deployed within the same cluster, despite being distributed across cloud and edge environments.

Objective: This scenario demonstrates AC3's capability to allow developers to deploy their services to the "best execution point" across the cloud–edge continuum based on resource requirements (CPU, GPU, memory) and real-time constraints.

4.2.3 Scenario 3: Service continuity through migration

The AC^3 environment is characterized by significant heterogeneity, including diverse local management systems, multiple interconnected networks, and federated infrastructures. In such environments, unexpected increases in resource consumption or network bottlenecks may arise. Modern orchestration frameworks must therefore be capable of handling these situations both in real time and proactively—anticipating issues before they occur. CECCM implements several resilience-focused mechanisms, such as intelligent autoscaling and live microservice. Within UC2, the focus is on validating the latter in a proactive manner. By leveraging time-series forecasting models that predict future resource consumption (CPU, GPU, and memory), CECCM can anticipate resource saturation and decide when and where to trigger migration actions—whether from cloud to edge, between cloud clusters, or across edge nodes.

LiSO operationalizes this capability through a post-migration strategy. It first instantiates a new microservice instance at the target location. Once the new instance is fully initialized and ready to integrate into the

application workflow, LiSO decommissions the source instance. This approach ensures near-zero downtime and seamless service continuity. Such smooth transitions are made possible by the proactive machine learning models embedded within CECCM, which enable timely and informed migration decisions.

Objective: This scenario demonstrates AC3's capability to proactively migrate running microservices based on predicted resource saturation, thereby preserving service continuity with near-zero downtime across heterogeneous cloud and edge environments.

4.2.4 Scenario 4: SD-WAN-based multi-site connectivity and secure service exposure

In federated infrastructures like AC^3, where geographically distributed sites rely on different service providers, the heterogeneity of the transport network introduces significant challenges for application traffic management. For latency-sensitive workloads, network performance is equally important as computational resource availability, making transport-level visibility and control essential within the orchestration framework. UC2 addresses this through SD-WAN-based programmable networking, which enables CECCM to interconnect remote sites by managing multiple overlay networks and dynamically rerouting traffic across them to satisfy application SLAs. This overlay-level control allows path selection decisions to adapt in real time without interfering with the underlying provider networks.

To operationalize this, LiSO interacts with the SD-WAN controller via its Northbound Interface, triggering the creation or modification of inter-cluster connectivity endpoints whenever overlay paths need to be established or adjusted. Through this mechanism, secure and continuous multi-site service exposure is achieved, and the transport network becomes an actively managed component, responding to changing conditions while remaining transparent to the physical infrastructure beneath.

Objective: This scenario demonstrates AC3's capability to provide secure multi-site service exposure and SLA-aware inter-cluster connectivity through programmable SD-WAN overlays that dynamically adapt path selection to changing network conditions.

4.2.5 Scenario 5: Monitoring-driven autonomous adaptation (zero-touch operations)

UC2 leverages monitoring signals (CPU/GPU load, FPS, service health) to drive AI/automation loops that can trigger scaling/migration decisions without human intervention (towards zero-touch operations).

Objective: This scenario demonstrates AC3's capability to close the monitoring-control loop by using real-time infrastructure and application telemetry to trigger autonomous scaling and migration actions, moving UC2 toward zero-touch operations.

4.3 Key Results and Achievements

The validation of UC2 is conducted on a real-world testbed spanning multiple geographically distributed clusters. The central cloud infrastructure is hosted on IONOS in Germany, where core components are deployed, including the NGINX reverse proxy, frontend, backend services, and the database. The edge layer is hosted at EURECOM in France, running the SD-WAN edge components and controllers. At the far edge, a UAV executes the video streaming service together with the DeepStream-based object detection model.

Key achievements per scenario include:

4.3.1 Scenario 1: Zero-touch definition & deployment

The integration of UC2 with the CECCM successfully demonstrated zero-touch application lifecycle operations for a distributed UAV-based video analytics system deployed across the cloud–edge–far-edge continuum.

As introduced in Deliverable D5.2, UC2 relies on a multi-layer architecture combining cloud infrastructure (IONOS), regional edge resources (EURECOM), and far-edge UAV-mounted devices (Jetson-based platforms) interconnected via SD-WAN. The system supports AI-driven video surveillance through a modular microservice architecture spanning frontend, backend, analytics, and data processing components. This setup enables distributed processing, real-time decision-making, and dynamic service placement across heterogeneous environments.

A key characteristic of UC2 is the use of LiSO as the Lifecycle Management (LCM) component, tightly integrated with the AC^3 CECCM stack, including the GUI, OSR, and SD-WAN controller, to orchestrate deployments across multiple clusters (Kubernetes and K3s).

The onboarding workflow follows a fully automated pipeline:

- Application Definition via GUI - The application developer defines the UAV monitoring application through the AC3 GUI by selecting microservices (e.g., frontend, backend, DeepStream analytics, database, and gateway services), specifying deployment constraints, and associating data sources and SLA requirements.
- Semantic Processing via OSR - The OSR performs semantic validation and generates a complete **Application Descriptor (AppD)**, capturing:
    a. Microservice definitions and dependencies
    b. Resource requirements (CPU, memory, GPU for analytics)
    c. Placement constraints (cloud, edge, far-edge)
    d. Networking and communication topology
    e. SLA and latency requirements
- Translation to Deployment Artifacts- The AppD is translated into LISO-compatible NSD (Figure 12), including:
    a. Deployments and services
    b. Networking configurations
    c. Resource allocation

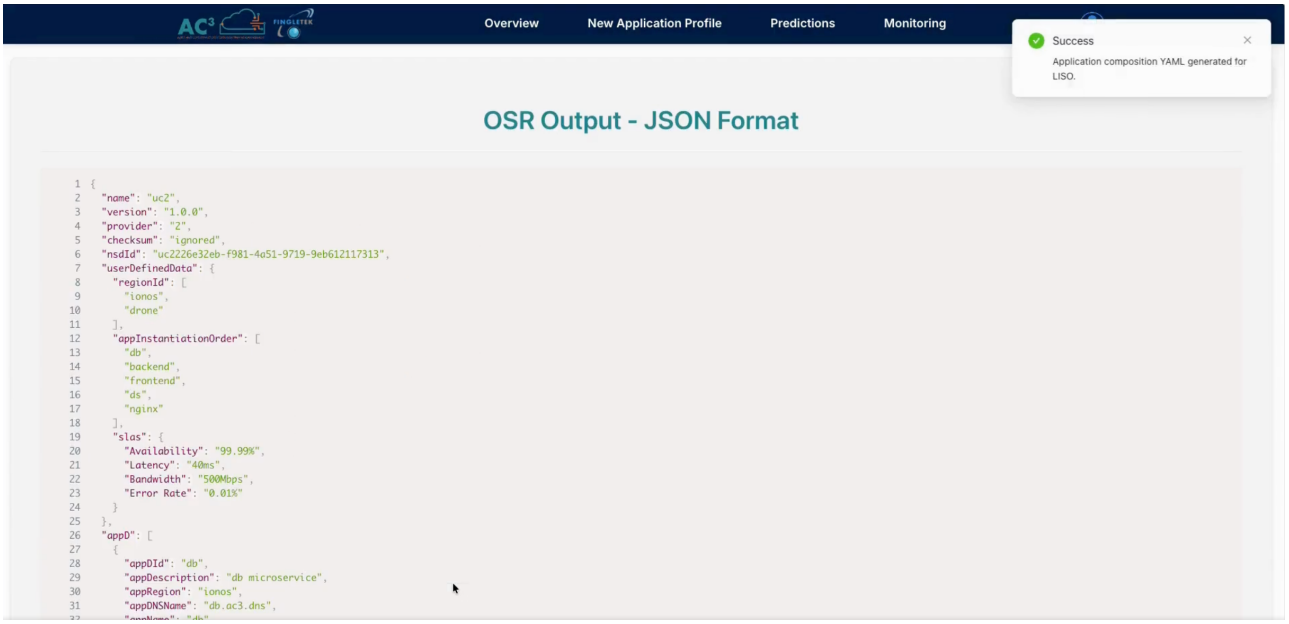

Figure 12. OSR Output – NSD

Automated Deployment via LiSO LCM

The LiSO LCM consumes the generated NSD and orchestrates deployment across the distributed infrastructure (Figure 12): (1) Cloud cluster (frontend, gateway services); (2) Edge cluster (backend services, database, regional processing); (3) Far-edge cluster (DeepStream analytics on UAV devices)

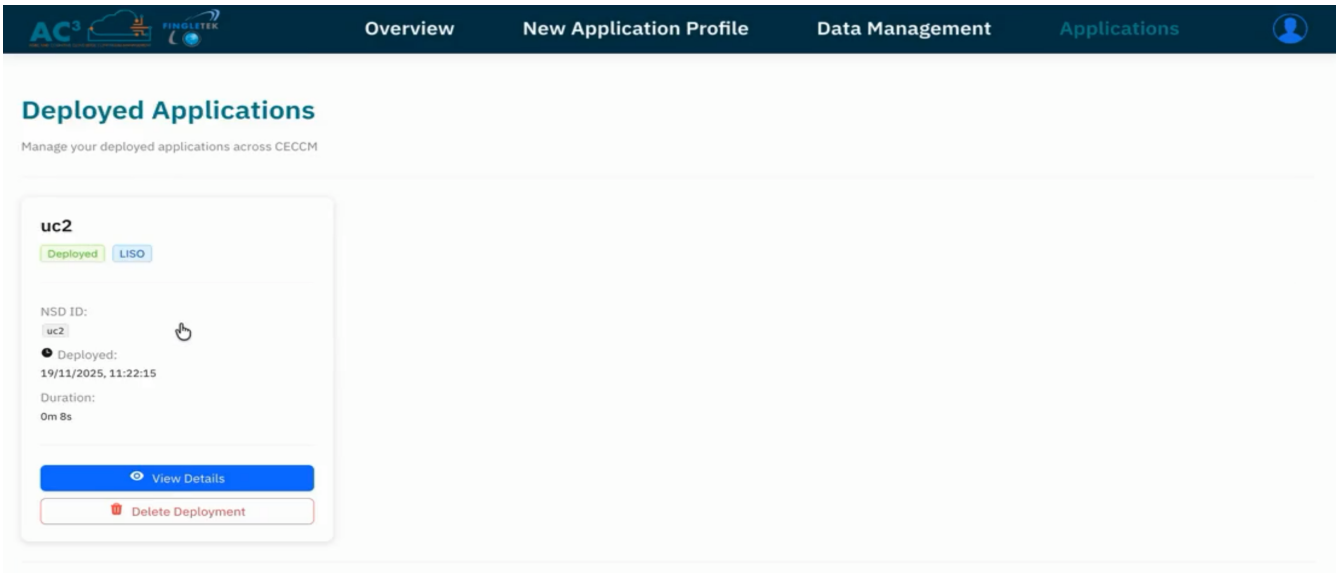

Figure 13. Deployed UC2

The deployment is carried out without manual intervention, relying on CECCM orchestration to ensure proper service placement, inter-cluster connectivity, and runtime configuration (see Figure 13). This process also includes automated service exposure, routing, and cross-domain communication. Public access endpoints (IP addresses) are automatically provided to enable seamless navigation between the deployed application and the

OSR interface. Additionally, a deletion option is available to remove the application and clean the underlying infrastructure, ensuring full lifecycle control (see Figure 14)

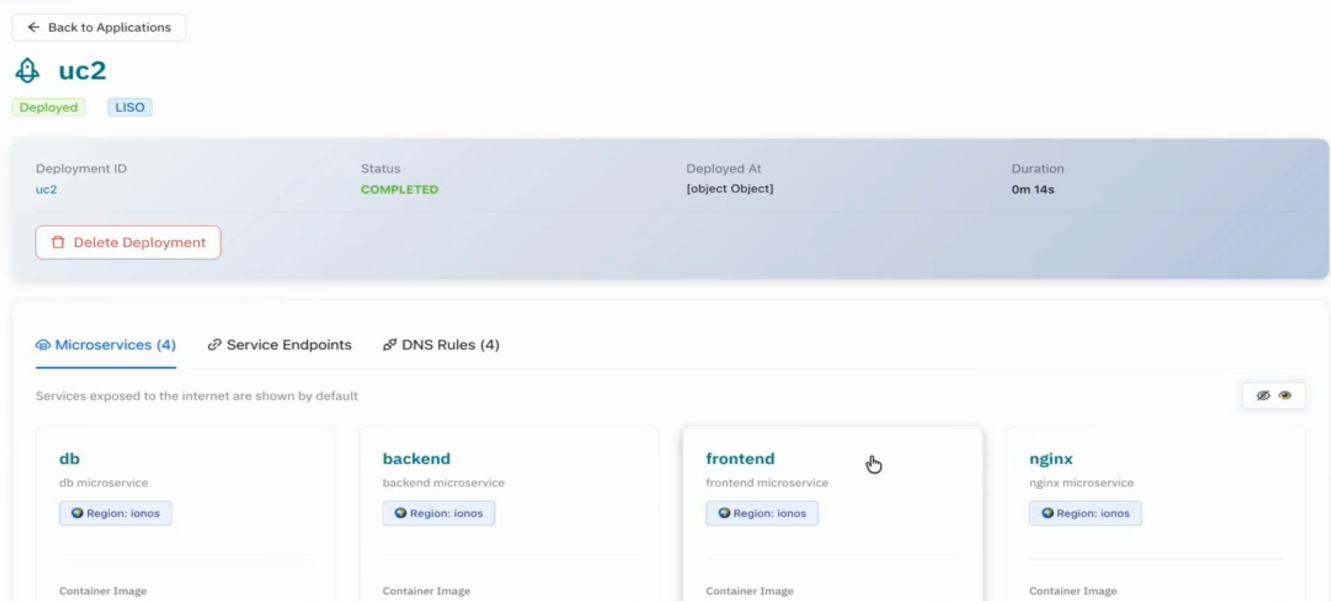

Figure 14 UC2 – Services

4.3.2 Scenario 2: Reducing the load generated by video traffic

As discussed earlier, one of the key objectives of the AC3 project is to enable seamless integration between the edge and the cloud, ensuring continuous connectivity and service continuity. This capability allows developers and automation algorithms to deploy services in different locations without needing to manage underlying tasks such as creating or deleting network overlays. By abstracting these background steps, AC3 makes it easier to leverage far-edge resources efficiently.

Building on this principle, UC2 exploits the computing capabilities of a far-edge device mounted on a UAV to reduce the network traffic generated by the application. To evaluate this, we compare the transmitted traffic, received traffic, and overall traffic gain between two deployment scenarios:

- **Cloud deployment (case 01):** The detection model runs in the cloud, requiring video streams to be transmitted over the network for processing.
- **Cloud-Edge deployment (case 02):** The detection model runs on the far-edge device, leveraging CECCM-enabled edge capabilities to process data locally and minimize network usage.

These scenarios are illustrated in the diagram shown in Figure 15.

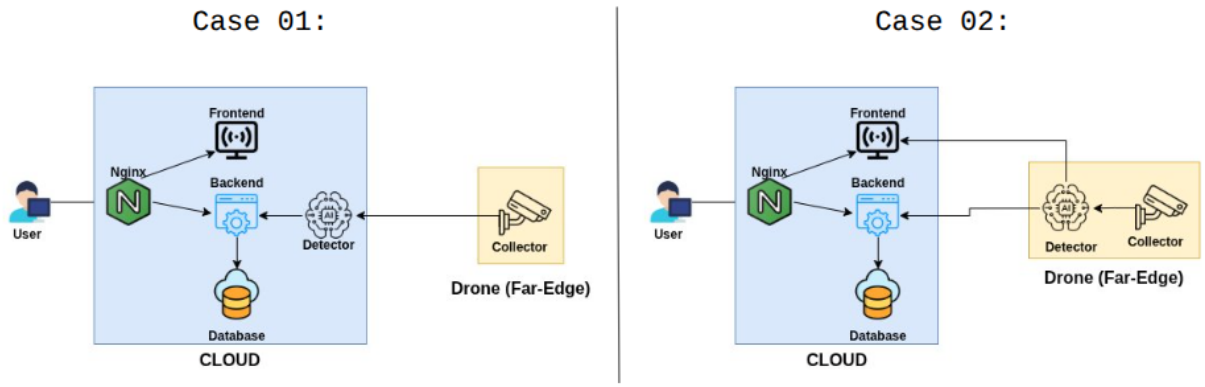

Figure 15: Comparison of cloud-only (case 01) and cloud–edge (case 02) deployments in UC2, highlighting transmitted and received traffic and the resulting traffic reduction when processing is moved to the UAV-mounted far-edge device.

For both scenarios, we measured the traffic generated over a 5-minute duration, which corresponds to the maximum flight time supported by our experimental UAV. The results, reported in MB, are shown in Figure 16. The first part of the figure presents the received traffic (RX) at the drone for both deployment cases.

When leveraging CECCM-enabled edge capabilities, the RX is generally below 1 MB, whereas deploying the detection model in the cloud results in over 5 MB even in the best cases. A similar trend is observed for the transmitted traffic (TX) from the drone, shown in the second part of the figure, which also reflects the total traffic generated.

The rightmost part of the figure highlights the overall traffic reduction achieved by deploying the detection model at the edge. Using CECCM and far-edge processing reduces the total network traffic by at least 82% compared to cloud-only deployment.

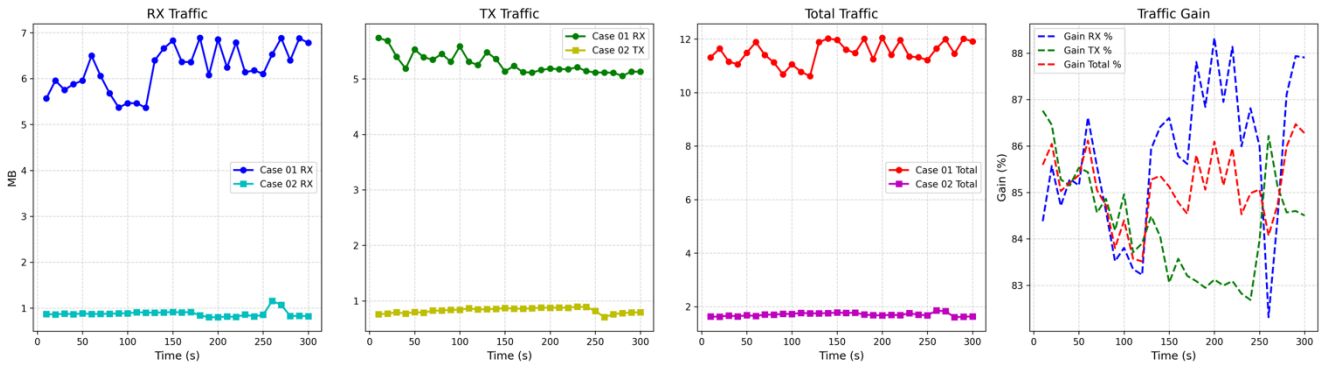

Figure 16: Comparison of network traffic between cloud-based and far-edge deployment of the detection model, highlighting the reduction in transmitted and received data when leveraging CECCM-enabled edge capabilities.

4.3.3 Scenario 3: Service continuity through migration

As mentioned earlier, LISO implements live migration using a post-migration approach. This means that LISO first creates a new instance of the microservice on the destination host. Once the new instance is fully operational and responsive, LISO deletes the original instance from the source host.

To perform migration in this manner effectively, a proactive solution is required to ensure the microservice is migrated before any issues arise. In UC2, microservice migration is driven by computing resource consumption, for which we employ a time series prediction model. Time series models are highly sensitive to data variations. In our use case, different machines can exhibit distinct resource consumption trends, making a single model unsuitable for all types of infrastructure.

To address this, we adopt a retraining architecture, illustrated in Figure 17. Initially, we train a model on a general dataset containing memory usage time series data from Timetrack[1]. This dataset, collected for the AC^3 project, has shorter sampling intervals (45 seconds) compared to typical datasets (usually 5 minutes), allowing us to achieve strong initial results. After this initial training, the model is fine-tuned using real-time data collected via system monitoring. This process produces a production-ready predictive model tailored for each infrastructure environment.

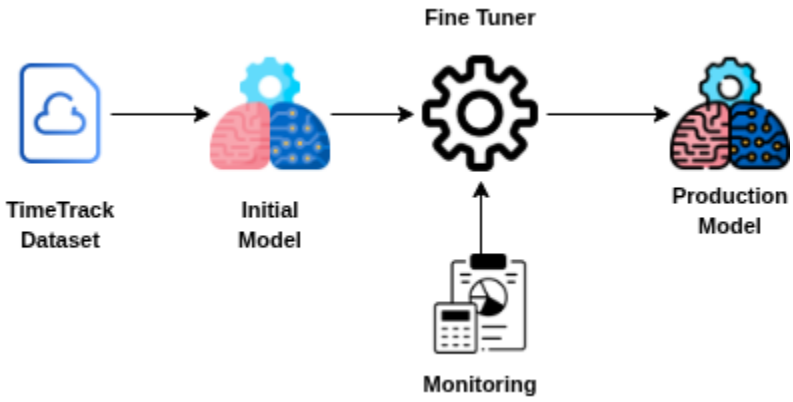

Figure 17: Fine-tuning pipeline for microservice migration: initial model training on a general dataset, followed by fine-tuning on real-time monitoring data for infrastructure-specific predictions.

The core concept of the migration algorithm is simple, relying on a machine learning-driven threshold. A migration decision is initiated for all applications hosted on a node whenever the output of the predictive model surpasses the established threshold. However, even after fine-tuning, the model captures the global resource consumption of the target machine. However, like most machine learning models—and particularly time series models—it struggles with peak values, as illustrated in Figure 18. This difficulty arises because the model is optimized to minimize error on the average patterns, making it inherently conservative when predicting outlier events.

[1] Abd Elghani Meliani, Mohamed Mekki, Adlen Ksentini, Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments, Computer Communications,Volume 236, 2025, 108111, ISSN 0140-3664, (https://www.sciencedirect.com/science/article/pii/S0140366425000684)

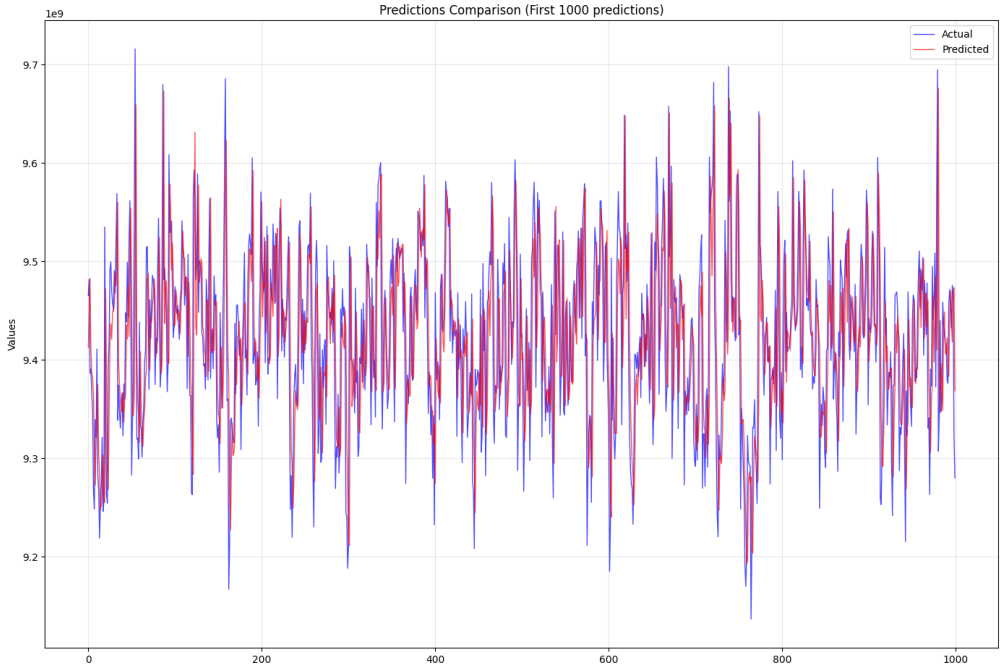

Figure 18: Illustration of the fine-tuned model's predictions versus actual resource usage, highlighting difficulties in accurately capturing peak values.

To address the model errors, we introduce a new threshold adjustment parameter, ***alpha (α).*** The predicted value from the model is now multiplied by this parameter. A migration decision is triggered only if the resulting adjusted value exceeds the predefined threshold:

$MigrationDecision = 1 \ \ if . \alpha \cdot \Pr e\, dictedValue > Threshold, else$ 0.5 illustrates the impact of different alpha values on two types of migration errors:

- Unnecessary Migrations (blue line): Migrations that were triggered but should not have occurred.
- Missed Migrations (orange line): Migrations that should have occurred but were not triggered.

As shown in Figure 19, increasing alpha raises the bar for triggering a migration. This results in fewer unnecessary migrations, but at the cost of more missed migrations. Conversely, decreasing alpha makes the system more sensitive, reducing missed migrations while increasing unnecessary ones.

The optimal alpha depends on system priorities:

- If the system is more sensitive to resource waste (e.g., avoiding unnecessary migrations to reduce overhead), a higher alpha is preferable, even at the cost of some missed migrations.
- If the system is more sensitive to performance degradation (e.g., prioritizing application responsiveness), a lower alpha is preferable to ensure timely migrations, even if some are unnecessary.

This flexibility allows the threshold to be tuned according to the specific operational requirements, balancing the trade-off between migration efficiency and performance protection.

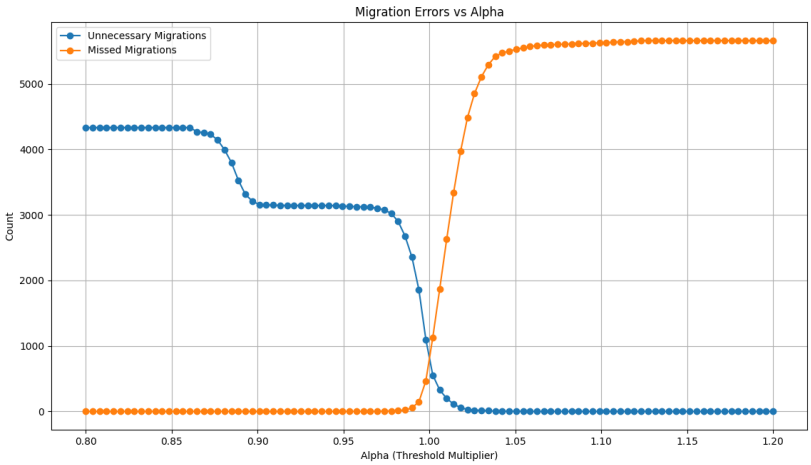

Figure 19: Migration error rates as a function of alpha. Lower alpha values reduce missed migrations (protecting performance), while higher alpha values reduce unnecessary migrations (minimizing resource waste).

To validate the end-to-end migration feature in UC 2, we tested the migration of several microservices of the application under different scenarios. The microservices involved in the migration tests were the frontend, the backend, and DeepStream, with all migrations performed from the edge to the cloud. A total of seven migration scenarios were executed:

1. Scenario 1: Frontend only
2. Scenario 2: Backend only
3. Scenario 3: Frontend + Backend
4. Scenario 4: DeepStream only
5. Scenario 5: Backend + DeepStream
6. Scenario 6: Frontend + DeepStream
7. Scenario 7: All three microservices (Frontend + Backend + DeepStream)

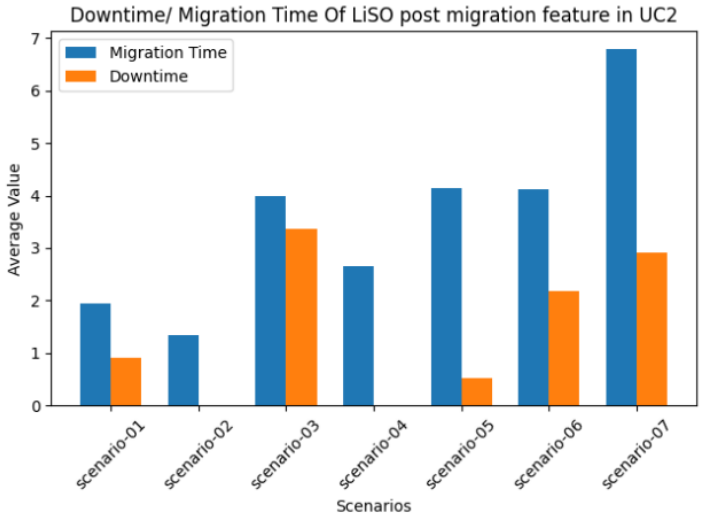

Figure 20: Downtime and total migration time for each scenario.

As shown in the Figure 20, the downtime remains minimal across all scenarios. This is because LiSO implements post-migration features, such as keeping the source node running during the migration process and only switching traffic once the target node is ready. This approach ensures that the application experiences only a brief interruption, regardless of the number or type of microservices being migrated.

In contrast, the total migration time increases progressively with the number and complexity of the microservices being migrated. Scenarios involving multiple services, particularly those including DeepStream (which may involve larger state or data), exhibit higher migration times. This is expected, as more data and service states need to be transferred from the edge to the cloud.

These results demonstrate that LiSO's migration mechanism effectively minimizes disruption to the running application, with downtime remaining consistently low even as the scope of the migration expands. The trade-off between migration time and minimal downtime highlights the efficiency of the post-migration approach in maintaining application availability during live migrations.

4.3.4 Scenario 4: SD-WAN multi-site networking

To validate the SD-WAN-based multi-site connectivity in UC2, we evaluate the network performance and application-level impact across two simulated geographically distributed sites representing the edge and the far-edge. The performance evaluations were conducted using a testbed comprising two edge devices and one SD-WAN controller, each implemented as a virtual machine, alongside the deployed application. The edge devices were interconnected using two different laboratory networks in EUR, with wide-area network characteristics emulated using “netem” to introduce controlled latency, jitter, and bandwidth constraints. The SD-WAN edge VMs were allocated 2GB of RAM and 1 CPU each, while the controller VM was allocated 4GB of RAM and 2 CPUs. The host machine provisioning the edge device VMs was equipped with an Intel® Core™ i7-1365U processor (10 cores, 3.9GHz clock speed) and 32GB LPDDR5-6400MHz RAM.

The application deployment followed the cloud-edge configuration described in Scenario 2 (case 02), where the DeepStream-based object detection model runs on the far-edge device alongside the video streaming service, while the frontend and backend components remain at the edge location.

Two distinct networking scenarios were compared. In the first scenario, a single static path connected the edge and far-edge locations, with traffic-redirect disabled, meaning no dynamic path selection was applied. In the second scenario, SD-WAN was enabled with traffic-redirect, allowing to manage two overlay paths and dynamically rerouting traffic using heuristic thresholds for latency and drops.

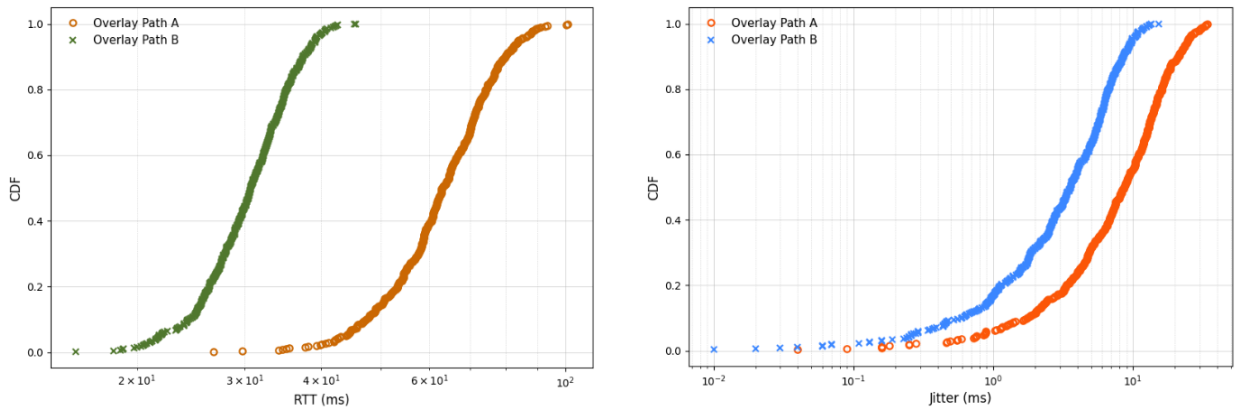

Figure 21: Jitter and RTT CDF Overlay A vs Overlay B.

As shown in Figure 21, Overlay Path B consistently achieves lower latency across the entire distribution, with RTT values ranging from approximately 18 ms to 42 ms and a median RTT (50th percentile) of about 30 ms. In contrast, Overlay Path A exhibits higher latency and greater variability, with RTT values spanning roughly 28 ms to 100 ms and a median RTT of around 65 ms. Similarly, Overlay Path B demonstrates lower jitter, with a median value of approximately 5 ms, compared to Overlay Path A, which has a median jitter of 8 ms.

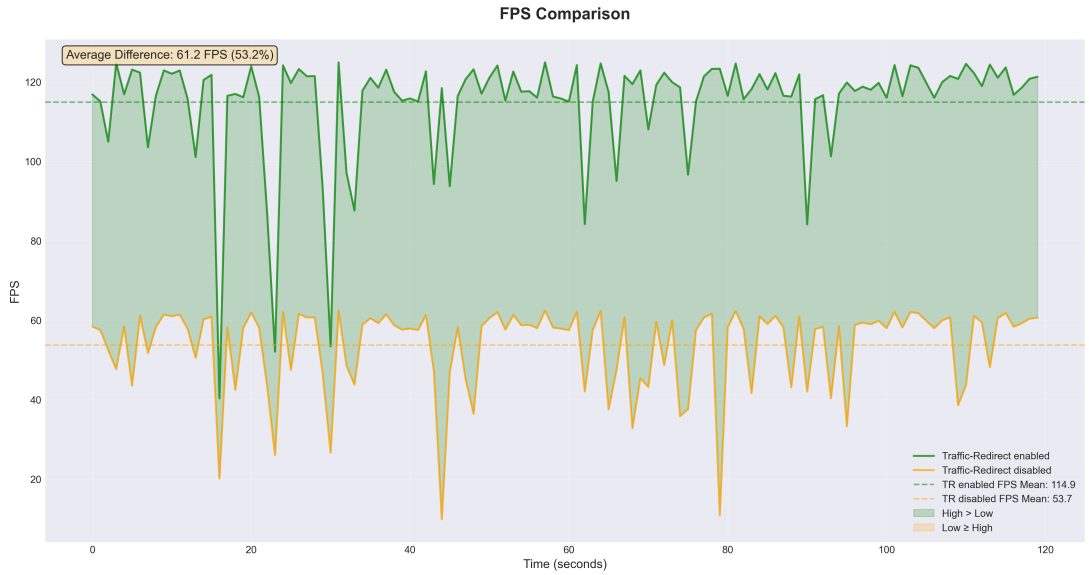

Figure 22: FPS comparing with and without SD-WAN enabled traffic redirection.

The impact of SD-WAN-enabled multi-path management on application performance was evaluated by comparing frames per second (FPS) of the video processing under the two networking scenarios (see Figure 22). With traffic-redirect disabled and only a single static path available, the application achieved a mean FPS of 53.7, with values fluctuating between 46 and 60 FPS throughout the 120-second measurement period. When traffic redirect was enabled, allowing the SD-WAN controller to dynamically reroute traffic across two parallel overlay paths based on heuristic thresholds for latency and packet loss, the application showed substantial improvement, reaching a mean FPS of 114.9. This represents an average gain of 61.2 FPS compared with without SD-WAN configuration. The results demonstrate that dynamic overlay path selection effectively avoids the application from network degradation, enabling stable and high-performance video streaming even under variable WAN conditions.

4.4 Lessons Learned and Recommendations

The implementation and analysis of UC2 yielded significant insights into the critical requirements for managing distributed applications across edge-to-cloud continuums. A principal finding is that conventional cloud management paradigms are insufficient for this distributed context. The exercise identified three fundamental pillars necessary for effective orchestration:

- **Resilient Connectivity and State Continuity:** Application functionality is inherently dependent upon a stable and continuous link between edge and cloud infrastructures. Disruptions to this connectivity directly compromise service delivery.
- **Minimal Downtime During Lifecycle Management:** Standard maintenance and update procedures present a considerable risk to user experience and operational continuity. The capacity to migrate

workloads with minimal or zero downtime is therefore an operational necessity for upholding Service Level Agreements (SLAs).

- **The Imperative of Network Programmability:** Given that contemporary applications are multi-regional and modern infrastructures utilize federation, static network configurations are inadequate. A dynamically programmable network fabric constitutes a critical enabler, allowing the orchestrator to respond effectively to fluctuations in resource availability and application demands across disparate administrative domains.

Informed by these findings, the following recommendations are proposed for the architectural evolution of next-generation cloud-to-edge orchestration platforms:

- **Prioritize the Development of Migration Capabilities:** Orchestration frameworks must advance beyond simplistic restart policies. Future Lifecycle Management (LCM) functionality should incorporate robust live migration features to guarantee business continuity and SLA compliance during updates, horizontal scaling events, or edge node failures.
- **Mandate Comprehensive Observability:** Effective management is predicated on comprehensive measurement. The exposure of granular resource metrics and application telemetry must be regarded as a mandatory core requirement. Such visibility is indispensable for intelligent workload scheduling, proactive troubleshooting, and assuring end-to-end performance across the distributed topology.
- **Implement a Network-as-Code Paradigm:** Orchestrators must treat network configuration as an integral, programmable component of the deployment model. Defining network policies and connectivity rules through declarative code provides the necessary agility to manage complex applications spanning multiple regions and federated infrastructure clusters.

Technological trajectories and application requirements continue to advance. Building upon the existing foundation—which already encompasses support for GPU acceleration—subsequent development phases should concentrate on the following strategic areas:

- **Intent-Based Networking and Autonomous Operations:** Progressing beyond basic programmability, the subsequent objective involves the integration of AI-driven, intent-based networking. This capability would enable the system to autonomously translate high-level service objectives into low-level network configurations, dynamically adapting to prevailing conditions without manual intervention.
- **Enhanced Security within a Zero-Trust Continuum:** The expansion of the edge environment concurrently broadens the potential attack surface. Future investigations should focus on implementing a Zero Trust architecture across the entire edge-to-cloud continuum. This includes exploring service mesh integrations for automated mTLS, fine-grained policy enforcement at the edge, and the utilization of hardware-based security features (such as TPMs for trusted boot and attestation) for edge nodes.
- **Advanced GPU Scheduling and Multi-Tenancy:** While GPU support is presently enabled, the subsequent challenge lies in optimizing resource utilization. Future efforts should be directed towards:
- **GPU Sharing and Slicing:** Facilitating the efficient sharing of a single GPU among multiple workloads through mechanisms such as time-slicing or Multi-Instance GPU (MIG) capabilities, thereby maximizing resource efficiency at the edge.

5 Use Case 3 - Distributed computing large-scale data

5.1 Introduction

UC3 validated the AC3 CECCM framework against a demanding real-world workload: the automated, scalable processing of large-scale astronomical spectral data across a federated OpenShift infrastructure. The experimental campaign demonstrated end-to-end zero-touch deployment across federated clusters, transparent inter-cluster networking that allowed distributed microservices to communicate without application-level changes, near-linear horizontal scaling efficiency as processing resources increased, observability feeding predictive autoscaling, and reliable bi-directional sovereign data exchange between data providers and consumers. The following sections describe these scenarios in detail, presenting quantitative results and lessons learned.

5.2 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.

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 scalable and distributed cloud-edge continuum approach.

Table 8 and Table 9 list the components and the KPIs (according to DoA) to measure for UC3, respectively.

Table 8. UC3 tested components

Component	Description (role)	Corresponding scenario(s)
GUI	Used as the user-facing entry point for application definition and service interaction	1
OSR	Used for descriptor generation and translation within the UC3 deployment chain	1
Cluster Network Operator	Used to link application microservices across namespaces and clusters	2
Monitoring	Supports the monitoring of application and resource metrics	4
Maestro	Used as the LCM component for lifecycle-managed deployment	1
Scaling Mechanism	Supports automatic scaling based on custom metrics	3

Data Connector (Provider & Consumer)	Provides bi-directional data transfer between the data source and UC3.	5
Data Catalogue	Used to represent the application / data assets in the AC^3 stack	1
Data Manipulator	Implements the core UC3 application logic	3

A detailed description of UC3 is available in deliverable D5.2.

5.2.1 KPI Validation

The KPIs (according to DoA) to measure for this UC and which section addresses them are summarized in Table 3:

Table 9: KPIs of UC3

KPIs	Short description	Corresponding scenario
Processing a 5GB in disk data cube	Reduced by at least 50% in comparison with a standalone computing node.	3
Guarantee high availability of the application	100% reliability.	1, 3, 5

To validate the processing time reduction KPI, the NGC7021 dataset (5.1 GB, 42,806 input files) was processed under two configurations: a single-processor baseline and a five-processor scaled deployment. The single-processor run completed in 80.4 hours, while the five-processor run completed in 15.9 hours, representing an 80.3% reduction in end-to-end processing time and a 5.07x speedup. This substantially exceeds the target of at least 50% reduction compared with a standalone computing node. Per-job processing time remained consistent across both configurations (37.8s vs 39.0s), confirming that the speedup derives from parallelism rather than any change in per-job behaviour. Full experiment results and job-level records are available at https://github.com/EU-AC3/data-manipulator-uc3/tree/main/results.

Availability is ensured through the system's decoupled architecture and Kubernetes-based deployment, while, in this context, reliability refers specifically to its capacity for lossless data processing. Under this framework, we define a successfully processed job as one that was queued, dispatched to a processor pod, executed, and recorded with full timing metrics by the infrastructure. Under this definition, both runs achieved 100% job processing reliability across all 42,806 input files. In the five-processor run, 35 input files (0.08%) did not produce output from the STARLIGHT spectral fitting tool due to convergence limitations on specific input spectra. These files were nevertheless fully processed by the system, logged as failed by the application, and recorded in the failed file manifest. The handling of these application-level failures demonstrates that the system correctly identifies, isolates, and logs problematic inputs without interrupting the processing of remaining jobs.

5.3 Experimental Scenarios

UC3 demonstrates the AC³ CECC framework's capabilities to enable distributed, scalable processing of astronomical datasets across the cloud-edge continuum. The architecture supports automated workload orchestration, dynamic resource allocation, comprehensive observability, and sovereign data exchange. The experimental scenarios evaluated are presented in this section.

The UC3 application is a distributed astronomy processing system that performs spectral analysis of integral field unit (IFU) data cubes to study galaxy stellar populations. Observational datasets are ingested from S3 storage by a producer service, which partitions the data into discrete jobs and distributes them via a RabbitMQ message queue to processor pods running spectral fitting tools such as STARLIGHT and pPXF. Completed results are returned through a separate result queue to the producer, which uploads them back to object storage. A custom Prometheus endpoint exposes application-specific metrics including job queue depth, processing latency, and throughput, enabling both real-time observability and autonomous horizontal scaling. An Eclipse Dataspace Connector (EDC) integration provides policy-controlled, IDS-compliant data exchange between sovereign storage locations, supporting cross-organisational dataset sharing.

The UC3 testbed comprises multiple OpenShift application clusters, each provisioned with 24 vCores, 64 GB RAM, and 200 GB SSD storage, coordinated by a single-node OpenShift hub cluster acting as the multi-cluster control plane. This hub integrates Advanced Cluster Management (ACM) for centralised governance, Maestro as the lifecycle orchestrator, a multi-cluster scheduler for workload placement, a network operator for cross-cluster connectivity, and the Prometheus Operator for observability. Together, this infrastructure provides the foundation for the experimental scenarios described in this section.

5.3.1 Scenario 1: Zero-Touch Application Lifecycle Management

UC3 demonstrates how developers define and manage distributed application deployments through the AC³ Application Gateway (GUI) by inputting configurations for microservices, resources, and requirements. The Ontology and Semantic-aware Reasoner (OSR) translates and validates these inputs into a semantic-aware Application Descriptor (AppD), which is then consumed by the Maestro Life Cycle Manager (LCM) to orchestrate onboarding, instantiation, updates, scaling, and decommissioning.

Objective: To validate that the full UC3 astronomy processing system can be onboarded, instantiated, and decommissioned across a federated OpenShift infrastructure without manual Kubernetes resource creation. This demonstrates the complete automation chain from GUI-based application definition, through OSR semantic validation and AppD composition, to Maestro and ACM orchestration.

5.3.2 Scenario 2: Federated Cloud-Edge Networking

UC3 enables seamless distributed processing of astronomical workloads across geographically dispersed cloud and edge sites. By federating infrastructure resources, the system allows data ingestion at edge locations (e.g., observatory sites) while offloading intensive spectral analysis to remote cloud clusters, ensuring efficient resource utilization without compromising workload orchestration or data flow transparency.

Objective: To demonstrate that UC3 microservices deployed across multiple OpenShift clusters can communicate transparently, without application-level modification, using the WP4 network operator. This validates that the

federated networking layer can sustain production-scale workloads by enabling bidirectional job and result flow between cross-cluster producer and processor pods.

5.3.3 Scenario 3: Dynamic Microservice Scaling

UC3 supports elastic scaling of processing resources to match varying astronomical dataset volumes and computational demands. Horizontal scaling of microservices allows the system to dynamically adjust compute capacity, achieving substantial reductions in processing time for large data cubes while maintaining high throughput and minimal overhead in distributed environments.

Objective: To quantify the horizontal scaling efficiency of the UC3 processing system by varying both the number of processor pods and input dataset size. This evaluates whether the event-driven, RabbitMQ-based architecture achieves near-linear throughput gains as compute resources increase, and confirms its suitability for autonomous HPA-driven scaling under production workloads.

5.3.4 Scenario 4: Monitoring and Performance Tracking

UC3 provides comprehensive, real-time observability across the distributed pipeline, capturing fine-grained metrics on job queues, processing latencies, and resource utilization. This enables proactive bottleneck detection, autonomous scaling decisions, and the generation of training data for predictive models, ensuring reliable performance and efficient operation in production-like astronomical analysis workflows.

Objective: To validate that application-specific metrics exposed through the UC3 custom Prometheus endpoint provide actionable observability for diagnosing bottlenecks, driving autoscaling decisions, and generating training data for predictive models.

5.3.5 Scenario 5: Data Management

UC3 facilitates secure, policy-compliant transfer of astronomical spectral data between federated storage locations. By enforcing data sovereignty and automated exchange lifecycles, the system supports cross-organizational sharing while maintaining consistency, reliability, and low-latency transfers essential for collaborative galaxy evolution studies.

Objective: To evaluate the reliability, performance, and policy compliance of sovereign data exchange between S3 storage locations using the IONOS EDC S3 connector extensions. This demonstrates the full IDS-compliant transfer lifecycle, from asset registration and contract negotiation through to data movement and deprovisioning, for bidirectional astronomical dataset transfers between provider and consumer data spaces.

5.4 Key Results and Achievements

The UC3 experimental scenarios have validated the viability of a cloud-native, distributed architecture for processing large-scale astronomical datasets, demonstrating significant achievements across all evaluated dimensions.

5.4.1 Scenario 1: Zero-Touch Application Lifecycle Management

The integration of UC3 with the CECCM successfully demonstrated zero-touch application lifecycle operations on the UC3 OpenShift testbed.

A key differentiator in UC3 is the use of Maestro as the LCM, integrated with Red Hat's Advanced Cluster Management (ACM) for deployment to the UC3 infrastructure. The onboarding pipeline proceeds through several automated stages: first, the developer defines the application via the GUI, selecting microservice blueprints and data sources from the Service and Data Catalogues. The OSR then performs semantic validation and composes a complete AppD (see Figure 23), capturing microservice definitions, inter-service dependencies, resource requirements, SLA constraints, location properties, and networking topology.

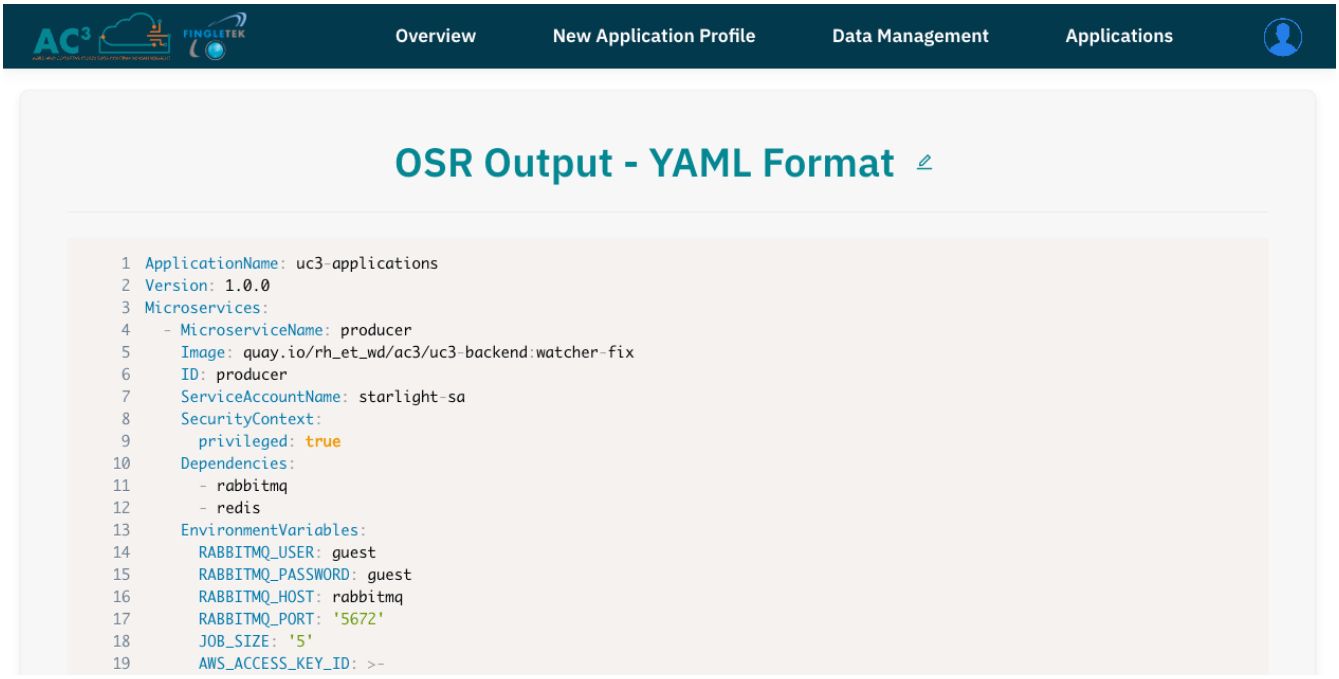

Figure 23: OSR Output - YAML Format

A dedicated AC3-OSR to Maestro Exposure translator converts the AppD into the full set of Kubernetes manifest files (deployments, services, service accounts, role bindings, and persistent volume claims), which are submitted to Maestro following the TMF-641 service ordering standard (see Figure 24). Maestro transforms these manifests into ManifestWork resources and dispatches them to ACM for deployment to the target cluster.

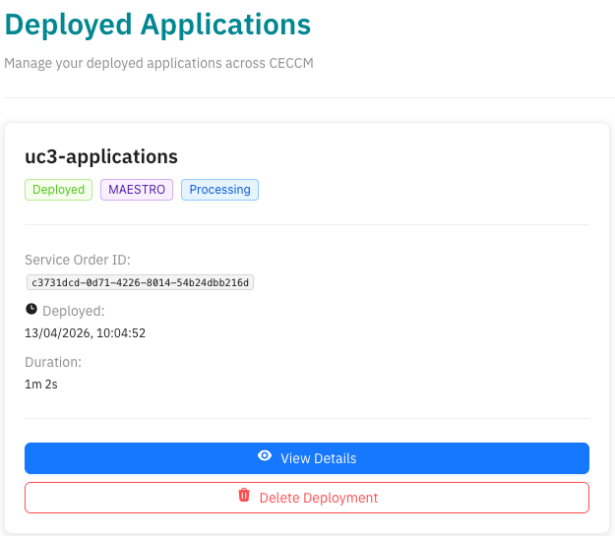

Figure 24: Deployed Applications

For UC3, the AppD specifies ten microservices (see Figure 25) comprising the orchestration and processing tiers of the astronomy pipeline. The scenario focuses on validating that this end-to-end pipeline can be executed without manual intervention, reducing the operational burden on both application developers and DevOps teams.

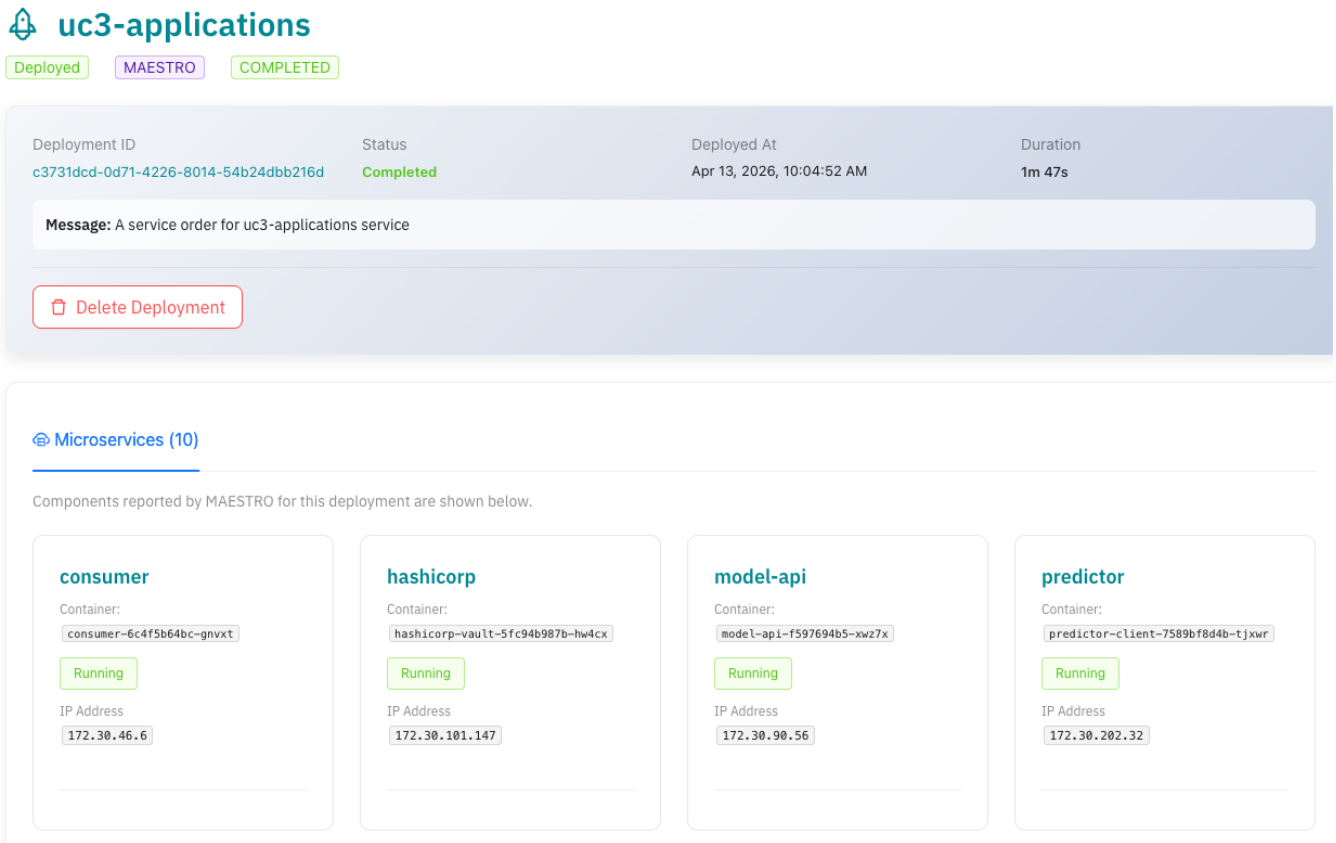

Figure 25: Use Case 3 – Applications.

Key achievements:

- Performed full CRUD (create, read, update, delete) operations on UC3 application deployments via the OSR GUI.
- OSR successfully translated GUI inputs into validated Application Descriptors (AppD).
- Maestro LCM autonomously handled deployment instantiation, configuration, and termination.

5.4.2 Scenario 2: Federated Cloud-Edge Networking

A key objective of the UC3 architecture is enabling distributed processing across geographically dispersed cloud and edge infrastructure while maintaining seamless workload orchestration. To validate this capability, the UC3 system components were deployed across two distinct OpenShift clusters, with the producer pod and supporting services running on Cluster 4, while processor pods were deployed on Cluster 1 (see Figure 26). This configuration simulates real-world scenarios where data ingestion occurs at one location (e.g., an observatory edge site) while compute-intensive processing is offloaded to a separate cloud infrastructure.

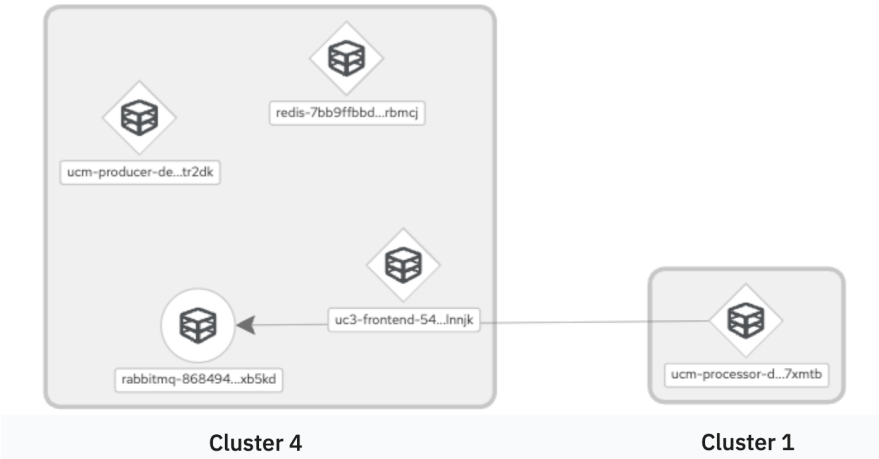

Figure 26: Network topology of Cluster 1 and Cluster 4.

To establish secure, transparent connectivity between the clusters, the Kubernetes network operator developed as part of the WP4 network programmability task was leveraged. This operator automates the deployment and configuration of Skupper (a layer 7 service interconnect that creates encrypted Virtual Application Network (VAN) links between Kubernetes namespaces across clusters). The operator abstracts the complexity of multi-cluster networking by exposing a declarative Custom Resource Definition (CRD), enabling administrators to establish cross-cluster service connectivity through simple YAML manifests rather than manual network configuration.

With the network link established, RabbitMQ instances deployed on both clusters are federated via the Skupper router (rabbitmq-skupper:5672), appearing as a unified message broker to the application components. The producer pod on Cluster 4 publishes job batches to the producer_to_processor_queue as normal; these messages are transparently routed across the inter-cluster link to the processor pods on Cluster 1 (see Figure 27), which consume and process them without any application-level awareness of the network topology. Completed results follow the reverse path through the processor_to_producer_queue, enabling bidirectional data flow across the federated infrastructure.

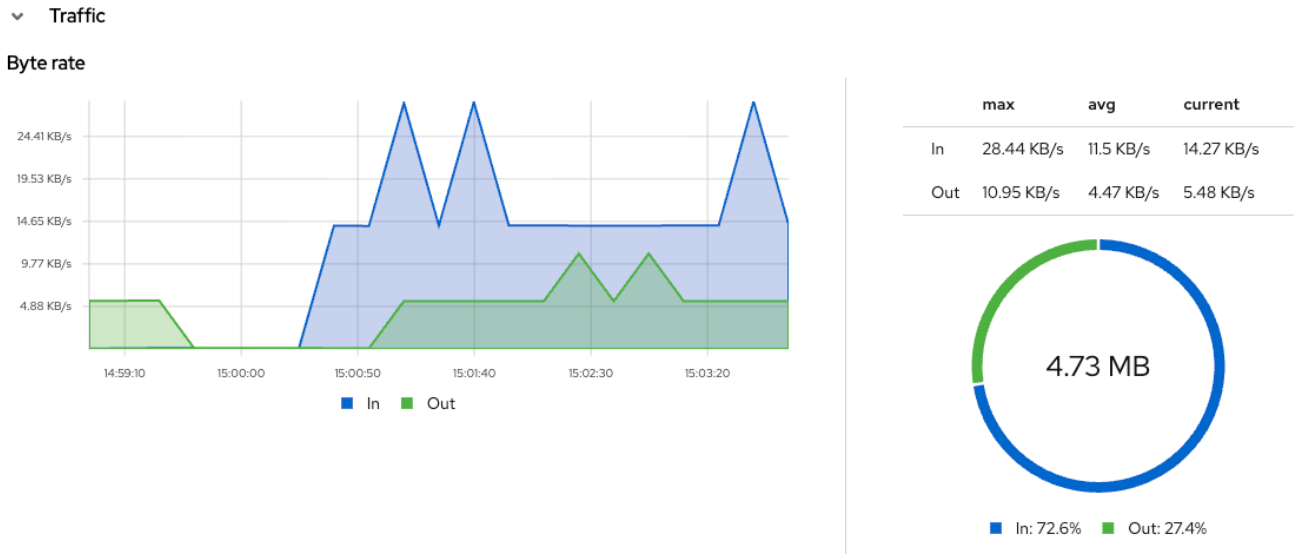

Figure 27: Skupper Router / Network Link Visualisation.

The Skupper console confirmed a stable connection sustained across the Skupper link (see Figure 28), demonstrating the viability of this approach for production-scale distributed astronomical data processing.

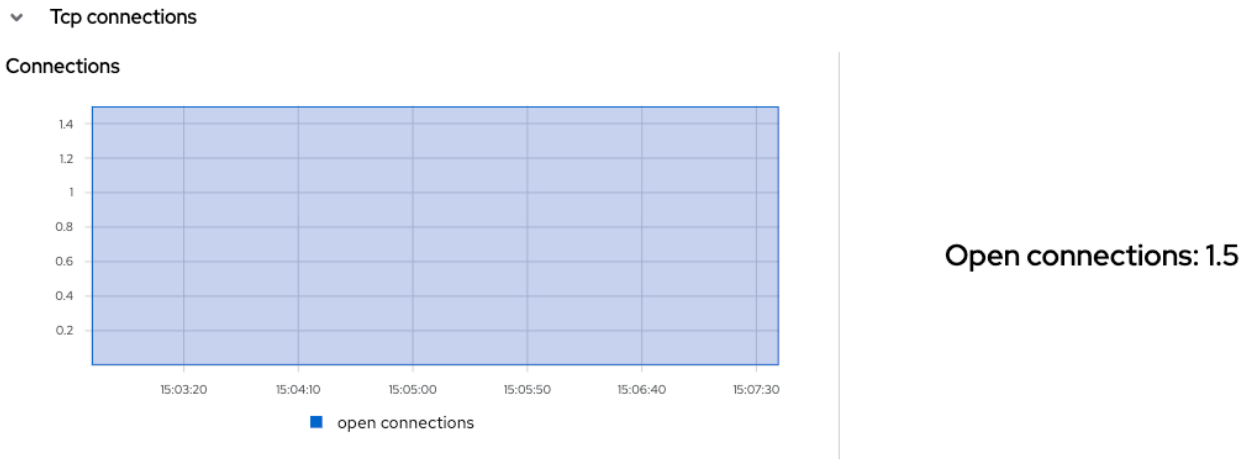

Figure 28: Open Connections Graph

Key achievements:

- Transparent bidirectional job and result flow across clusters
- Declarative CRD-based setup eliminates manual configuration
- Stable high-concurrency operation for production-scale processing

5.4.3 Scenario 3: Dynamic Microservice Scaling

A fundamental requirement for processing large-scale astronomical datasets is the ability to dynamically scale computational resources in response to workload demands. To validate the horizontal scaling capabilities of the UC3 system, a series of experiments was conducted, varying both the number of processor pods (1, 3, and 7) and the input dataset size across three batch configurations: Batch 1 (16.27 MB), Batch 5 (81.35 MB), and Batch 10 (162.7 MB). Each batch represents a proportional subset of the full NGC galaxy spectral data, enabling systematic evaluation of how processing time scales with both data volume and available compute resources.

Table 10. UC3 Horizontal Scaling Results. Processing time decreases significantly as the number of pods increases across all dataset sizes, demonstrating effective dynamic microservice scaling.

Metric	1 Pod	3 Pods	7 Pods
Batch 1 (16.27 MB)	712.72 / 11m 53s	280.47 / 4m 40s	170.26 / 2m 50s
Batch 5 (81.35 MB)	4085.99 / 1h 8m 6s	1226.37 / 20m 26s	429.83 / 7m 16s
Batch 10 (162.7 MB)	8759.65 / 2h 26m 59s	2919.88 / 48m 40s	1103.10 / 18m 23s

The results demonstrate near-linear scaling efficiency across all tested configurations. For the largest workload (Batch 10, 162.7 MB), increasing from 1 to 3 processor pods reduced total processing time from 2 hours 27 minutes to 48 minutes 40 seconds, a 3× speedup closely matching the 3× increase in resources (see Figure 29). Scaling further to 7 pods achieved an additional 2.6× improvement, completing the same workload in just 18 minutes 23 seconds. This represents an overall 7.9× speedup relative to single-pod execution, indicating minimal coordination overhead in the distributed architecture. The RabbitMQ-based job distribution effectively balances workload across available processors, with each pod independently consuming and processing batches from the shared queue.

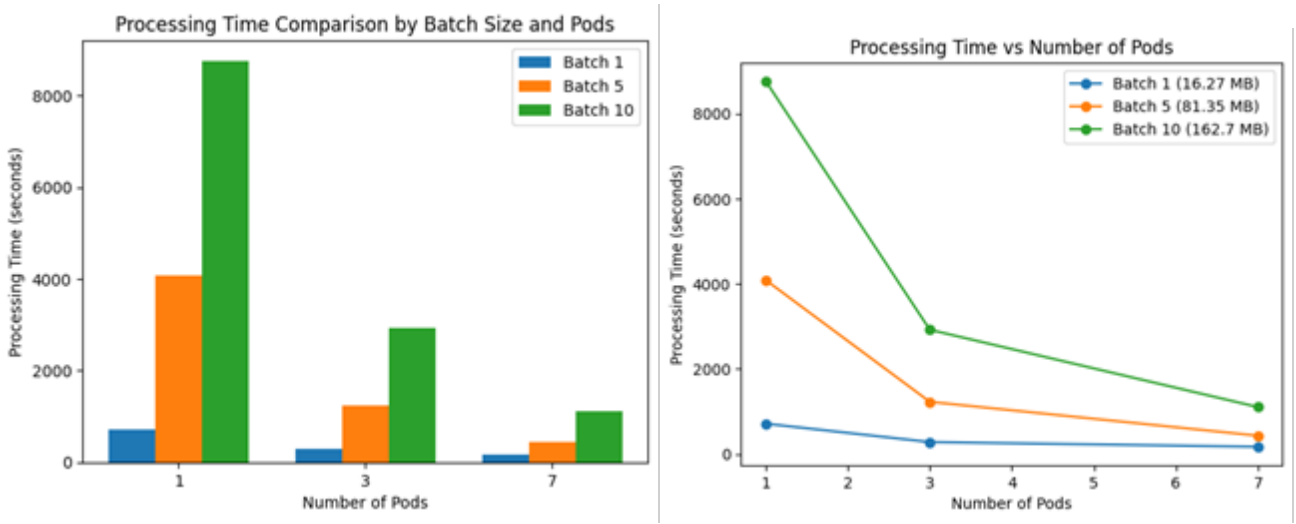

Figure 29: Processing Time vs Number of Processing Pods

Notably, the scaling efficiency remains consistent across different data volumes, suggesting that the system architecture can accommodate significantly larger datasets without degradation in parallel efficiency. The near-linear relationship between pod count and throughput validates the event-driven microservice design, where stateless processor pods can be dynamically instantiated or terminated based on queue depth. This capability integrates seamlessly with Kubernetes Horizontal Pod Autoscaler (HPA) for autonomous resource management in production deployments.

Key achievements:

- 7.9× overall speedup for largest batch (from 2h 27m to 18m 23s)
- Consistent near-linear scaling across volumes with low overhead
- Effective autonomous resource adjustment for varying workloads

5.4.4 Scenario 4: Monitoring and Performance Tracking

Comprehensive observability is essential for understanding system behaviour, diagnosing performance bottlenecks, and generating training data for predictive autoscaling models. The UC3 system implements a multi-layered monitoring architecture that captures metrics at the job, batch, and infrastructure levels, with seamless integration into a centralised Prometheus monitoring framework.

At the application layer, the UC3 backend exposes a custom /metrics endpoint implementing the Prometheus exposition format. This endpoint publishes fine-grained job-level metrics, including queue wait time (*astroapp_job_queue_duration_seconds*), processing duration (*astroapp_job_processing_duration_seconds*), total end-to-end latency (*astroapp_job_total_duration_seconds*), job payload size (*astroapp_job_size_mb*), and queue depth at time of submission (*astroapp_job_queue_ahead_length*). Additionally, aggregate batch-level metrics track total batch duration and completed job counts, enabling analysis of throughput characteristics under varying load conditions. A dedicated queue length monitor polls RabbitMQ every 10 seconds, exposing real-time queue depth via the *astroapp_queue_length* gauge, a critical signal for horizontal pod autoscaler (HPA) configuration.

Integration with the centralised AC3 monitoring infrastructure is achieved through OpenShift's User Workload Monitoring capability. A ServiceMonitor custom resource, annotated with openshift.io/user-monitoring: 'true', instructs the platform's Prometheus Operator to automatically discover and scrape the UC3 metrics endpoint at 30-second intervals. A dedicated UC3-monitoring service account, bound to the cluster-monitoring-view ClusterRole, enables secure cross-namespace metrics federation, allowing the central Prometheus instance to aggregate UC3 metrics alongside infrastructure telemetry from all participating clusters. This unified observability plane provides operations teams with a single-pane-of-glass view across the distributed multi-cluster deployment, supporting both reactive incident response and proactive capacity planning.

Key achievements:

- Detailed, real-time job and queue telemetry for diagnostics and autoscaling
- Unified cross-cluster observability supporting incident response and planning

5.4.5 Scenario 5: Data Management

To evaluate the data transfer and sovereignty capabilities of the UC3 system, a controlled experiment using the S3 IONOS Extensions for Eclipse Dataspace Connectors (EDC) was conducted to transfer astronomical spectral data between S3 storage locations. The test was initiated through the application's GUI, which triggers the EDC transfer workflow for each file in the dataset. This workflow encompasses the complete data exchange lifecycle mandated by International Data Spaces (IDS) standards: asset registration, ODRL policy creation, contract negotiation, data transfer execution, and resource deprovisioning. A total of 30 files were transferred bidirectionally between provider and consumer data spaces, achieving 100% reliability across both transfer directions.

Table 11. UC3 Data Transfer Performance via EDC. Inbound and outbound S3 transfers achieved 100% reliability with consistent average durations (~16–17 seconds), confirming robust and policy-compliant data exchange performance.

Metric	Inbound (Provider - Consumer)	Outbound (Consumer - Provider)
Average Duration	15.94 seconds	16.90 seconds
Median Duration	~16 seconds	~17 seconds
Reliability	100% Success (30/30 files)	100% Success (30/30 files)
Consistency (Std Dev)	±2.1 seconds	±1.72 seconds

The results demonstrate consistent transfer performance across both directions, with inbound transfers completing in an average of 15.94 seconds and outbound transfers averaging 16.90 seconds, a difference of approximately 9% (see Figure 30). This modest asymmetry is attributable to the contract negotiation phase, where outbound transfers require additional credential provisioning steps (INITIAL → VERIFIED → FINALIZED) compared to the streamlined inbound path (REQUESTING → FINALIZED).

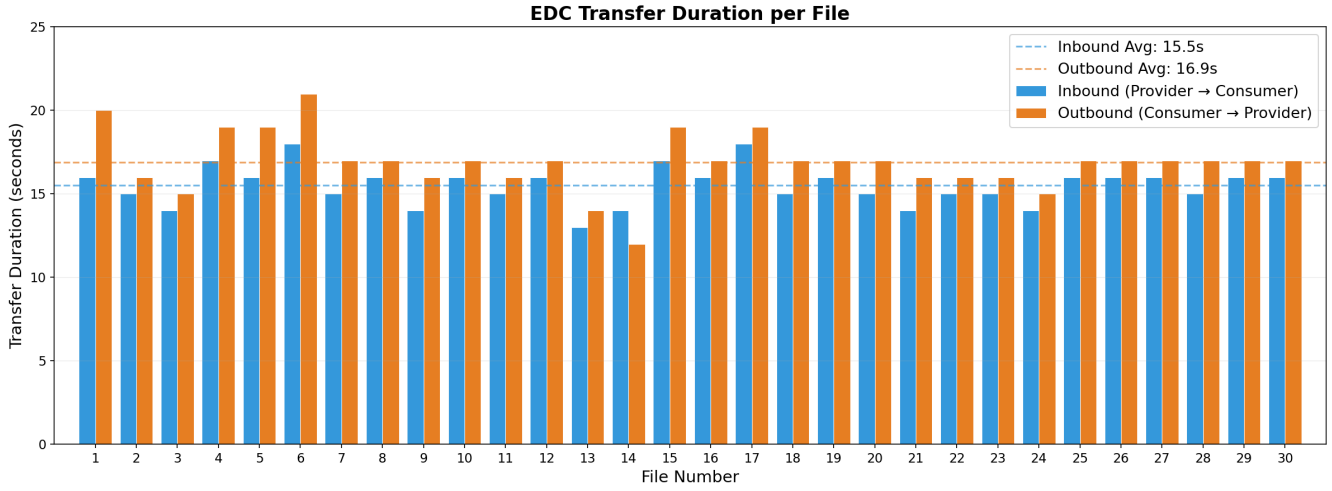

Figure 30: EDC Transfer Duration per File

Detailed lifecycle analysis reveals that the majority of transfer time is consumed by the transfer provisioning phase (~6.3 seconds) and actual data movement (~4.2 seconds), which remain consistent regardless of direction.

The pre-transfer setup comprising asset registration, policy creation, contract definition, and catalogue fetch, adds approximately 1.5 seconds of overhead per transfer.

Table 12. UC3 EDC Transfer Lifecycle Breakdown. End-to-end transfer time (~17–18s) is primarily dominated by provisioning and data movement phases, with minor directional differences arising from extended outbound contract negotiation steps.

Lifecycle Stage	Transition	Inbound (P→C)	Outbound (C→P)
Asset Creation	API Call	0.58s	0.11s
Policy Creation	API Call	0.14s	0.13s
Contract Definition	API Call	0.12s	0.11s
Catalogue Fetch	Query provider	0.81s	0.63s
Contract Negotiation	INITIAL - FINALIZED	2.53s	4.50s
Transfer Provisioning	INITIAL - STARTED	6.41s	6.32s
Data Transfer	STARTED - COMPLETED	4.27s	4.20s
Deprovisioning	DEPROVISIONED	2.44s	2.11s
Total Lifecycle	End-to-end	~17.3s	~18.1s

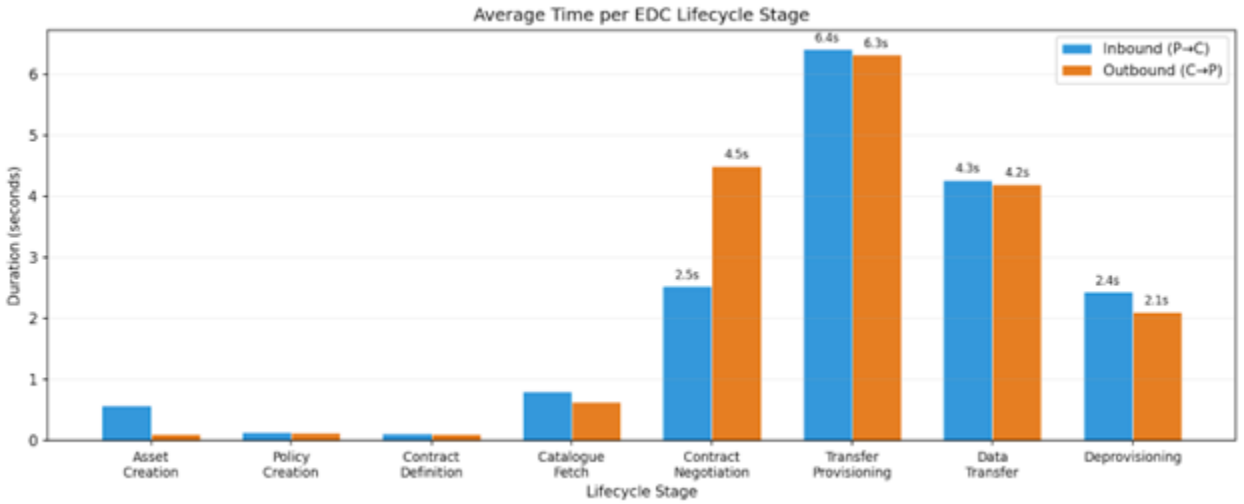

Figure 31: Average Time per Lifecycle Stage

The contract negotiation phase accounts for the primary directional difference: outbound transfers spend approximately 4.5 seconds in negotiation compared to 2.5 seconds for inbound (see Figure 31), reflecting the additional handshake required when the consumer initiates data retrieval with ephemeral credential generation. Despite this overhead, the connector demonstrated consistent and reliable operation with low variance (standard deviation of 1.8s), validating its suitability for federated astronomical data exchange scenarios where data sovereignty and policy enforcement are paramount requirements.

Key achievements:

- 100% reliable sovereign transfers with low variance
- Autonomous IDS/GAIA-X lifecycle execution
- Consistent ~16s performance suitable for federated sharing

5.5 Lessons Learned and Recommendations

The experimental validation of UC3 provided practical insights into multi-cluster orchestration, event-driven scalability, federated observability, and sovereign data exchange for compute-intensive scientific workloads. The main lessons learned per scenario are summarised below.

5.5.1 Scenario 1 - Zero Touch Configuration

The automated deployment pipeline from OSR through Maestro to ACM successfully eliminated manual Kubernetes resource creation on the UC3 OpenShift testbed. The AC3-OSR to Maestro Exposure translator bridged the technology-agnostic AppD model and the infrastructure-specific manifests, generating Kubernetes manifest files from the UC3 AppD covering deployments, services, service accounts, role bindings, and persistent volume claims. The TMF-641 compliant service ordering ensured reproducible deployment workflows.

A key observation was that the expressiveness of the AppD model plays a significant role in the quality of the resulting deployment. Descriptors that capture security contexts, persistent storage requirements, and inter-service networking enable the automation pipeline to make well-informed decisions at every stage. The AppD already includes location-aware fields such as geographical affinity, region, and infrastructure affinity, which provide a foundation for future label-based cluster routing across federated environments.

Recommendation: The location-aware fields in the AppD (geographical affinity, region, infrastructure affinity) should be leveraged to support label-based cluster routing as federated infrastructures grow in scale and heterogeneity. Enriching the label vocabulary to capture additional infrastructure characteristics such as storage class, network locality, and available processing capacity would allow Maestro and ACM to make increasingly fine-grained placement decisions.

5.5.2 Scenario 2 - Federated Cloud-Edge Networking

The declarative CRD-driven Network Operator dramatically simplified multi-cluster connectivity, enabling transparent RabbitMQ federation across OpenShift clusters. By abstracting away manual network configuration (token distribution, secret management, and Skupper link lifecycle), the operator reduced operational overhead to a single YAML manifest per cross-cluster link. The application-level transparency was particularly valuable: producer and processor pods communicated across clusters without any awareness of the underlying network topology, validating the viability of this approach for production-scale distributed processing.

The primary challenge encountered was ensuring reliable synchronisation between the hub and managed clusters as network configurations evolve, particularly when scaling events require new links to be established dynamically.

Recommendation: The declarative network-as-code model should be extended to support event-driven link creation, where scaling or migration actions automatically trigger the establishment of new cross-cluster connectivity. Integrating link health monitoring into the LCM feedback loop would also enable proactive re-routing if inter-cluster connectivity degraded.

5.5.3 Scenario 3 - Dynamic Microservice Scaling

The event-driven architecture with RabbitMQ-based job distribution proved highly scalable and low-overhead. Stateless processor pods independently consuming from a shared queue eliminated the need for complex leader-election or sharding logic, making Kubernetes HPA integration straightforward and effective. The near-linear scaling efficiency (7.9x speedup with 7 pods) was consistent across all tested data volumes, indicating that the architecture can accommodate significantly larger datasets without degradation in parallel efficiency.

A key insight was that for this work/event dispatching model, queue depth and average batch processing time proved to be substantially better indicators of workload demand than traditional CPU and RAM metrics. The stateless nature of the processing (each batch is self-contained) means that CPU utilisation alone does not accurately reflect whether the system is falling behind on its workload.

Recommendation: For event-driven microservice architectures, queue-based metrics should be adopted as the primary autoscaling signal, with CPU and RAM serving as secondary safeguards for resource exhaustion. Predictive autoscaling models for these workloads should be trained on representative production loads, as training on synthetic or limited experimental data risks poor generalisation to real variability, such as seasonal observation cadences, bursty data ingestion from different instruments, and multi-tenant interference in shared infrastructure.

5.5.4 Scenario 4 - Monitoring and Performance Tracking

Application-specific metrics exposed via the custom Prometheus /metrics endpoint proved significantly more actionable than infrastructure-level metrics alone. The fine-grained telemetry, including queue wait time, processing duration, end-to-end latency, and queue depth at submission, enabled precise diagnosis of bottlenecks and provided the necessary training data for predictive autoscaling models. OpenShift's User Workload Monitoring capability, combined with the ServiceMonitor custom resource, simplified Prometheus integration by automatically discovering and scraping the UC3 metrics endpoint without requiring manual configuration. Cross-cluster federation via Thanos provided a unified observability plane across the distributed deployment.

The experience confirmed that the value of observability is ultimately bounded by the quality and representativeness of the collected data. Sustained collection over extended operational periods, capturing the natural variability of different instrument data volumes and processing characteristics, is essential for building reliable predictive models.

Recommendation: Application-level metrics should be treated as first-class citizens alongside infrastructure metrics in any cloud-edge monitoring strategy. Exposing domain-specific signals (such as queue depth and batch throughput) through standardised interfaces like the Prometheus exposition format enables seamless integration with platform-native monitoring tools and avoids the need for bespoke observability infrastructure.

5.5.5 Scenario 5 - Data Management

The EDC S3 Extended Connectors achieved 100% transfer reliability across all 30 bidirectional file transfers, with consistent and low-variance performance (approximately 16 seconds average per file). The full IDS-compliant lifecycle (asset registration, ODRL policy creation, contract negotiation, data transfer, and deprovisioning) executed autonomously via the GUI-triggered workflow, validating the suitability of this approach for federated astronomical data exchange where data sovereignty and policy enforcement are paramount.

The lifecycle analysis revealed that contract negotiation is the primary source of directional asymmetry (approximately 4.5 seconds outbound versus 2.5 seconds inbound), attributable to the additional credential provisioning required when the consumer initiates retrieval. While acceptable for the current transfer volumes, this overhead would compound at scale.

Recommendation: For high-volume transfer scenarios, the contract negotiation phase should be optimised, for instance by caching or reusing negotiated contract agreements for repeated transfers of the same asset type and policy combination. Exploring bulk transfer mechanisms that amortise the per-file negotiation cost across an entire dataset batch would also improve throughput for large observational campaigns.

Recommendations for Future Development

Building on the scenario-specific findings, two recommendations emerge for the continued evolution of the AC3 platform in the context of scientific computing workloads:

Continue onboarding additional astronomical processing applications: UC3 validated the architecture across two distinct processing applications (Starlight and pPXF), each with different computational profiles, language runtimes, and execution models. The successful integration of these heterogeneous tools within the same event-driven pipeline confirms that the architecture generalises beyond a single application. Future work should prioritise expanding the range of supported analysis tools, verifying that the deployment pipeline, scaling behaviour, and monitoring integration remain robust as the diversity of workload characteristics increases.

Leverage stateless workload characteristics for cross-cluster scaling: UC3 independently validated both federated cross-cluster networking (Scenario 2) and near-linear horizontal scaling (Scenario 3). Since processor pods are stateless (each batch is self-contained), there is a natural opportunity to combine these two capabilities for cross-cluster workload distribution. Rather than scaling only within a single cluster, processor pods can be replicated across clusters to dynamically exploit spare capacity across the entire federated infrastructure. This pattern is particularly well-suited to scientific computing workloads, which tend to be highly parallel and tolerant of variable processing latency.

6 Conclusions

This document has presented the successful integration and experimental validation of the Agile and Cognitive Cloud-edge Continuum management (AC3) framework, demonstrated across three highly diverse use cases. The results consistently show that the Cloud Edge Computing Continuum Manager (CECCM) empowers zero-touch configuration, dynamic orchestration, and latency-aware placement of distributed services.

In UC1 (IoT and Data), deploying microservices and AI-powered intelligence at the edge resulted in a 65% reduction in processing latency and a massive 96.9% decrease in egress traffic volume compared to cloud-only architectures. Furthermore, the system successfully demonstrated semantic-aware application management and seamless microservice migration, maintaining service continuity during changing infrastructure conditions.

In UC2 (Smart Monitoring System using UAVs), the architecture validated the automated deployment of a multi-domain application spanning cloud, edge, and far-edge (UAV) resources. Key achievements included latency-aware placement of video analytics to minimize bandwidth, resilient service continuity through the migration of workloads when UAV resources degraded, and secure SD-WAN overlay networking across geographically separated testbeds.

Finally, UC3 (Astronomy Data Processing) proved the platform's capability to handle massive, compute-intensive astronomical datasets. The federated cloud-edge networking, combined with event-driven architecture, enabled dynamic and near-linear scaling of microservices that achieved a 7.9x processing speedup for large data cubes. Additionally, it demonstrated 100% reliable sovereign data transfers and unified cross-cluster observability.

Overall, the experimental scenarios confirm that the AC3 architecture significantly optimizes performance, minimizes cloud infrastructure costs, and enhances operational resilience. By bridging physical and digital realms and providing automated lifecycle management, AC3 successfully accelerates the development and deployment of complex microservice applications across the entire cloud-edge continuum.