D5.4 Final Report on Components and Testbed Integration

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**	June 30, 2026
Project URL	www.ac3-project.eu		
Deliverable	D5.4 Final Report on Components and Testbed Integration (S)		
Work Package	WP5		
Contractual due date	M37: 20 January 2026	**Actual submission date**	M42: June 2026
Type	R (Report, Document)	**Dissemination Level**	PU (Public)
Lead Beneficiary	IQU		
Responsible Author	Eduardo Ojeda (IQU)		
Contributors	Ray Carroll (RHT), Ben Capper (RHT), D. Amaxilatis (SPA), D. Klonidis, N. Psaromanolakis (UBI), Mario Chamorro (UCM), Jeffrey Redondo (IQU), Abdelhak Kadouma (FIN), Mohamed Mekki (EUR), Meliani Abd Elghani (EUR), Amadou Ba (IBM)		
Peer reviewer(s)	John Beredimas (CSG), Vrettos Moulos (UPR)		

Revision history (including peer reviewing & quality control)

Funded by the European Union

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

Version	Issue Date	% Complete	Changes	Contributor(s)
V0.1	10/10/2025	5%	Initial Deliverable Structure	Eduardo Ojeda (IQU) Jeffrey Redondo (IQU) John Vardakas (IQU)
V0.2	20/02/2026	10%	First Round of Contributions	All
V0.3	16/03/2026	30%	Review, format and conclusion	Eduardo Ojeda (IQU)
V0.4	20/04/2026	40%	Addressing internal reviewer comments	All
V0.5	24/04/2026	50%	Review, format and conclusion	Eduardo Ojeda (IQU) Jeffrey Redondo (IQU)
V0.6	10/05/2026	80%	Interim review	John Beredimas (CSG) Vrettos Moulos (UPR)
V0.7	27/05/2026	90%	TM review	Adlen Ksentini (EUR)
V1.0	11/06/2026	100%	Addressing TM Reviewers	Eduardo Ojeda (IQU)

Disclaimer

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

While the information contained in the documents is believed to be accurate, the author(s) or any other participant in the AC^3 consortium make no warranty of any kind with regard to this material, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the AC^3 consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing, neither the AC^3 Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information, advice or inaccuracy or omission herein.

Copyright message

© AC^3 Consortium. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged

Table of Contents

© AC³ 2023

© AC3 2023

List of Figures

List of Tables

© AC³ 2023

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
CNI	Container Network Interface
CORS	Cross-Origin Resource Sharing
CRD	Custom Resource Definition
CRUD	Create, Read, Update, Delete
CSRF	Cross-Site Request Forgery
DCAT	Data Catalogue Vocabulary
DNS	Domain Name System
EDC	Eclipse Data Connector
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HPA	Horizontal Pod Autoscaler
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
K3s	Lightweight Kubernetes distribution designed for resource-constrained environments
K8s	Kubernetes
KPI	Key Performance Indicator
LCM	Life-Cycle Management
LiSO	Lightweight Edge Slice Orchestration

© AC³ 2023

LMS	Local Management System
LSTM	Long Short-Term Memory
ML	Machine Learning
MEC	Multi-Access Edge Computing
MEP	MEC Platform
NFV	Network Functions Virtualisation
NSD	Network Service Descriptors
OSR	Ontology and Semantic-Aware Reasoner
OSS	Open-source Software
RBAC	Role-Based Access Control
SD-WAN	Software-Defined Wide Area Network
SLA	Service Level Agreement
UAV	Unmanned Aerial Vehicle
UC	Use Case
VIM	Virtual Infrastructure Manager
XGBoost	Extreme Gradient Boosting

© AC3 2023

1 Executive Summary

This deliverable, **D5.4 – Final Report on Components and Testbed Integration** supports the final software delivery of the AC^3 project. In contrast to the previous deliverables in WP5, its objective is not to present a new technical design, a detailed description of experimental campaigns, or an extended analysis of performance results. Instead, D5.4 provides evidence of the software assets implemented within AC^3 as part of the final Cloud-Edge Computing Continuum Manager (CECCM) implementation and software release. It includes the components made available through the project GitHub organization, their ownership, access conditions, degree of openness, and role in the final integrated use case environments.

This report provides a consolidated overview of the software modules developed in AC^3. It includes the main functional building blocks of the framework, the repositories in which they are maintained, the responsible partners, and their relation to the three project use cases. It also describes the final software-integrated state of UC1, UC2, and UC3, focusing on which modules were deployed and distributed across the relevant domains (e.g., cloud, edge, far edge, data source, and networking). In addition, it establishes how the delivered software can be accessed, reused, and maintained beyond the project lifetime. This scope is directly reflected in the structure of D5.4, which includes a software inventory summary, detailed component reporting, per-use-case software integration sections, and a dedicated section on software governance and post-project sustainability.

The software components covered in this deliverable span the main AC^3 functional areas, including the Application Gateway, the Ontology and Semantic-Aware Reasoner, catalogue-related components, Local Management Systems, monitoring components, zero-touch configuration and lifecycle management components, and data management components. These software assets collectively support the operational realization of the AC^3 CECCM across the three project use cases. Therefore, this report serves as the final reference for WP5, detailing the software that has been implemented, integrated, and made available by the consortium.

For clarity, this deliverable should be read together with the earlier WP5 outputs, but it does not duplicate them. Architectural design and framework definition are addressed in the WP2–WP4 deliverables, while the initial integration planning was covered in D5.1 [1] and the component integration status was documented in D5.2 [2]. Experimental validation, achievements, and KPI-oriented results are treated separately in D5.3 [3] and related WP5 reporting. D5.4, therefore, complements these documents by providing the final software-oriented view of the AC^3 implementation, with emphasis on repository availability, software traceability, and post-project accessibility.

© AC3 2023

2 Introduction

This section establishes the context of the work detailed in this deliverable. It outlines the main purpose of the deliverable, its core objectives, and how it connects to the broader project framework and associated deliverables. Additionally, this section delineates the anticipated outcomes and their alignment with the commitments outlined in the Grant Agreement. It concludes by providing the structural organization of the deliverable.

2.1 Overview – Purpose and Objectives

The purpose of this deliverable is to provide the final software-oriented view of the AC^3 implementation achieved within WP5. In line with the agreed structure of D5.4, this document focuses on the software components delivered by the consortium and made available through the AC^3 GitHub organization, together with the information required to identify them, access them, and understand their role within the overall AC^3 framework. Rather than presenting a new architectural specification or a detailed validation report, D5.4 serves as a concise supporting document for the final software release associated with the implementation of the CECCM.

More specifically, the objectives of this deliverable are threefold. First, it documents the AC^3 software assets delivered through GitHub, including the implemented modules, their repository references, responsible partners, and availability conditions. Second, it provides authorised stakeholders with a clear entry point to the implemented components, enabling traceability between the software reported in the project and the repositories through which that software is distributed. Third, it describes how the delivered software maps to the main AC^3 functional blocks, thereby linking the implemented repositories to the CECCM capabilities exercised across the project use cases. This objective is consistent with the WP5 role of integrating the different components devised in WP2, WP3, and WP4 into a coherent CECCM software stack connected to the AC^3 testbeds and data sources.

Accordingly, D5.4 should be understood as the final report on the software components delivered in WP5. It complements the earlier WP5 deliverables without duplicating them: D5.1 [1] defined the initial integration and proof-of-concept plan, while D5.2 [2] reported the intermediate status of CECCM implementation and component integration. Building on that foundation, the present deliverable consolidates the final set of software components, their repository-level availability, and their relation to the AC^3 functional architecture and use case deployments.

By fulfilling these objectives, D5.4 provides a consolidated overview of the AC^3 software framework in its final integrated state, ensuring the accessibility, traceability, and long-term availability of the project's technical outputs.

2.2 Link with other Project Activities

This deliverable is the final software-oriented output of WP5 – Integration and Demonstration. According to the Grant Agreement [3], WP5 covers the integration of the different AC^3 components into the CECCM software, their connection with the project testbeds and data sources, and the demonstration of the project use cases through the activities of Task T5.1 (AC^3 components integration) [4] and Task T5.2 (Testbed integration) [4] . Within this context, D5.4 concludes the WP5 workflow by documenting the implemented CECCM software components and their release as software assets, including the repositories made available through GitHub and, where applicable, as open source.

D5.4 directly builds on the earlier WP5 deliverables. D5.1 [1] defined the initial integration plans, described the testbeds, and established the use-case-driven integration approach for exercising the AC3 functional

© AC3 2023

components across UC1, UC2, and UC3. D5.2 [2] reported the intermediate implementation and integration status of the CECCM components, mainly in relation to T5.1 [4] and T5.2 [4], while D5.3 [3] is dedicated to experimentation, validation, and KPI-oriented assessment. In this sequence, D5.4 does not repeat planning, integration-status, or evaluation material; instead, it consolidates the final software-facing outcome of those activities by reporting the implemented components, their repositories, and their availability conditions.

This deliverable is also linked to the technical work carried out in WP2, WP3, and WP4. The CECCM architectural framework and use case requirements were defined in D2.1 [5], D2.3 [6], and D2.4 [7], while the mechanisms later implemented as software artefacts were specified in the WP3 and WP4 deliverables, including D3.1 [12], D3.3 [8], D3.4 [9], and D4.1 [10] and D4.2 [11]. Therefore, D5.4 acts as the final bridge between those design and mechanism-oriented activities and the actual software repositories delivered by the consortium. It also has a practical connection with WP6, since the software availability reported here contributes to dissemination, accessibility, and post-project exploitation through the AC³ website and related project outputs.

2.3 Mapping AC³ Outputs

The purpose of this section is to map AC³ Grant Agreement (GA) commitments, both in terms of the formal deliverable description and the relevant task descriptions, 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			
D5.4 – Final Report on components and testbed integration: *"Report on the implementation of the CECCM and as software release of the different components. Parts of this software release will be also released as open source."*			
TASKS			
Task T5.1: AC³ components integration	*"Each partner involved will develop their individual components and/or functions and show their project results based on the tests done in their labs, where all essential functions can be tested."*	Sections 4.1, 4.2, 5.1, 5.2, 5.3 and 5.4	Section 4 provides the software inventory and the detailed per-component reporting of the implemented CECCM modules, including GUI, OSR, catalogues, LMS, monitoring, zero-touch management, and data management software. Sections 5.1–5.3 show how these implemented modules are combined in the final UC-level software deployments, while Section 5.4 highlights cross-use-case software reuse and therefore the consolidation of CECCM software across the project.

© AC³ 2023

Task T5.2: Testbed integration	*"This task will concern the integration of the software and hardware needed to run the three UCs."*	Sections 5.1, 5.2 and 5.3	The three "Final Integrated State" sections document the software deployed for each UC, the domain distribution across cloud, edge, far-edge, and data-source environments, the configuration adjustments applied, and the deployment/replication guidance. In this way, they provide the final software-facing account of how the testbed environments host the integrated AC^3 components needed to run the three use cases.

2.4 Software Availability, Release Model and Repository Visibility

Most software components described in this deliverable are hosted within the official AC^3 GitHub organization:

- AC3 GitHub Organization: https://github.com/EU-AC3/

The AC3 GitHub organization serves as the principal entry point for the majority of AC^3-related codebases, supporting repository-level traceability, version control, and long-term accessibility for the European Commission, authorised reviewers, consortium partners, and, where applicable, the wider research and open-source community. A limited subset of components is hosted outside the AC^3 GitHub organization in repositories managed by the responsible partners or by external open-source communities. These repositories remain governed by their respective maintainers, licensing terms, and access policies.

To avoid ambiguity, this deliverable distinguishes between three separate concepts: release model, repository visibility, and software License.

The release model describes how the software is released or made available by the project or by the responsible partner. The release model may be one of the following:

I. **Open-Source Software (OSS):** The software is released as open-source software under the License indicated in the corresponding component description. OSS components are intended to support transparency, reuse, and further research or community uptake.
II. **Internal:** The software is made available under restricted conditions, typically to consortium partners, authorised reviewers, or selected stakeholders. This release model may be used where partner exploitation strategies, intellectual property considerations, or organisational policies require controlled access.

The repository visibility describes the technical access level of the repository. A repository may be:

I. **Public:** The repository is publicly accessible without requiring explicit invitation.
II. **Private:** The repository is hosted under restricted access and requires GitHub authentication and explicit authorisation.

The software License defines the legal conditions under which the software may be used, modified, and redistributed by users who have access to the repository. Repository visibility and License are independent dimensions. Therefore, a private repository may still apply an open-source License to the software once access

© AC3 2023

is granted, or when the repository is later made public. Conversely, public availability of a repository does not remove the need to comply with the applicable License.

Accordingly, each component report in Section 4 includes a dedicated "Repository Information" block specifying the repository name, URL, release model, repository visibility, License, and main branch. This structure ensures that reviewers can clearly distinguish between the openness of the software, the accessibility of the repository, and the legal terms governing reuse.

2.4.1 Software Access and Release Categorisation

The AC^3 software portfolio follows a mixed availability model reflecting the different maturity levels, exploitation strategies, and ownership conditions of the components delivered by the consortium. This model combines public OSS repositories, restricted internal repositories, and externally hosted open-source repositories.

For clarity, the following combinations may appear in the component descriptions:

- **OSS + Public + License:** The software is publicly available and released under an open-source License.
- **OSS + Private + License:** The software applies an open-source License, but repository access is restricted during the project review or handover period. Access may be granted to authorised stakeholders.

This distinction is important because repository visibility does not define the License. Repository visibility only indicates whether the code can be technically accessed without invitation. The License defines the legal conditions for using the code once access has been granted.

Access to private or invitation-only repositories is coordinated through the project coordinator and the responsible component owner. Public repositories remain accessible through the AC^3 GitHub organization or through the external repository indicated in the component description.

2.4.1.1 Repository Ownership and Autonomy

Although the repositories are grouped under a common AC^3 GitHub organization, technical responsibility remains with the corresponding partner. As foreseen in the D5.4 structure, each repository may follow its own release schedule, licensing model, and visibility settings. The project therefore ensures a common access point and common traceability, but not a single uniform software publication policy across all components.

2.4.1.2 Continuity beyond the project

The software availability model also supports post-project continuity. The GitHub organization provides a stable reference point for the software assets reported in D5.4, while long-term maintenance and future release decisions remain under the responsibility of the owning partners, in line with the final project data management and exploitation framework.

2.5 Deliverable Overview and Report Structure

In this section, a brief description of the deliverable structure is provided, as follows:

- **Section 3 - AC^3 GitHub Organization Overview** presents a synopsis of the AC^3 GitHub organization as the project-level entry point to the software reported in this deliverable. It describes the overall repository organization, the logic used to group repositories, and the general versioning and release approach adopted for the software assets delivered within AC^3. This section provides a system-level view of software availability, rather than a component-by-component technical description.
- **Section 4 – Final Report on AC^3 Components** provides the final report on AC^3 components. It starts with a software inventory summary table giving reviewers a consolidated view of the implemented modules,

© AC3 2023

their repositories, responsible partners, release model, repository visibility, licensing information, related AC³ functions, and use-case relevance. It then presents a detailed component-level report covering the software modules made available through GitHub, including the Application Gateway, OSR, catalogues, LMS-related components, monitoring components, zero-touch configuration and application management components, and data management components.

- **Section 5 – Final Software Integration per Use Case** presents the final software integration state for UC1, UC2, and UC3, following a common structure based on deployed modules, topology, configuration, deployment guidance, and operational status. Section 5.4 additionally highlights software reuse across use cases.
- **Section 6 – Software Governance and Post-Project Sustainability** addresses software governance and post-project sustainability. It describes the repository ownership model, the open source and access policy applied across the software assets, and the expected continuity of the delivered repositories beyond the formal end of the project. This section complements the technical reporting by clarifying the governance and maintenance perspective associated with the software release.
- **Section 7 - Conclusions** summarise the key outcomes of the deliverable and reiterate its role as the final WP5 report on implemented components and software availability, in line with the Grant Agreement description of D5.4 as a report on the CECCM implementation and software release of the different components.

This structure ensures a logical flow from the reporting of individual software components to their final integration in the AC³ use cases and the governance model supporting their future accessibility. Overall, the deliverable provides a consolidated view of the final CECCM software implementation, its use-case relevance, and its availability within the AC³ framework.

© AC³ 2023

3 AC³ GitHub Organization Overview

This section provides a system-level overview of how the AC³ software release is organised in the AC³ GitHub organization. This section explains the overall technical structure through which the delivered software is exposed, grouped, and traced across the project. This organization-level view complements the component-by-component reporting of Section 4 and the use-case-oriented software integration view of Section 5.

3.1 Global AC³ GitHub Organization Structure

The EU-AC3 GitHub Organization (https://github.com/EU-AC3/) serves as the central access point for the software developed within the project. Rather than being implemented as a single codebase, the AC³ software release is organised as a collection of dedicated repositories, typically with one repository corresponding to each software component or module. Collectively, these repositories cover the three AC³ planes (User, Management, and Infrastructure), as well as the software supporting the validation use cases.

The GitHub organization follows a repository-centric model, in which each software component is maintained in its own repository. This approach ensures clear traceability between individual components, their functional roles within the AC³ framework, the responsible partners, and their contribution to specific use cases. The present deliverable complements this structure by providing the necessary cross-references between repositories and the corresponding AC³ functional blocks.

At a high level, the repositories can be grouped according to their functional role: (i) core cross-use-case components, which provide shared CECCM capabilities, such as the GUI, OSR, monitoring, and AI-based profiling modules; (ii) domain- or use-case-specific management components, which support cloud, edge, far-edge, and networking environments; (iii) data management components, including catalogues, connectors, mappers, and manipulators; and (iv) use-case aggregation repositories, which serve as integration entry points for UC1, UC2, and UC3.

This grouping reflects the AC³ functional decomposition defined in WP5 while remaining aligned with the project's use-case-driven integration strategy. Some components are reused across multiple use cases, whereas others are specific to individual deployment scenarios. The GitHub organization, therefore, balances modularity with integration needs, reflecting the practical implementation of the AC³ architecture.

Repository naming generally reflects the functional role of the component and, where relevant, its use-case association, although no single naming convention is enforced across all repositories.

At the organizational level, no unified folder hierarchy is imposed across repositories. Instead, each repository is designed to be self-contained, including its own source code, configuration files, deployment descriptors, container artefacts, and documentation.

To ensure a minimum level of consistency, repositories are expected to include a standard set of elements:

- **README.md**: component description, installation guidelines, and usage information
- **LICENSE**: definition of the applicable access and licensing model
- Dependency files (e.g., requirements.txt, pom.xml, go.mod)
- **Dockerfile**, where applicable, to support containerised deployment

Overall, the GitHub organization supports two complementary perspectives: (i) a functional view, structured around AC³ software families, and (ii) a use-case integration view, reflecting the final composition of UC1, UC2, and UC3. This dual perspective enables the software to be understood both as a set of reusable components and as an integrated system spanning the project use cases.

© AC³ 2023

3.2 Versioning and Release Strategy

The AC^3 GitHub organization adopts a decentralised approach to versioning and release management, whereby each repository is maintained independently by the responsible partner.

As a result, no single uniform versioning or release strategy is enforced across all components. Instead, individual repositories follow practices that best align with their respective development workflows, levels of maturity, and intended usage.

In general, the following conventions are commonly applied:

- **Versioning approach:** Several repositories adopt semantic versioning principles (e.g., v1.0.0), while others rely on simpler or incremental version identifiers, depending on the nature of the component.
- **Branching model:** Branching strategies are defined at the repository level. Typical approaches include the use of a primary branch (e.g., main or master) for stable code, complemented by development or feature branches as required.
- **Release tagging:** Where applicable, releases are identified through Git tags to denote stable versions or significant milestones. However, the use of formal release processes is not consistent across all repositories.
- **Container image tagging (where applicable):** For components distributed as containerised services, image-tagging conventions (e.g., latest or version-specific tags) are defined individually by the responsible partners and may therefore vary between repositories.

This approach provides the necessary flexibility for partners to manage their components according to their internal practices, while ensuring that the software artefacts reported in this deliverable remain accessible through the AC^3 GitHub organization. The versions, branches, tags, and repository states reported in this deliverable correspond to the final WP5 software handover snapshot. Because AC^3 follows a decentralised component ownership model, each repository may use its own versioning, tagging, and release-management practices. Where available, the relevant branch, tag, or stable version is reported in the corresponding component-level “Repository Information” block.

© AC3 2023

4 Final Report on AC³ Components

This section provides a consolidated overview of all AC³ software components delivered through the project AC³ GitHub organization. It is intended to offer a clear and immediate entry point to the software assets, enabling quick identification of each module, its repository, ownership, and role within the AC³ framework.

4.1 Software Inventory Summary

The software inventory is structured as a master table, where each row corresponds to a specific software module. The table establishes traceability between the implementation repositories, the AC³ functional architecture, and the validation use cases.

For each component, the table reports the software module name, repository URL, owner or maintainer, release model, repository visibility, License, AC³ function, use-case relevance, and the detailed section where the component is described. The release model, repository visibility, and License are intentionally reported as separate fields. This distinction ensures that readers can clearly identify whether a component is open source, whether the repository is publicly accessible or restricted, and which legal terms apply to the software.

The information included in Table 2 has been provided and validated by the responsible partners for each component.

Table 2: Consolidated overview of all AC3 software components.

Software Module	Repository URL	Owner	Release model	Repository Visibility	License	AC3 Function	Used in UC(s)	Detailed Section
GUI	https://github.com/EU-AC3/gui-core-uc1-uc2-uc3	FIN	Internal	Private	Fingletek Public License, Version 1.0	Web-based entry point for managing application profiles and triggering deployments.	All	4.2.1
OSR	https://github.com/EU-AC3/osr-core-uc1-uc2-uc3	FIN	Internal	Private	Fingletek Public License, Version 1.0	Semantic engine translating application profiles into platform-specific deployment descriptors.	All	4.2.2
Data and Service Catalogue	https://github.com/EU-AC3/catalogue-ui	ION/SPW	OSS	Public	Apache License Version 2.0	GAIA-X-compatible catalogue including information for the available Data Sets and Services used to deploy the UC application components.	UC1 UC3	4.2.3
Resource Exposer	https://github.com/EU-	EUR	Internal	Private	Custom License	Framework for Efficient Service and Resource	All	4.2.4

	AC3/resource-exposer-core-uc1-uc2-uc3					Orchestration in the Cloud-Edge Continuum.		
Resource Discovery	https://github.com/EU-AC3/discovery	EUR	Internal	Private	Custom License	Collects and aggregates resource information from multiple Resource Exposer instances and exposes aggregated information through a centralized API.	All	4.2.5
LiSO	https://github.com/EU-AC3/lcm-liso-uc1-uc2	EUR	Internal	Private	Custom License	MEC-aligned orchestrator for managing edge application lifecycles and traffic redirection.	UC1 UC2	4.2.6
MAESTRO	https://labs.etsi.org/rep/osl/hypo	UBI	OSS	Public	Apache License Version 2.0	Manages the lifecycle of microservice-based applications in UC3.	UC3	4.2.7
Migration Algorithm UC1	https://github.com/EU-AC3/migration-algorithm-uc1	IQU	Internal	Private	Apache License Version 2.0	Decision-making module for relocating applications between cloud and edge domains.	UC1	4.2.8
Migration Algorithm UC2	https://github.com/EU-AC3/migration-algorithm-uc2	EUR	Internal	Private	Custom License	This is the checkpoint/restore operator for the stateful microservices.	UC2	4.2.9
Migration Operator	https://github.com/EU-AC3/migrat	EUR	OSS	Public	Apache License Version 2.0	Kubernetes operator for container checkpointing and runtime state	UC2	4.2.10

	or_operator					preservation during migration.		
Horizontal Autoscaling Components	https://github.com/EU-AC3/autoscaling-ac3	IBM/RHT	Internal	Private	Apache License Version 2.0	AI-driven system scaling replicas based on predicted workload requirements. Controller for intelligent workload placement across federated computing environments.	UC3	4.2.11
Scheduler (P2CODE) UC3	https://github.com/rh-waterford-et/p2code_scheduler_operator	UBI/RHT	OSS	Public	Apache License Version 2.0	Controller for intelligent workload placement across federated computing environments.	UC3	4.2.12
AI-based Resource Predictor	https://github.com/EU-AC3/ai-resource-predictor	IBM	Internal	Private	Apache License Version 2.0	Predicts application execution behaviour to support proactive autoscaling and resource-management decisions.	UC3	4.2.13
AI-based Application Profile	https://github.com/EU-AC3/ai-application-profile-core-uc1-uc2-uc3	FIN	Internal	Private	Fingletek Public License, Version 1.0	Supports predictive capacity planning, application profiling, and policy-driven deployment decisions.	UC1 UC2 UC3	4.2.14
Data Provider Connector UC1	https://github.com/EU-AC3/data-management	SPW	OSS	Public	Apache License Version 2.0	EDC-based provider-side connector exposing IoT data from the UC1 data-source domain.	UC1	4.2.15

EDC Connector for IONOS S3 UC3	https://github.com/EU-AC3/data-provider-conector-uc3	RHT/ION	Internal	Private	Apache License Version 2.0	Implementation for bidirectional file transfer between IONOS S3 buckets.	UC3	4.2.16
Data Consumer Connector UC1	https://github.com/EU-AC3/data-management	SPW	OSS	Public	Apache License Version 2.0	Provides access to data exposed by the provider connector.	UC1	4.2.17
Data Mappers	https://github.com/EU-AC3/data-management	SPW	OSS	Public	Apache License Version 2.0	Transforms incoming data into formats required by downstream UC1 services.	UC1	4.2.18
Data Manipulator UC1	https://github.com/EU-AC3/use-case-1/tree/main/application/manipulator	SPW	OSS	Public	Apache License Version 2.0	AI/ML processing layer for edge-side anomaly detection and time-series forecasting.	UC1	4.2.19
Data Manipulator UC3	https://github.com/EU-AC3/data-manipulator-uc3	RHT	Internal	Private	Apache License Version 2.0	Backend managing distributed astronomy data processing and job orchestration.	UC3	4.2.20
Message Broker UC1	https://github.com/EU-AC3/data-management	SPW	OSS	Public	Apache License Version 2.0	RabbitMQ-based message routing and queue management for the UC1 edge pipeline.	UC1	4.2.21

Remark: Repository visibility and software License are reported separately. A repository may be private or access-controlled while still applying an open-source License to the software once access has been granted or if the repository is subsequently made public. Conversely, public repository visibility does not remove the obligation to comply with the applicable License. Repositories marked as private or accessible by invitation are not publicly accessible and may require GitHub authentication and explicit authorisation by the responsible partner or the AC^3 GitHub organization. Access for authorised reviewers and stakeholders can be coordinated through the project coordinator and the relevant component owner.

4.2 Detailed Component Report

This section provides the detailed reporting for each AC^3 software component delivered through the project GitHub organization. Each subsection is self-contained and includes the key information required to identify,

understand, access, and assess the corresponding software artefact, including its functionality, repository information, packaging and deployment approach, integration in AC^3, and documentation and maintenance status. Only software components that are available in GitHub at the time of preparation of this deliverable are included in the detailed report.

4.2.1 Application Gateway (GUI) Components

4.2.1.1 Functional Description

The Application Gateway, referred to as the GUI or Management Dashboard, serves as the primary user-facing entry point within the AC^3 software architecture. It provides a unified web-based interface through which end users interact with the backend components of the AC^3 ecosystem, including the Ontology and Semantic Reasoner (OSR), the Lifecycle Management (LCM) subsystem, monitoring capabilities, and external orchestration connectors. The GUI does not embed domain-specific business logic directly; rather, it functions as a control and interaction layer that delegates processing, reasoning, and orchestration tasks to the appropriate backend services via RESTful API calls.

The core functionalities implemented by the Application Gateway are as follows:

- **Application Profile Management:** The GUI enables users to create, view, update, and manage application profiles that describe CECC applications. A guided multi-step form allows users to define application metadata, microservice composition (including container images, resource requirements, exposed ports, and environment variables), data sources, network traffic and load characteristics, security and access control policies, deployment context, and application consumer specifications. Additionally, users may upload pre-existing application profile JSON files that conform to the CECC schema, with automatic structural validation performed client-side prior to persistence. Profiles are stored and retrieved through the Application Profile Service backend (MongoDB-based CRUD API).
- **Descriptor Generation and Editing:** Upon completing an application profile, the GUI invokes the OSR API to generate platform-specific deployment descriptors. These descriptors may be in JSON or YAML format, depending on the target platform. The GUI renders the generated output using syntax-highlighted viewers and, as of the most recent iteration, provides an inline code editor (based on Ace Editor) that allows users to manually inspect and modify the generated descriptor before deployment. This feature supports both JSON and YAML modes with real-time syntax validation, enabling advanced users to fine-tune deployment descriptors without leaving the interface.
- **Deployment Orchestration:** The GUI supports triggering deployment workflows to multiple orchestration platforms. For LiSO-based deployments (UC1-UC2), the interface submits the generated Network Service Descriptor (NSD) to the OSR API, which in turn forwards it to the LiSO Converter and subsequently to the LiSO NFVO for instantiation. For Maestro-based deployments (UC3), the GUI communicates with the Maestro Translator API through the OSR API layer. Deployment results, including NSD identifiers, service order identifiers, microservice instance details, and deployment states, are displayed to the user in structured card-based layouts. Users may also delete active deployments through the interface, with deletion requests routed to the appropriate platform-specific endpoints.
- **Application Monitoring and Management:** The Applications page provides a consolidated view of all deployed applications, grouped by deployment platform (LiSO or Maestro). Users can inspect individual deployment details, review microservice instance information, and access monitoring data when available. The GUI retrieves deployment records from the OSR API's deployment database and presents instance-level details including instance identifiers, regions, and status indicators.
- **Data Catalogue Management:** The Data Management module enables users to register, browse, and manage datasets and services within the Piveau data catalogue. The interface supports CRUD operations

© AC3 2023

for both dataset and service entries, with metadata fields including title, description, distribution format, and access URLs. Catalogue entries persisted through the Piveau catalogue API via the OSR backend.

- **Authentication and Session Management:** User authentication is handled through a Django-based backend service. The GUI implements login, registration, password reset, and session management flows. Authentication state is maintained via session cookies and communicated to the backend through standard Hypertext Transfer Protocol (HTTP) mechanisms with CORS and CSRF protections.

The relationship between the GUI and each AC³ function is summarised below:

- **Lifecycle Management (LCM):** The GUI triggers lifecycle operations (deployment, deletion) through the OSR API, which acts as the intermediary to the LCM layer (LiSO NFVO or Maestro Translator).
- **Ontology and Semantic Reasoner (OSR):** The GUI invokes the OSR API for descriptor generation (converting application profiles to platform-specific NSD formats), LiSO/Maestro deployment orchestration, and Piveau catalogue interactions.
- **Connectors / APIs:** The GUI communicates exclusively through HTTP REST APIs. It interacts with the Application Profile Service (Node.js/MongoDB), the Auth Backend (Django/MySQL), and the OSR API (Flask), which in turn connects to the LiSO Converter (FastAPI), the Avaruus Converter, the Maestro Translator, and the Piveau catalogue.

The GUI itself does not perform semantic reasoning, descriptor transformation, or orchestration logic. All such computations are delegated to the respective backend services, ensuring a clean separation between the presentation layer and the domain logic.

4.2.1.2 Repository Information

- **Repository name:** gui-core-uc1-uc2-uc3
- **URL (or access instructions):** https://github.com/EU-AC3/gui-core-uc1-uc2-uc3
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Fingletek Public License, Version 1.0 (provided in the repository; not a standard open-source License)
- **Main branch:** main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.1.3 Integration in AC³

Use Case Integration

The Application Gateway is integrated into the following AC³ Use Cases:

- **UC1-2 (LiSO-based deployment):** The GUI enables users to create application profiles, generate LiSO-compatible Network Service Descriptors (NSD) via the OSR, and trigger deployment to the LiSO NFVO. The full flow from profile creation through descriptor generation, optional manual editing, deployment triggering, and deployment status tracking is supported end-to-end through the GUI. The deployment details, including NSD identifiers, microservice instance mappings, and instance regions, are displayed upon successful deployment.
- **UC3 (Maestro-based deployment):** The GUI supports Maestro-based orchestration workflows through a parallel path. Application profiles can be submitted to the Maestro Translator API via the OSR, resulting

© AC³ 2023

in a Service Order. Deployment state tracking (COMPLETED, PARTIAL, PROCESSING) is reflected in the interface. Deletion of Maestro deployments is also supported.

Deployment Domain

The GUI is deployed in the cloud domain as a centralised management interface. It communicates with backend services that may themselves interact with edge and far-edge infrastructure through the LCM layer. The GUI itself does not execute on edge or far-edge nodes.

Inter-Component Interaction

The Application Gateway interacts with other AC^3 software modules exclusively through HTTP REST APIs. The interaction model is summarised below:

- **Northbound interface:** The GUI exposes a web interface (port 3000) to end users via a browser. In production, this is served through a Nginx reverse proxy with SSL termination.
- **Southbound interfaces:** The GUI makes outbound HTTP calls to three internal backend services (OSR API, Application Profile Service, Auth Backend). It does not communicate directly with external orchestration platforms; all such interactions are mediated by the OSR API.

All user actions originate from the GUI. The GUI does not implement background jobs, scheduled tasks, or event-driven processing. Every interaction follows a synchronous request–response pattern initiated by the user.

4.2.1.4 Documentation and Maintenance

Documentation for the Application Gateway is provided at multiple levels within the repository:

- **Front-End/README.md:** Contains setup instructions for the GUI component, including cloning, dependency installation, and local development server startup procedures. This README is oriented towards developers who wish to run the frontend locally during development**.**
- **Root README.md:** Provides an overview of the full monorepo and links to component-specific documentation (The GUI documentation is continuously updated as the project evolves, ensuring that users have access to the most current information.).
- **PRODUCTION_DEPLOYMENT.md:** A comprehensive production deployment guide covering server preparation, Docker installation, image building, Docker Compose deployment, Nginx configuration, SSL setup, health checks, monitoring, troubleshooting, and security considerations. This document is available on the lcm2.0-handover-cleanup branch.
- **docker-compose.example.yml and nginx-osr.example.conf:** Sanitized deployment configuration examples that serve as operational documentation for production setup.
- Inline code documentation is present in the form of descriptive variable names, component structure, and occasional code comments where non-obvious logic is implemented. Formal API documentation (e.g., OpenAPI/Swagger) for the GUI's consumed endpoints is not currently provided but is recommended as a future enhancement.

Maintainer

The Application Gateway is developed and maintained by FIN Oy as part of the AC^3 consortium. Development and issue resolution are handled internally by the FIN engineering team.

4.2.2 Ontology and Semantic-Aware Reasoner (OSR) Components

4.2.2.1 Functional Description

The OSR constitutes a central component within the AC³ software architecture, serving as the intelligent intermediary between the GUI and the downstream lifecycle management, orchestration, and data catalogue platforms. The OSR is responsible for semantic modelling of application structures, knowledge-driven transformation of application descriptors into platform-specific deployment artefacts, mediation of deployment and lifecycle operations across heterogeneous orchestration backends, semantic data cataloguing, and deployment tracking. Within the broader AC³ vision, the OSR fulfils the role of the knowledge-processing layer that bridges high-level application intent (expressed through application profiles) with the concrete operational actions required by the underlying cloud-edge continuum infrastructure.

Semantic Descriptor Translation and Validation

The primary function of the OSR is the transformation of application profiles — structured JSON documents describing an application's microservice composition, resource requirements, networking topology, SLA constraints, security policies, data sources, and deployment context — into platform-specific deployment descriptors. This translation process is semantically grounded: the OSR interprets the application profile's structure and content according to the AC³ application profile model, resolves references to external data and service catalogue entries, and produces normalised output in the appropriate format for the target orchestration platform.

For LiSO-based deployments (UC1 and UC2), the OSR performs a two-stage transformation. First, the application profile is translated into an intermediate application composition descriptor (YAML format) that captures microservice definitions, volume configurations, networking graphs, and SLA specifications in a platform-neutral representation. This intermediate descriptor is then forwarded to the LiSO Converter service, which produces a LiSO-compliant Network Service Descriptor (NSD) in JSON format conforming to the European Telecommunications Standards Institute (ETSI) SOL005-inspired schema used by the LiSO NFVO. The NSD includes appD (application descriptor) entries for each microservice, with swImageDescriptor fields specifying container images, resource requirements (numVirtualCpu, virtualMemSize), port mappings, and environment variable configurations.

For Maestro-based deployments (UC3), the OSR translates application profiles into YAML descriptors compatible with the Maestro Translator API, which subsequently generates Helm charts and service orders for multi-domain orchestration.

The OSR operates across two knowledge categories:

- Static knowledge encompasses the OWL ontologies defining metric taxonomies, valid operators, and permissible actions; the application profile model schema that governs the structure of input descriptors; the NSD schema expected by the LiSO NFVO; the Piveau JSON-LD and RDF Turtle vocabularies used for semantic data cataloguing (leveraging DCAT, Dublin Core, Gaia-X, and Schema.org namespaces); and the policy rule definitions that encode expected metric–condition–action mappings.
- Dynamic knowledge includes the runtime application profiles submitted by users through the GUI; the real-time deployment state retrieved from the LiSO NFVO (instance status, monitoring data, resource utilisation); Maestro service order states (PROCESSING, COMPLETED, PARTIAL, FAILED); and the deployment history maintained in the OSR's internal SQLite database, which tracks deployment records including user associations, platform identifiers (NSD IDs for LiSO, service order IDs for Maestro), microservice instance details, and temporal metadata.

Lifecycle Management Support

The OSR directly supports application lifecycle management by orchestrating the full deployment lifecycle across platforms:

© AC³ 2023

1. **Deployment**: The OSR manages the end-to-end deployment flow for LiSO, which includes NSD onboarding, lifecycle operation monitoring, service instance creation, instantiation with resource limits, and result retrieval. For Maestro, the OSR submits descriptors and tracks service order processing.
2. **Deletion**: The OSR supports deletion of deployed instances. For LiSO, this involves terminating the running instance, waiting for a graceful shutdown, offboarding the NSD, and confirming completion. For Maestro, deletion is performed via the service order API.
3. **Deployment tracking**: All deployment operations have persisted in the OSR's internal database, providing an auditable record of lifecycle events per user and application.

Data Catalogue Management

The OSR integrates with the Piveau data catalogue through two complementary components. The PiveauTransformer converts user-submitted form data into Piveau-compliant JSON-LD and RDF Turtle representations, using standard semantic web vocabularies (DCAT, Dublin Core, Gaia-X, Schema.org, vCard). The PiveauAPIClient manages CRUD operations against the Piveau REST API for both datasets and services, with authentication via API key. The DataCatalogueClient and ServiceCatalogueClient provide read access to the catalogue for resolving data source and service references during descriptor translation.

Operational Mode

The OSR operates as a standalone service within the AC³ deployment stack. It is deployed as an independent containerised microservice that exposes a RESTful API consumed by the GUI and, indirectly, by the LiSO Converter service (which is itself a separate microservice within the OSR deployment scope). The OSR does not embed within any other component; rather, it serves as the central orchestration and knowledge-processing hub through which the GUI interacts with all backend platforms.

4.2.2.2 Repository Information

- **Repository name:** osr-core-uc1-uc2-uc3
- **URL (or access instructions):** https://github.com/EU-AC3/osr-core-uc1-uc2-uc3
- **Release model:** Internal
- **Repository visibility:** Private
- **License**: Fingletek Public License, Version 1.0 (provided in the repository; not a standard open source License)
- **Main branch**: main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.2.3 Integration in AC³

Use Case Integration

The OSR is integrated into the following AC³ Use Cases:

1. UC1 and UC2 (LiSO-based Cloud-Edge Deployment): In UC1 and UC2, the OSR implements the full pipeline from application profile intake through NSD generation to LiSO NFVO deployment. The flow begins when the Application Gateway submits an application profile to the OSR API's /osr/api/translate-LiSO endpoint. The OSR translates the profile into an application composition YAML descriptor using the deployment_interface module, which extracts microservice definitions (container images, resource requirements, ports, environment variables), volume configurations, networking graphs, and SLA

specifications. This YAML is forwarded to the LiSO Converter (/convert), which produces a LiSO-compliant NSD JSON. The NSD is returned to the GUI for user review and optional editing. When the user confirms deployment, the GUI submits the NSD to /osr/api/deploy-LiSO, which invokes the LiSO SDK to execute the full deployment lifecycle: NSD on-boarding, lifecycle operation monitoring, service instance creation, instantiation with resource limits, and result retrieval. The OSR also supports instance querying (/osr/api/LiSO/instance/{id}), monitoring data retrieval (/osr/api/LiSO/instance/{id}/monitoring), and deployment deletion. The LiSO SDK implements intelligent routing between multiple LiSO NFVO endpoints based on application naming conventions (e.g., UC1-specific versus default infrastructure).

2. UC3 (Maestro-based Multi-Domain Orchestration): In UC3, the OSR translates application profiles into YAML descriptors and forwards them to the Maestro Translator API via the MaestroAPIClient. The Maestro API returns a service order identifier, which is tracked by the OSR. The OSR provides status monitoring (/osr/api/maestro/status/{serviceOrderId}) and deletion capabilities for Maestro deployments.
3. Data Management Integration: The OSR integrates with the Piveau data catalogue to support semantic data management within AC^3. Through the PiveauTransformer, the OSR converts user-defined dataset and service metadata into JSON-LD representations conforming to DCAT, Dublin Core, Gaia-X, and Schema.org vocabularies, and serialises them as RDF Turtle for persistence in the Piveau triple store. This integration supports the data governance and cataloguing requirements of the AC^3 data space.

Deployment Domain

The OSR API and LiSO Converter services are deployed in the cloud domain as centralised services. The OSR communicates with infrastructure that spans cloud and edge domains through the LiSO NFVO and Maestro translator APIs, which manage the actual placement of application instances across the cloud-edge continuum. The OSR itself does not execute on edge or far-edge nodes.

Inter-Component Interaction

The OSR interacts with other AC^3 software modules through HTTP REST APIs.

Input data consumed by the OSR:

1. Application profiles (JSON) from the GUI, describing application structure, microservices, resources, networking, SLAs, and deployment context.
2. NSD JSON documents from the GUI for deployment operations (after user review/editing).
3. Dataset and service metadata from the GUI for Piveau catalogue operations.
4. Runtime instance data from the LiSO NFVO (instance status, monitoring metrics, deployment state).
5. Service order status from the Maestro Translator API.

Output artefacts produced by the OSR:

1. Application composition YAML descriptors (intermediate platform-neutral format).
2. LiSO-compliant NSD JSON (via the LiSO Converter) conforming to ETSI-inspired schemas.
3. Maestro-compatible YAML descriptors for multi-domain orchestration.
4. Piveau-compliant JSON-LD and RDF Turtle documents for semantic data cataloguing.
5. Deployment records (stored in SQLite) providing an auditable lifecycle history.
6. Deployment status, instance details, served to the GUI.

Contribution to closed-loop control and intelligent automation:

The OSR contributes to closed-loop control within AC^3 through several mechanisms. The descriptor translation pipeline automates the conversion from high-level application intent to platform-specific deployment artefacts,

© AC3 2023

eliminating manual descriptor authoring. The LiSO SDK implements automated lifecycle sequencing (on-board → create instance → instantiate → monitor → report), with built-in error handling for blocked states (e.g., instances stuck in FAILED or NOT_INSTANTIATED states). The deployment tracking database enables state-aware operations, including automatic deletion-before-redeployment to handle idempotent deployment scenarios. The ontology-based policy validation framework provides the foundation for automated constraint checking, enabling future integration of monitoring-driven policy re-evaluation into the orchestration loop. The monitoring data retrieval endpoints provide the observability interface through which the GUI and future automation components can assess deployment health and trigger corrective actions.

4.2.2.4 Documentation and Maintenance

Documentation

Documentation for the OSR is provided at the following locations within the repository:

- OSR/readme.md: Contains basic startup instructions for the OSR API service. Due to the operational focus of the current codebase, this README is minimal and references the Flask entry point.
- Inline source documentation: The OSR codebase includes docstrings for key modules and functions. The LiSO SDK (LiSO_sdk.py) contains detailed module-level documentation describing the deploy and delete workflows. The Piveau Transformer includes comprehensive docstrings for the JSON-LD transformation logic. API client classes (MaestroAPIClient, PiveauAPIClient) document request/response formats.
- Environment variable documentation: An env.example file is provided (on the cleanup branch) documenting all required and optional configuration parameters with placeholder values and descriptions.
- Formal API documentation (e.g., OpenAPI/Swagger) is available for the LiSO Converter service (FastAPI auto-generates this at /docs). OpenAPI documentation for the OSR Flask API is identified as a future enhancement.

Maintainer

The OSR is developed and maintained by Fingletek Oy as part of the AC^3 consortium. Development, integration, and issue resolution are handled internally by the Fingletek engineering team.

4.2.3 Data and Service Catalogue Component

4.2.3.1 Functional Description

The Catalogues component implements a GAIA-X-compatible data and Service Catalogue that stores, validates, and exposes metadata for the datasets and deployable services available within the AC^3 application ecosystem. It is built on the Piveau (https://piveau.io) open source platform developed by Fraunhofer FOKUS, and extended with AC^3-specific SHACL validation shapes and a custom catalogue configuration that aligns the metadata model with GAIA-X, DCAT-AP, Dublin Core, Schema.org, and the AC^3 ontology namespace (`https://ac3-project.eu/#`).

The catalogue manages two categories of resources:

- **Datasets (*gx:DatasetShape*):** A dataset entry describes a data source available within the AC^3 ecosystem. The schema captures human-readable title, name and description, License (constrained to SPDX identifiers), contact point (name, email), geographic coordinates, asset identifier, endpoint description, endpoint URL, the required EDC connector service offering (*ac3:requiredConnector*), and optionally a required service offering (*ac3:requiredServiceOffering*). This structure allows data sources to be precisely identified, located, and referenced during application composition, including the connector chain required to access them.

- **Services (*ac3:ServiceShape*):** A service entry describes a deployable microservice component. The schema captures title, name, description, container image reference, environment variables (key=value format), exposed ports (*host:container* format), volume definitions, container resource limits (using GAIA-X ContainerResourceLimits), and microservice SLA constraints including service availability (0–100%), maximum response time (Low / Medium / High), and data throughput (Low / Medium / High). Service entries therefore serve as reusable, semantically described building blocks that can be referenced by name during application descriptor generation.

Within AC^3, the Catalogues component is primarily related to the following functions:

- **OSR**: The OSR integrates with the catalogue through a PiveauAPIClient for CRUD operations and through dedicated DataCatalogueClient and ServiceCatalogueClient interfaces for resolving data source and service references during application descriptor generation. When a user creates or updates a dataset or service entry through the GUI, the OSR's PiveauTransformer converts the submitted metadata into Piveau-compliant JSON-LD and RDF Turtle representations and persists them in the catalogue triple store.
- **GUI**: The GUI exposes a Data Catalogue Management module that allows users to register, browse, update, and delete both dataset and service entries via the OSR API. Catalogue content is therefore visible and editable from the user-facing interface without requiring direct access to the catalogue backend.
- **LCM**: Catalogue service entries are referenced within the application descriptor generated by the OSR. LiSO consumes these descriptors to instantiate microservices on the target LMS environments. The catalogue thus provides the reusable service inventory from which application compositions are assembled and deployed.

4.2.3.2 Repository Information

- **Repository name:** catalogue-ui
- **URL:** https://github.com/EU-AC3/catalogue-ui
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** main

AC^3 DCAT-AP configuration repository (catalogue schema and platform configuration)

- **Repository name:** dcat-ap
- **URL:** https://github.com/EU-AC3/dcat-ap
- **Release model:** OSS
- **Repository Visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** main

The dcat-ap repository provides the AC^3-specific assets required to configure and operate the catalogue: the dataset-shape.ttl and service-shape.ttl SHACL shape files that define the metadata schemas for datasets and services, and the piveau.json catalogue configuration file that registers these shapes as the two resource types (dataset and service) of the gaia-x catalogue instance.

4.2.3.3 Integration in AC^3

Integrated in UC1 and UC3

© AC3 2023

The Catalogue component is integrated in UC1 and UC3.

In UC1, the catalogue stores the metadata for the data sources (IoT sensor streams from the Data-Source-UC1 domain) and for the deployable microservices that compose the UC1 application pipeline (Data Provider Connector, Data Consumer Connector, Message Broker, Data Mappers, Data Manipulator). The Catalogues complement the Data-Source-UC1 domain by storing the corresponding data-source metadata and service descriptions, allowing the dataset and its required connector services to be discovered and referenced during application composition.

In UC3, the catalogue provides equivalent service and data descriptions for the UC3 application components, supporting the same OSR-driven descriptor generation and deployment workflow.

Deployment domain

The Catalogue component is deployed in the Cloud domain as a centralised service. It does not execute on edge or far-edge nodes. Its REST API endpoints are accessible to the OSR API, which is also deployed in the cloud domain, and to external clients through the management network.

Interaction with other AC3 software modules

The Catalogues component interacts with the following AC3 software modules:

- **OSR**: The OSR is the primary integration point for the catalogue. It writes catalogue entries (datasets and services) via the PiveauAPIClient using the Piveau hub-repo REST API. It reads catalogue entries via the DataCatalogueClient and ServiceCatalogueClient during descriptor translation, resolving data source references and service definitions to populate the application composition YAML and, subsequently, the LiSO NSD. The OSR's PiveauTransformer serialises metadata as JSON-LD and RDF Turtle using DCAT, Dublin Core, GAIA-X, Schema.org, and vCard vocabularies before submitting it to the catalogue.
- **GUI**: The GUI exposes the catalogue management functionality to end users through its Data Catalogue Management module. All GUI interactions with the catalogue are mediated by the OSR API; the GUI does not communicate directly with the Piveau hub-repo or hub-search endpoints.
- **LiSO (LCM)**: LiSO consumes application descriptors generated by the OSR. These descriptors include microservice definitions derived from catalogue service entries. The catalogue therefore contributes indirectly to lifecycle management by providing the reusable service descriptions that are translated into NSD appD entries by the LiSO Converter.

4.2.3.4 Documentation and Maintenance

Documentation location

README.md files in the code repositories provide documentation for all contributions.

The Catalogue components of AC^3 are developed and maintained by Spark Works (SPW) as the primary responsible partner, in coordination with IONOS (ION) for catalogue infrastructure hosting within the AC^3 project. Development, configuration, and issue resolution are handled internally by the Spark Works team.

Issue tracking mechanism

GitHub repository issues are used to track problems, supplemented by consortium-internal coordination channels.

4.2.4 Resource Exposer

4.2.4.1 Functional Description

The Resource Exposer is a service designed to collect infrastructure metrics from a monitoring source—typically Prometheus—and expose them through a standardized API interface. These exposed metrics describe computing, networking, energy, and infrastructure characteristics of a cluster.

The main goal of the Resource Exposer is to make infrastructure metrics easily accessible to other system components involved in orchestration, scheduling, or decision-making processes. By transforming raw monitoring data into structured API endpoints, the service enables higher-level platforms to retrieve information about: Compute resources (CPU, memory, disk), Network performance (latency, jitter, packet loss, throughput), Infrastructure characteristics, Energy type associated with the deployment environment

The Resource Exposer plays an important role in resource and service discovery across the cloud–edge continuum, facilitating efficient orchestration of distributed services.

The Resource Exposer repository contains the implementation of the service along with deployment and configuration resources.

Typical repository structure:

- Application Source Code
  - **app.py:** Main application entry point
  - Supporting Python modules implementing metric collection and API endpoints
- Dependencies
  - **requirements.txt:** Python dependencies required by the application
- Deployment Resources
  - **deploy/helm/:** Helm chart for Kubernetes deployment
  - **deploy/kubernetes/:** Raw Kubernetes manifests for manual deployment
- Containerization
  - **Dockerfile:** Container image definition used to build the Exposer runtime environment

The application is written in Python and relies on external libraries for metric collection and communication with monitoring systems such as Prometheus.

4.2.4.2 Repository Information

- **Repository name:** Exposer
- **Repository URL:** https://github.com/EU-AC3/resource-exposer-core-uc1-uc2-uc3
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Custom Proprietary License
- **Main branch:** main
- **Current stable tag:** exposer: master

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

© AC3 2023

4.2.4.3 Integration in AC³

The Resource Exposer is deployed within each infrastructure domain (such as cloud or edge clusters) and is responsible for collecting local infrastructure metrics from monitoring systems and resource managers. These metrics may include compute resources, network performance, and infrastructure characteristics, which are then exposed through standardized APIs. In this way, the Resource Exposer acts as an interface between the local infrastructure and higher-level components that need access to resource information. The Discovery service complements this functionality by aggregating the data provided by multiple Resource Exposers deployed across different infrastructures. It maintains a registry of the available exposers and periodically retrieves their metrics, providing a global view of the available resources across the system and enabling other entities to query infrastructure capabilities by region or resource type. The technical implementation and deployment details of these components are described in Deliverable 4.2 [11], while further architectural details can be found in [9].

4.2.4.4 Documentation & Maintenance

The repository is maintained by EUR. Issues and updates are managed through the repository issue tracker and consortium-internal coordination channels.

4.2.5 Resource Discovery

4.2.5.1 Functional Description

The Discovery Microservice is responsible for collecting and aggregating resource information from multiple Resource Exposer instances deployed across the infrastructure.

Each Resource Exposer provides metrics about its local infrastructure (compute resources, network performance, energy characteristics, etc.). The Discovery service maintains a registry of these exposers, periodically retrieves their metrics, and exposes aggregated information through a centralized API.

This service enables higher-level orchestration systems to obtain a global view of the available infrastructure across multiple clusters or regions.

The Discovery Microservice provides the following capabilities:

- **Exposer registration and management:** Allows Resource Exposer instances to dynamically register and unregister themselves.
- **Metrics aggregation:** Collects and aggregates compute and network metrics from all registered exposers.
- **Region-based queries:** Supports filtering metrics based on geographic or logical regions.
- **Centralized access point:** Provides unified API endpoints to retrieve infrastructure metrics.
- **Health monitoring:** Allows other services to verify the operational status of the discovery system.

The Discovery repository contains the source code and deployment resources required to run the service.

The main components of the repository include:

Application code:

- app/__main__.py – Main entry point of the microservice
- Supporting modules responsible for exposer registration and metrics aggregation

Dependencies:

- requirements.txt – Python dependencies required by the application

Containerization:

© AC³ 2023

- Dockerfile – Defines the container image used to run the Discovery service

Deployment resources:

- kubernetes/discovery.yaml – Kubernetes manifests for deploying the service

The service is implemented in Python and exposes a REST API for interacting with the resource discovery system.

4.2.5.2 Repository Information

- **Repository name:** Discovery
- **Repository URL:** https://github.com/EU-AC3/discovery
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Custom Proprietary License
- **Main branch:** main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.5.3 Integration in AC³

The Discovery Microservice is responsible for collecting and aggregating resource information from multiple Resource Exposer instances deployed across the infrastructure.

4.2.5.4 Documentation and Maintenance

The repository is maintained by EURECOM. Issues and updates are managed through the repository issue tracker and consortium-internal coordination channels.

4.2.6 Lightweight Edge Slice Orchestration (LiSO) Components

4.2.6.1 Functional Description:

LiSO is a Multi-access Edge Computing (MEO) orchestrator aligned with ETSI MEC 003/010-2 standards. It manages the full lifecycle of MEC applications using Application Descriptors (AppD) to define container images, resource requirements, and traffic redirection policies.

The project is implemented as an HTTP server that exposes a comprehensive set of high-level RESTful APIs, giving users full control over all orchestration operations. These APIs are grouped into functional categories:

- **Resource Management** – Check resource availability and reserve computational/storage resources for a service.
- **Service Onboarding** – Onboard, retrieve, enable, and manage service packages (NSDs).
- **Service Instantiation** – Create, delete, instantiate, update, query, and migrate service instances and individual applications.
- **Service Termination** – Terminate running network service instances.
- **Service Offboarding** – Disable and delete service packages.
- **Monitoring** – Start, retrieve, and terminate monitoring of service instances.
- **Logging** – Start, retrieve, and terminate logging for service instances.
- **Lifecycle Operation Status** – Query the status of any ongoing lifecycle operation (Instantiation, Termination, Onboarding …).

These APIs enable seamless interaction with the orchestrator, covering everything from resource reservation to full lifecycle management, monitoring, and logging. For development and testing, LiSO includes example

© AC³ 2023

application packages and a Swagger UI for API exploration.

The LiSO repository is organized into several top-level directories, each serving a distinct purpose in the project. Below is a high-level overview of its structure, followed by a detailed breakdown of the most critical component—the application folder.

The LiSO repository is organised into the following main directories:

- **application:** contains the core orchestrator logic, including the HTTP server, controllers, plugins, and data models.
- **deployment**: contains the Kubernetes manifests used to deploy LiSO on edge clusters.
- **example**: contains sample Network Service Descriptor (NSD) files and application package examples.

test/: contains end-to-end test scenarios used to validate orchestrator functionality.

Technical Architecture & Structure

Instead of an exhaustive file list, the repository is organized into ' functional domains:

1. **Core Logic (/application):** The "brain" of the orchestrator, built as a Python-based HTTP server.
   - **Controllers:** Implements ETSI-aligned REST APIs for resource reservation, onboarding, and migration.
   - **Plugins:** Modular drivers for Lifecycle Management (LCM), Network (traffic steering), and Telemetry (monitoring).
   - **State Store:** Uses an internal SQLite cache (cachedns.db) for DNS state and MongoDB for persistent deployment metadata.
2. **Deployment Framework (/deployment):** Ready-to-use manifests for cloud-edge environments:
   - **Kubernetes/K3s:** Standard YAMLs including a pre-configured VIM layer (DNS, Private Registry, and VIM Manager).
   - **OpenShift:** Optimized configurations using **Red Hat UBI 8** base images, BuildConfigs, and Security Context Constraints (SCC).
3. **Configuration Strategy:** LiSO utilizes a **Zero-Touch** configuration model to decouple code from infrastructure:
   - **Region Config (ConfigMaps):** JSON files defining VIM/MEP endpoints for different geographic clusters.
   - **Dynamic Variables:** Environment variables manage database credentials (DATABASE_HOST), logging levels, and custom DNS resolution behavior (USE_CUSTOM_DNS_RESOLVER).

4.2.6.2 Repository Information

- **Repository name:** lcm-LiSO-uc1-uc2
- **Repository URL:** https://github.com/EU-AC3/lcm-liso-uc1-uc2
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Custom Proprietary Research License
- **Main branch:** main
- **Current stable tag:** LiSO:main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.6.3 Integration in AC³

LiSO is a Multi-access Edge Computing (MEC) Orchestrator (MEO) aligned with ETSI MEC 003 and ETSI MEC 010-2 standards. It is purpose-built to orchestrate application packages at the network edge, enabling traffic redirection from the core network or cloud to edge-hosted applications. LiSO manages the full lifecycle of MEC applications (equivalent to VNFs in NFV) using application descriptors (AppD) that define container images, resource requirements, DNS rules, and traffic redirection policies. It employs Kubernetes as its Virtual Infrastructure Manager (VIM) and supports complex deployments with extended AppD fields for Network Service Descriptors (NSD), allowing instantiation of multiple applications. LiSO is compatible with OpenShift, Kubernetes, and K3s.

4.2.6.4 Documentation & Maintenance

The repository is maintained by EURECOM. Issues and updates are managed through the repository issue tracker and consortium-internal coordination channels.

4.2.7 MAESTRO

The MAESTRO service orchestration platform, developed by UBI, functions as the primary Application Lifecycle Management (LCM) engine for Use Case 3 within the AC^3 framework. It provides a modular and standards-based approach to deploy, manage, and continuously adapt containerised applications across a federated cloud-edge environment. By employing TM Forum open APIs, MAESTRO coordinates complex service deployments and translates high-level application descriptors into concrete infrastructural actions.

REMARK: Since January 14, 2026, UBITECH's Maestro has officially joined the ETSI community as a new Module Development Group (MDG). As part of this transition, Maestro has been rebranded as Hyper Orchestrator (**HypO**) to align with ETSI's open source ecosystem. HypO's codebase is publicly available on the ETSI GitLab. To avoid confusion, references within AC^3 project will maintain the name MAESTRO until the end of the project. However, references to repository and documentation will show links to HypO since this is the open version of MAESTRO.

4.2.7.1 Functional Description

MAESTRO operates as a holistic end-to-end service orchestration platform, assuming the role of the Life-Cycle Management (LCM) engine for Use Case 3 as detailed in Deliverable D5.2. Its main role is to bridge the gap between high-level service requirements and the underlying distributed infrastructure by offering zero-trust, multi-domain orchestration abstractions. The platform relies on a modular, service-based architecture composed of multiple loosely coupled microservices that handle everything from service onboarding to runtime adaptation. The following paragraphs present the key functional capabilities of the deployed MAESTRO SO module

Standardised Northbound Interfaces

At its boundary, MAESTRO exposes a comprehensive suite of REST APIs compliant with TM Forum specifications. This standardized approach allows service providers to seamlessly manage the entire lifecycle of their applications. The primary interfaces include Service Catalogue Management for organising abstract service specifications, Service Ordering for requesting service deployments, and Service Inventory Management for monitoring and manipulating active runtime instances. This standards-based design ensures interoperability and simplifies the integration of external user interfaces and access control gateways.

Service Order Management Engine

The core processing component of MAESTRO is an orchestration engine that governs the lifecycle of every service order. This component leverages Business Process Model and Notation (BPMN) workflows to encode the exact sequence of operations required to transition a service from an initial request to a fully active state. The engine operates asynchronously by listening to message broker topics, bootstrapping workflows upon receiving new

© AC3 2023

order events, and coordinating with peripheral microservices to execute packaging, network configuration, and deployment tasks. Furthermore, it provides runtime capabilities, offering dedicated interfaces to modify an active service's characteristics dynamically. This includes adjusting compute resource requests, changing the number of active replicas for scaling, or updating the service package version.

Application Translation and Manifest Generation

Within the AC^3 deployment pipeline, MAESTRO integrates directly with the Ontology and Semantic-Aware Reasoner (OSR). When a deployment request is initiated, MAESTRO receives a structured Application Descriptor from the OSR. An internal translation module interprets this descriptor to automatically generate a complete set of Kubernetes manifest files. These generated files encompass all necessary infrastructure parameters, including deployment specifications, service exposures, role-based access control bindings, service accounts, and persistent volume claims.

Multi-Cluster Distribution and Placement

To manage applications across a federated cloud-edge environment, MAESTRO interfaces directly with Red Hat's Advanced Cluster Management (ACM) system. The orchestrator employs a sophisticated label-based routing mechanism to match the specific requirements of each microservice with the capabilities of available target clusters. This allows the intelligent distribution of workloads across various environments, such as standard Kubernetes clusters, OpenShift environments, or nodes with specialised hardware. Following the identification of the target cluster, MAESTRO transforms the standard Kubernetes manifests into specialised CustomManifestWork resources. These resources encapsulate the deployment instructions required by the ACM Hub, ensuring a centralised and accurate dispatch of workloads to the designated edge sites.

Package and Registry Management: To handle the diverse ways applications are constructed, MAESTRO includes a dedicated Package Manager component. While it supports native Kubernetes manifests and docker-compose setups, it primarily utilises Helm as its core abstraction layer for managing complex service deployments. To securely pull container images and deployment charts from external sources, the orchestrator features a Registry API. This API integrates with secure vault services to manage sensitive credentials, ensuring that interactions with private repositories remain secure and trusted.

Continuous Monitoring and Closed-Loop Adaptation:

MAESTRO supports automated application adaptation by orchestrating closed feedback loops based on real-time telemetry. It employs a Service Monitor daemon that evaluates the health and stability of running instances over defined time windows. For proactive adaptation, MAESTRO translates service level agreements specified during the initial request into actionable Kubernetes resources. The orchestrator dynamically creates ConfigMaps for Prometheus adapters, instructing them on how to query and aggregate specific metrics. Simultaneously, MAESTRO deploys Horizontal Pod Autoscaler (HPA) resources to the target clusters. These HPA controllers query the local cluster metrics APIs to retrieve the exposed custom metrics. This mechanism allows the underlying infrastructure to execute fine-grained scaling decisions autonomously based on the thresholds defined by the AI-based LCM algorithms, ensuring continuous operational reliability.

4.2.7.2 Repository Information

- **Repository name:** ETSI Module Development Group for Hyper Orchestrator (HypO)
- **Repository URL:** https://labs.etsi.org/rep/osl/hypo
- **Release model:** OSS

© AC3 2023

- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** /code https://labs.etsi.org/rep/osl/hypo/code
- **Current stable version:** 2025.12.4 (first ETSI release by May 2026)

4.2.7.3 Integration in AC³

Within the AC³ project, MAESTRO is exclusively integrated into UC3 to manage the complex, data-heavy astronomy processing pipelines. It handles the entire lifecycle of the UC3 application, which includes components such as the data orchestrator, RabbitMQ message broker, and the specialised scientific processors (Starlight, pPXF, and STECKMAP). Following its core architectural principle, MAESTRO operates from a centralized Cloud Management Domain observing all the registered UC domains (i.e., it does not run on the edge nodes directly but interacts with all registered edge domains). Furthermore, it interacts with a multi-cluster control plane to dispatch workloads across federated application clusters, spanning OpenShift environments hosted on external cloud infrastructures.

Based on these applied integration and architectural principles, the integration of MAESTRO with other AC³ software modules relies on clearly defined boundaries and standard HTTP/REST protocols. The interaction model is structured as follows:

- **Interaction with the Ontology and Semantic-Aware Reasoner (OSR): MAESTRO** receives deployment requests directly from the OSR. The OSR translates user intents into an AC³ Application Descriptor. MAESTRO's internal translation layer ingests this descriptor and programmatically generates the required Kubernetes manifest files, such as Deployments, Services, Persistent Volume Claims, and RoleBindings, ensuring all dependencies and resource requests are accurately mapped.
- **Interaction with Advanced Cluster Management (ACM): To** execute the deployment, MAESTRO interfaces with Red Hat's ACM Hub, which acts as the compute Local Management System (LMS). MAESTRO applies a label-based routing scheme to evaluate cluster capabilities and packages the generated Kubernetes manifests into CustomManifestWork resources. These resources are then transmitted to the ACM Hub, which handles the final scheduling and placement onto the target OpenShift clusters.
- **Interaction with the Telemetry Framework and AI Models:** MAESTRO is tightly integrated with the AC³ monitoring and AI-based lifecycle mechanisms. It creates Prometheus Adapter ConfigMaps and Horizontal Pod Autoscaler (HPA) resources on the target clusters. This configuration instructs the local Prometheus instances to expose specific custom metrics, such as the RabbitMQ queue length, to the Kubernetes Custom Metrics API. The HPA controllers subsequently read these values to make autonomous, fine-grained horizontal scaling decisions based on thresholds predicted by the external AI algorithms.
- **Interaction with Network Programmability Operators:** To guarantee connectivity between microservices scheduled on physically separated clusters, MAESTRO relies on underlying network operators. While the application manifests are distributed by MAESTRO, the platform ensures that the required exposure parameters are present so that the AC³ Network Operator can establish secure Layer 7 tunnels between the remote application components.

4.2.7.4 Documentation and Maintenance

Documentation is provided at: https://osl.etsi.org/hypo/documentation/

This includes: a) High-level information on orchestrator logic, mission, and addressed ecosystem, b) Architecture and design details, c) APIs, d) a detailed development guide with all services based on Apache Quarkus, e) instructions on how to use HypO (Maestro), and f) Installation instructions

All core updates are managed by UBITECH but are open to involved community members.

4.2.8 Migration Algorithm UC1

4.2.8.1 Functional Description

This module is responsible for the decision-making process regarding the migration of applications between the cloud and the edge. It retrieves information about resource availability and determines whether a service should be migrated, as well as the most suitable target domain (cloud or edge).

The module is integrated with LiSO, which acts as the LCM component responsible for executing the migration once a decision has been made.

4.2.8.2 Repository Information

- **Repository name:** Migration Algorithm UC1
- **URL:** https://github.com/EU-AC3/migration-algorithm-uc1
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Apache License Version 2.0
- **Main branch:** main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.8.3 Integration in AC³

This module is integrated with the LCM component, which receives the migration decision and performs the requested application migration.

Dependencies:

- LCM deployed and operational
- OSR available

4.2.8.4 Documentation and Maintenance

Documentation

The repository contains a README file describing the module's functionality, the process for building the Docker image, and deployment instructions. The component must be deployed close to the LCM to provide migration decisions.

Issue Tracking

Managed through the repository issue tracker.

4.2.9 Migration Algorithm UC2

4.2.9.1 Functional Description

The UC2 Migrator Algorithm is a software component responsible for real-time resource management and orchestration decisions. The component operates through a four-module architecture that performs continuous monitoring, prediction, and migration coordination:

Core functionality

The algorithm collects real-time resource usage data from the monitoring component, feeds this data to an intelligent time-series prediction model, and determines whether microservices should be migrated from a specific host based on the predictions.

Adaptive learning capability

The system continuously enhances its prediction model with new operational data to adapt and optimise performance for the specific machine environment in which it is deployed.

Modular architecture: The algorithm comprises four specialised components—the Fine Tuner, Memory Consumer, Migrator, and Predictor Model—each of which will be detailed in the following subsections. The repository is organized into four main components, each serving a specific purpose in the migration pipeline.

- **Fine_tuner:** Automatic predictor model fine-tuning module
- **Memory_consumer:** Memory consumption tool for triggering migrations
- **Migrator:** Core migration logic and coordination component
- **Predictor_model:** Initial model training and development

Fine Tuner Component

This module is responsible for collecting real-time data from the exposure framework, persisting it, and periodically fine-tuning the prediction model. If the performance metrics of the newly trained model exceed those of the current production model, the fine tuner automatically promotes the improved version to production. This component is essential as each target machine requires the model to be trained on its specific data patterns to achieve optimal accuracy.

The module consists of:

- **Dockerfile:** Container definition for the fine-tuning service
- **archive.py**: Data archiving and versioning functionality
- **config.py:** Configuration parameters and environment settings
- **db_controller.py:** Database operations and object management
- **finetuner.py:** Core fine-tuning logic and model evaluation
- **main.py:** Application entry point and orchestration
- **monitoring.py:** Real-time data collection from exposure framework
- **processing.py:** Data preprocessing and transformation utilities
- **requirements.txt:** Python package dependencies

Memory Consumer Component

This is an experimental utility designed for testing and development environments only. It is not intended for production deployment. The component artificially loads system memory with sufficient data volume to trigger migration events, enabling controlled testing of the migration pipeline under load conditions.

Migrator Component

This component handles the core migration decision logic by processing model outputs and communicating with the orchestrator (LCM) to execute microservice migrations. The component is currently validated in laboratory

© AC3 2023

environments using local test data. It operates at the VIM level and does not integrate with LiSO in its current implementation. The module consists of:

- **collector.py:** Data collection functions interfacing with the exposure framework
- **config.py:** Component configuration and runtime parameters
- **db_controller.py:** Database connectivity and object persistence
- **main.py:** Application entry point and workflow coordination
- **migration_controller.py:** LCM communication and migration request handling
- **utils.py:** Supporting utility functions
- **workloads.json:** Workload definitions and migration target specifications

Predictor Model Component

This module contains the codebase for training and maintaining the initial prediction model that powers the migration algorithm. The implementation currently utilises a multi-layer Long Short-Term Memory (LSTM) architecture for time-series forecasting. The module consists of:

- **api.py:** Model inference API endpoints
- **compare.py:** Model comparison and evaluation utilities
- **config.py:** Model hyperparameters and training configuration
- **model.py:** LSTM model architecture definition
- **ploter.py:** Visualisation tools for model performance and predictions
- **test_api.py:** API testing and validation suite

4.2.9.2 Repository Information

- **Repository name:** migration-algorithm-uc2
- **Repository URL:** https://github.com/EU-AC3/migration-algorithm-uc2
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Custom License
- **Main branch (AC3):** main
- **Core source path:** application/ (within the repo root)

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.9.3 Integration in AC³

Subcomponents interactions

The four subcomponents operate within a distributed architecture, communicating both internally among themselves and externally with surrounding infrastructure services through REST API interfaces. This API-driven design ensures loose coupling, scalability, and interoperability across the migration pipeline.

External System Integration

At the external interface level, the Migrator component integrates with two critical infrastructure services. It communicates with the Orchestrator (LCM) through a dedicated REST API to request and coordinate microservice migrations based on model predictions and decision logic. Additionally, it interfaces continuously with the

© AC³ 2023

Exposure Framework to collect real-time resource usage metrics and system telemetry that feed into the prediction pipeline.

The Fine Tuner component similarly maintains an external connection to the Exposure Framework, consuming the same real-time data stream that feeds the production prediction system. This parallel data collection enables the Fine Tuner to accumulate independent training datasets without interfering with production operations, allowing for offline model evaluation and validation before potential deployment.

4.2.9.4 Documentation and Maintenance

The repository is maintained by EUR. Issues and updates are managed through the repository issue tracker and consortium-internal coordination channels.

4.2.10 Migration Operator

4.2.10.1 Functional Description

The Migrator is a Kubernetes operator designed to checkpoint stateful running containers and push them to remote container registries, enabling the preservation and restoration of container runtime state. This capability is essential for stateful workload migration scenarios where container memory state, open file descriptors, and in-memory data structures must be preserved during migration between nodes. The operator enables the creation of container images that capture the exact runtime state of a running container, which can subsequently be used to restore the container to its exact state on a different node or cluster.

The Migrator component is structured as a standard Kubebuilder-based operator project with the following organization:

- **api/:** Custom Resource Definition (CRD) API types and validation logic
- **controllers/:** Reconcile loops and controller business logic
- **models/:** Data models and internal representations
- **main.go:** Operator entry point and manager initialization
- **config/:** Kubernetes manifests for deployment, RBAC, and CRD samples
- **Dockerfile:** Multi-stage build definition for containerization
- **Makefile:** Build automation and deployment targets
- **go.mod and go.sum:** Go module dependencies and version locking

The project follows the Kubernetes Operator pattern, implementing controllers that provide reconcile functions responsible for synchronizing resources until the desired state is reached on the cluster. The API definitions can be modified and regenerated using make manifests, which updates the CRD and CR manifests.

4.2.10.2 Repository Information

- **Repository name:** Migrator_operator
- **Repository URL:** https://github.com/EU-AC3/migrator_operator
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License 2.0
- **Main branch:** main
- **Core source path:** application/ (within the repo root)

4.2.10.3 Integration in AC³

The Migration Operator is reported in D5.4 as an AC3 software innovation item associated with the UC2 migration work. It was demonstrated as a checkpoint/restore capability but is not part of the final UC2 end-to-end deployed software path. The repository remains available as a reusable component for future stateful migration scenarios.

4.2.10.4 Documentation and Maintenance

The repository is maintained by EUR. Issues and updates are managed through the repository issue tracker and consortium-internal coordination channels.

4.2.11 Horizontal Autoscaling Components

4.2.11.1 Functional Description

The UC3 Horizontal Autoscaling system provides predictive, AI-driven scaling of astronomy data processing pods based on anticipated workload rather than reactive CPU/memory thresholds. The system consists of three cooperating components: a predictor client, a model API, and a Prometheus Adapter configuration that bridges the prediction output to the Kubernetes Horizontal Pod Autoscaler (HPA).

Predictor Client

The predictor client is a Python application that runs as a sidecar deployment alongside the UC3 processing workloads. On a configurable interval (default 10 seconds), it queries the OpenShift Thanos Querier for three operational metrics: the number of running processor pods (num_processors), the current job size in megabytes (job_size), and the number of jobs waiting in the RabbitMQ queue ahead of the current job (queue_ahead_length). These metrics are forwarded to the Model API as a JSON payload. The returned prediction is added to a sliding window average (configurable window size, default 5 samples over 5 minutes) to smooth out transient spikes. The averaged prediction is exported as a Prometheus metric (predicted_job_time_avg) via a metrics endpoint on port 9090, which is scraped by the OpenShift monitoring stack via a ServiceMonitor.

Model API

The Model API is a standalone Flask application that serves a trained Extreme Gradient Boosting (XGBoost) machine learning model. It accepts POST requests at /predict with processor count, job size, and queue length as input features, and returns a predicted job execution time in seconds. The model is trained on historical batch processing data collected from the UC3 testbed. A /health endpoint reports model loading status. The API is deployed as a separate pod (model-api) accessible within the cluster on port 5000.

Prometheus Adapter

A custom Prometheus Adapter deployment bridges the gap between the predictor client's exported metric and the Kubernetes External Metrics API. The adapter is deployed in the openshift-user-workload-monitoring namespace with a ConfigMap that defines an external rule: it queries predicted_job_time_avg from Prometheus and exposes it as predicted_job_time_seconds via the external.metrics.k8s.io API. The adapter runs with TLS certificates issued by the OpenShift service CA and authenticates to Thanos using the cluster monitoring TLS chain.

An APIService resource (v1beta1.external.metrics.k8s.io) registers the adapter with the Kubernetes API server, making the external metric available to HPA controllers cluster-wide.

Horizontal Pod Autoscaler

© AC3 2023

The HPA resource (ucm-processor-hpa) targets the ucm-processor-deployment and scales between 1 and 10 replicas. It uses the predicted_job_time_seconds external metric with an AverageValue target of 100 seconds. When the predicted average job time exceeds this threshold, the HPA increases the number of processor pods to distribute the workload. When predictions fall below the threshold, the HPA scales down to conserve resources.

4.2.11.2 Repository Information

The horizontal autoscaling components are distributed across two locations:

- **HScaler (predictor client and model API):** Located within the EU-AC3 organizations use-case-3 repository under astroapp/application/hscaler/.
- **URL:** https://github.com/EU-AC3/use-case-3/tree/main/astroapp/application/hscaler
- **HScaler model:** Located within the EU-AC3 organizations autoscaling-ac3 repository
- **URL:** https://github.com/EU-AC3/autoscaling-ac3
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Apache License Version 2.0
- **Kubernetes manifests (HPA, Prometheus Adapter, ServiceMonitor): Located** in the ac3_astroapp repository under astroapp/application/deployments/ (hpa.yaml, hscaler.yaml, hscaler-model.yaml, prom-adapter.yaml).

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.11.3 Integration in AC³

Use Case Integration

The horizontal autoscaling system is used exclusively in UC3. It integrates into the processing pipeline as follows:

1. The UC3 application backend (the manipulator component) exports operational metrics (job queue length, job size, batch processing time) to Prometheus via Redis-backed aggregation and a metrics endpoint.
2. The predictor client reads these metrics from the OpenShift Thanos Querier and forwards them to the XGBoost model API.
3. The model API returns a predicted job time, which the predictor client averages and re-exports to Prometheus.
4. The Prometheus Adapter exposes this prediction to the Kubernetes External Metrics API.
5. The HPA reads the external metric and scales the ucm-processor-deployment accordingly.

This creates a closed-loop, predictive autoscaling system where scaling decisions are driven by anticipated workload rather than lagging resource utilisation indicators.

© AC³ 2023

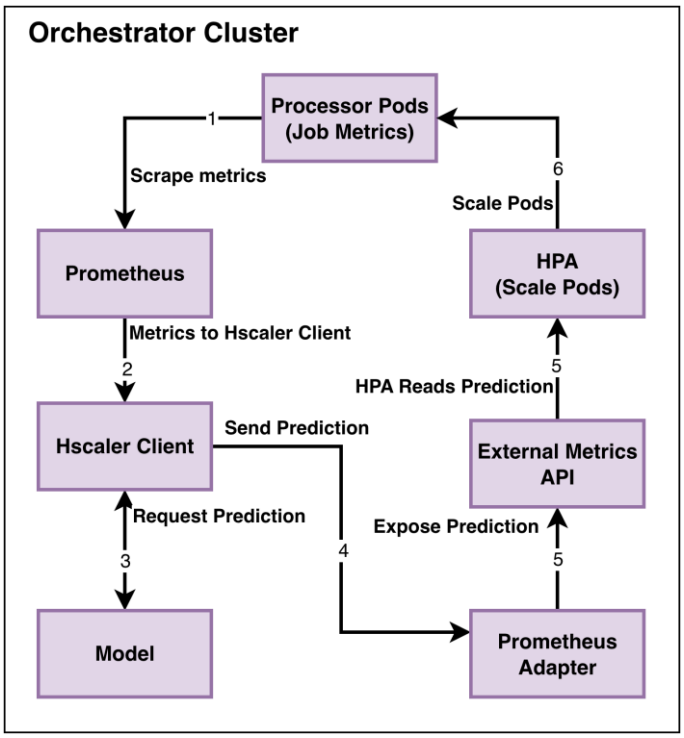

Figure 1: UC3 Autoscaling Flow

Deployment Domain

All autoscaling components are deployed in the cloud domain on the UC3 application clusters. The Prometheus Adapter is deployed in the openshift-user-workload-monitoring namespace (a platform namespace), while the predictor client, model API, and HPA operate in the uc3-applications namespace.

Inter-Component Interaction

- **Input:** Prometheus metrics from the UC3 application backend (astroapp_job_size_mb, astroapp_job_queue_ahead_length, kube_pod_status_phase)
- **Processing:** Predictor client to Model API (HTTP POST on port 5000)
- **Output:** predicted_job_time_avg metric exported to Prometheus, consumed via External Metrics API by HPA
- **Effect:** HPA scales ucm-processor-deployment between 1 and 10 replicas

4.2.11.4 Documentation and Maintenance

Documentation is provided at:

astroapp/application/hscaler/README.md: Comprehensive documentation covering component architecture, local installation, Kubernetes deployment, configuration, API contracts, and Prometheus scrape configuration.

Deployment manifests in astroapp/application/deployments/ serve as operational documentation for the Kubernetes resources.

The horizontal autoscaling components are developed and maintained jointly by IBM (ML model development and training, decision engine design) and Red Hat Waterford Emerging Technologies (predictor client, Prometheus Adapter integration, HPA configuration, Kubernetes deployment).

© AC³ 2023

4.2.12 Scheduler (P2CODE) UC3

The Multi-Cluster Scheduler operates as a central workload placement controller for Use Case 3. It extends the capabilities of Red Hat Advanced Cluster Management to intelligently distribute complex application deployments across federated edge and cloud environments.

4.2.12.1 Functional Description

The primary function of the scheduler is to evaluate incoming workload requests and determine the most appropriate target cluster for deployment within a federated infrastructure. It acts as an advanced decision-making layer sitting on top of the multi-cluster control plane. The key implemented are listed below.

Claim-Based Scheduling Logic

The scheduler employs a sophisticated annotation mechanism to route workloads. Cluster administrators maintain specific claims on each managed cluster to advertise available features. These features include physical location, the specific Kubernetes distribution installed, or the presence of specialised hardware like GPUs. The scheduler evaluates application requests against these claims using two distinct mechanisms: filters and ranks. Filters act as strict requirements that must be met for a deployment to proceed. Ranks act as optimization preferences to maximise the availability of requested hardware resources such as CPU or memory. The scheduling logic strictly prioritises filter annotations over rank annotations to ensure compliance with absolute infrastructure constraints.

Resource Bundling

To ensure deployment consistency, the scheduler dynamically bundles a core workload with all its ancillary resources. This process automatically groups the main application deployment with its associated service definitions, configuration maps, secrets, service accounts, and persistent volume claims. This bundling guarantees that all necessary components arrive at the target cluster together, preventing orphaned manifests and partial deployment failures.

Automated Cross-Cluster Networking

The scheduler possesses the intelligence to establish network connections automatically between application components spread across multiple isolated clusters. It analyses the environment variables of all workloads to build a comprehensive map of component dependencies. If the scheduler places dependent components on different physical clusters, it automatically generates a multi-cluster network request. This request triggers the underlying network operator to create secure virtual application networks, allowing distributed services to communicate as if they resided on the same local network.

4.2.12.2 Repository Information

- **Repository name:** p2code_scheduler_operator
- **URL:** https://github.com/rh-waterford-et/p2code_scheduler_operator
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** main

4.2.12.3 Integration in AC³

© AC³ 2023

The scheduler is integrated into Use Case 3 to support the distribution of demanding astronomical data processing workloads across federated computing environments.

The component operates strictly within the Cloud Management Domain. It runs alongside the MAESTRO orchestrator on the central hub cluster to oversee the downstream application clusters.

The scheduler interacts with surrounding AC^3 components through standard Kubernetes APIs.

- **Interaction with MAESTRO:** The scheduler acts as the southbound execution target for MAESTRO. Once MAESTRO translates the high-level AC^3 Application Descriptor into concrete Kubernetes manifests, it enriches the metadata fields with specific location and resource labels. MAESTRO then submits these enriched requests to the scheduler. The scheduler processes the labels against available cluster claims to decide the final placement.
- **Interaction with the AC3 Network Operator:** When the scheduler distributes related microservices across different clusters, it interacts directly with the AC^3 Network Operator. It submits network link requests to the operator API. The operator subsequently establishes the required Layer 7 tunnels to ensure seamless data flow between the separated application components.

4.2.12.4 Documentation and Maintenance

Documentation is provided at:

- **README.md:** Comprehensive documentation covering component architecture, local installation, Kubernetes deployment, configuration, usage examples, and License details.

The P2Code Scheduler component is developed and maintained jointly by UBITECH and Red Hat Waterford Emerging Technologies.

4.2.13 AI-Based Resource Predictor - Transformer-based predictive capabilities for resource management

The AI-Based Resource Management component operates as an intelligence layer for predicting latency and allowing for autoscaling resources across the CECC. It leverages Transformer-based approaches to predict latency, enabling proactive autoscaling. The component provides predictive insights into job execution time based on workload characteristics, allowing orchestration systems to dynamically allocate compute resources, minimise queue delays, and optimise system throughput.

4.2.13.1 Functional Description

The primary function of the AI-Based Resource Management component is to predict workload execution duration and support data-driven autoscaling decisions. It consists of a prediction service that integrates with scheduling and orchestration layers to improve resource utilisation and reduce latency.

The Prediction Service exposes data that returns predicted values for job execution duration (total_duration_seconds) based on specific features. As an example, the model uses the following input features:

- processor_count – number of allocated compute units
- job_size_mb – size of the workload/data to be processed
- queue_ahead_length – number of jobs waiting in the queue

The service is powered by a Transformer-based model, which captures dependencies and nonlinear relationships between the workload, number of allocated compute units, number of jobs behaviour. The model is trained on

© AC3 2023

historical job execution data and is capable of learning complex interactions between system load, job size, and compute allocation.

The predictions that are generated can allow to:

- Trigger autoscaling actions (scale up/down compute resources)
- Optimise scheduling decisions and queue management
- Improve SLA adherence and reduce execution delays

The output is returned as a .json enabling seamless integration with orchestration frameworks. The component supports autoscaling by translating predicted execution durations into actionable scaling

information. For example:

- Long predicted durations under high queue load may trigger horizontal scaling
- Short predicted durations may prevent unnecessary scaling actions
- Queue-aware predictions enable proactive scaling before bottlenecks occur

This enables a shift from reactive to predictive autoscaling, improving both performance and cost efficiency.

4.2.13.2 Repository Information

- **Repository name:** ai-resource-predictor
- **URL:** https://github.com/EU-AC3/ai-resource-predictor
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Apache License Version 2.0
- **Main branch:** Main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.13.3 Integration in AC³

The predictive capability of the AI-Based Resource Predictor component integrates with the use case 3 to support proactive autoscaling. The integration with the orchestration components provides execution time predictions to the schedulers and enables resource provisioning decisions.

4.2.13.4 Documentation and Maintenance

A README.md file is provided at https://github.com/EU-AC3/ai-resource-predictor/blob/main/README.md. The file contains the key features, input features, output, project structure, installation, model training, and inference. The file also provides the docker architecture, model architecture, integration and configuration mechanisms.

4.2.14 AI-Based Application Profile

The AI-Based Application Profile component operates as a central intelligence layer for predictive resource planning and application profile management across Use Cases 1, 2, and 3. It delivers ML-driven forecasts of application resource consumption and provides a persistent storage service for application profiles and policies, enabling informed decision-making for workload placement and capacity planning in federated edge and cloud environments.

4.2.14.1 Functional Description

The primary function of the AI-Based Application Profile component is to predict future resource demand for applications and to manage structured application profiles that describe microservice architectures,

dependencies, and associated policies. The component consists of two complementary services that together support the AC³ Application Profile Manager ecosystem.

ML-Based Resource Prediction

The Prediction Service exposes a REST API that returns forecasted values for twelve application metrics at four-time horizons (1 hour, 6 hours, 12 hours, and 24 hours). The metrics cover CPU usage, memory usage, disk throughput (read and write), network uplink and downlink, data transferred and received, request rate, latency, throughput, and error rates. The service uses a pre-trained scikit-learn model with scaled inputs and outputs, trained on historical application behaviour patterns. The model leverages the current time-of-day as a feature to capture diurnal patterns typical of operational workloads. The predictions are returned in a structured format compatible with dashboard visualisation components, enabling operators to anticipate resource needs and plan scaling or migration actions in advance.

Application Profile CRUD Operations

The Application Profile Service provides full create, read, update, and delete capabilities for application profiles. Each profile describes an application's microservice topology, dependencies between services, data sources, network traffic characteristics, and deployment context. The service persists profiles in MongoDB and exposes a RESTful API for integration with the AC³ GUI and other orchestration components. It supports file upload for profile import and export.

4.2.14.2 Repository Information

- **Repository name:** ai-application-profile-core-uc1-uc2-uc3
- **URL (or access instructions):** https://github.com/EU-AC3/ai-application-profile-core-uc1-uc2-uc3
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Fingletek Public License, Version 1.0 (provided in the repository; not a standard open source License)
- **Main branch:** main

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.14.3 Integration in AC³:

The AI-Based Application Profile component is integrated into Use Cases 1, 2, and 3 to support predictive capacity planning, application profiling, and policy-driven deployment decisions.

- **Interaction with the AC³ GUI:** The Application Profile Service is used by the GUI for creating, viewing, editing, and deleting application profiles and policies.
- **Interaction with Orchestration Components:** Application profiles managed by the Application Profile Service can be consumed by other AC³ components (such as MAESTRO or LiSO via OSR) to inform placement decisions and validate deployment constraints.

© AC³ 2023

The component operates within the Application Profile and Dashboard domains of the AC³ architecture. It provides northbound interfaces (REST APIs) for the GUI and other consumers and does not require southbound agents on downstream clusters.

4.2.14.4 Documentation and Maintenance

Documentation for the AI-Based Application Profile component is provided within the repository and includes service-level descriptions, API usage, and deployment guidelines. A root README.md outlines the overall component structure, while each service (Prediction Service and Application Profile Service) includes its own documentation covering configuration, environment variables, and execution procedures. Additional inline code comments are provided where necessary to clarify implementation details.

The component is maintained by Fingletek Oy, which is responsible for ongoing development, bug fixing, and updates to both the prediction models and the application profile management logic.

The component follows the decentralized maintenance model of AC3, allowing independent evolution while ensuring compatibility with other system components through stable API contracts.

REMARK: The following components outlines the core data-management modules of the AC3 framework, detailing their roles in handling, processing, and transforming data across different use cases. The modules include Data Providers, Data Mappers, and Data Manipulators, which are responsible for acquiring, transforming, and applying data to the respective applications. These data management components serve distinct purposes in Use Case 1 (UC1) and Use Case 3 (UC3), with UC1 focusing on environmental and IoT data processing in a smart building context, and UC3 handling large-scale astronomy data for in-depth scientific analysis.

4.2.15 UC1 Data Provider Connector Components

4.2.15.1 Functional Description

The UC1 Data Provider Connector is an Eclipse Dataspace Connector (EDC) based component that exposes IoT data from the UC1 data-source domain as policy-governed data assets accessible to consumers via the IDS Dataspace Protocol (DSP). It implements the provider side of the AC3 data management chain for UC1, enabling federated, contract-negotiated data sharing between the UC1 data-source domain and the edge processing domain.

The connector is provided in two variants to handle both static and live IoT data:

- Cold data provider (sparkworks/ac3-connector-http-http-provider). Exposes static files stored in a local directory as EDC data assets of type LocalFiles. The provider registers the asset directory, defines an access policy, and creates a contract definition, making the files available to consumers via HTTP data transfer.
- Hot / IoT streaming data provider (sparkworks/ac3-connector-iot-http-http-provider). Extends the cold provider with a MQTT bridge (bridge.py) that subscribes to a configurable MQTT topic on the SparkWorks IoT broker and continuously writes incoming sensor messages to the local data directory. Each message is persisted as a per-device file, creating a rolling snapshot of the live IoT stream. The EDC connector then exposes these files as an HTTP data asset (uc1-stream), making the real-time sensor data available for transfer to the consumer connector in the edge domain. The bridge runs concurrently with the EDC connector process within the same container.

The EDC framework handles the complete data sharing lifecycle on the provider side:

© AC³ 2023

- **Asset Management**: The provider registers the IoT data directory as a named data asset (uc1-stream) with a LocalFiles data address pointing to the container-internal data directory (/usr/src/app/data).
- **Policy Definition**: An access policy (no-constraint-policy) is defined using ODRL, specifying the conditions under which the asset can be shared.
- **Contract Definition**: A contract definition links the access policy to the available assets, making the asset discoverable via the IDS catalogue endpoint.
- **Data Serving**: When a consumer initiates a transfer after contract negotiation, the EDC data plane serves the asset content via HTTP PUSH to the consumer's nominated destination URL.

Provider setup and asset registration are automated via a Python helper (provider_helper.py) that wraps the EDC Management API calls for asset creation, policy definition, and contract definition. A reference script (0-prepare_provider.py) demonstrates full provider initialisation for the UC1 deployment, targeting the provider endpoint at http://ds.uc1.ac3.sparkworks.net.

4.2.15.2 Repository Information

- **Repository name:** data-management
- **URL:** https://github.com/EU-AC3/data-management
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** master

4.2.15.3 Integration in AC3

Use Case Integration

The UC1 Data Provider Connector is integrated in UC1 only.

Deployment Domain

The provider connector is deployed in the UC1 data source. It runs as part of the UC1 data source software deployed by SPA and IQU, receiving sensor data from the sensors through an MQTT.

Interaction with other AC3 software modules

- **UC1 Data Consumer Connector (4.2.17):** The consumer connects to the provider's DSP endpoint at http://ds.uc1.ac3.sparkworks.net:18182/protocol to negotiate a contract and initiate transfer of the uc1-stream asset.
- **Catalogues (4.2.3):** The provider connector is registered in the AC3 Data and Service Catalogue as a dataset's distribution endpoint, allowing it to be referenced and included in application descriptors generated by the OSR.

4.2.15.4 Documentation and Maintenance

Documentation location

- README.md in the software's repository: Overview, deployment instructions, and usage guidance.

Maintainer

The UC1 Data Provider Connector is developed and maintained by Spark Works (SPW) as part of the AC3 consortium.

Issue tracking mechanism

GitHub repository issue tracker on the use-case-1 repository, supplemented by consortium-internal coordination channels.

4.2.16 EDC Connector for IONOS S3

4.2.16.1 Functional Description

The UC3 Data Provider Connector is based on the Eclipse Dataspace Connector (EDC) framework, extended with IONOS S3 integration modules. It implements federated, policy-compliant data sharing between the UCM (Universidad Complutense de Madrid) astronomy data provider and the RHT (Red Hat) consumer environment, following the International Data Spaces (IDS) protocol.

The connector system consists of two EDC instances (provider and consumer) and an automated transfer trigger. The provider connector, operated by UCM, exposes astronomy datasets stored in IONOS S3 buckets as data assets with associated access policies and contract definitions. The consumer connector, operated by RHT, discovers available assets by querying the provider's catalogue, negotiates contracts to establish data sharing agreements, and initiates data transfers between S3 buckets.

The EDC framework handles the complete data sharing lifecycle:

- **Asset Management:** The provider registers S3 objects as data assets with metadata describing the data format, loc and access conditions.
- **Policy Definition:** Access policies define the conditions under which data can be shared (e.g., permitted actions, usage constraints).
- **Contract Negotiation:** The consumer initiates a negotiation with the provider. If the consumer's request satisfies the provider's policy, a contract agreement is established.
- **Data Transfer:** Once a contract agreement is in place, the consumer requests a data transfer. The EDC data plane handles the actual S3-to-S3 transfer, with temporary credentials managed by HashiCorp Vault.

The transfer trigger component (trigger.py) automates this workflow by monitoring the provider and consumer S3 buckets for new files. When a new file appears in the provider bucket, the trigger automatically creates an asset, policy, and contract definition, then initiates the transfer to the consumer bucket. This enables continuous, bidirectional data synchronisation without manual intervention.

The connector uses the IONOS S3 EDC extensions (core-ionos-s3, data-plane-ionos-s3, provision-ionos-s3), which extend the base EDC with S3-specific provisioning, data plane transfer, and credential management capabilities.

4.2.16.2 Repository Information

- **Repository:** data-provider-conector-uc3
- **URL:** https://github.com/EU-AC3/data-provider-conector-uc3
- **Release model:** Internal
- **Visibility:** Private
- **License:** Apache License Version 2.0
- The upstream EDC IONOS S3 extensions originate from:
  - **URL:** https://github.com/Digital-Ecosystems/edc-ionos-s3
- **Connector image:** quay.io/bcapper30/s3-conn-env

© AC3 2023

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

4.2.16.3 Integration in AC³

Use Case Integration

The EDC connectors are used in UC3 to implement the Data Management layer of the AC³ architecture. The provider connector runs in the UCM environment, and the consumer connector runs on the RHT producer cluster. Together they ensure that astronomy datasets are transferred securely and in compliance with predefined policies before being processed by the UC3 application pipeline.

The data flow is: UCM uploads astronomy data to the provider S3 bucket. The transfer trigger detects the new data. The EDC negotiates a contract and initiates the transfer. The data lands in the consumer S3 bucket. The UC3 producer (the manipulator component) picks up the data from the consumer bucket for processing. After processing, the results are detected by the transfer trigger and transferred back to the provider S3 bucket.

Deployment Domain

Both provider and consumer connectors are deployed in the cloud domain. The provider connector runs alongside the data source, and the consumer connector runs on the UC3 producer cluster. The Vault instance is co-located with the consumer connector.

Inter-Component Interaction

- **Input:** Astronomy data files in IONOS S3 buckets; EDC API request templates (asset.json, create-policy.json, create-contract.json, fetch-catalogue.json, contract-negotiation.json, transfer.json).
- **Processing:** Contract negotiation between provider and consumer EDC instances; S3-to-S3 data transfer via the EDC data plane.
- **Output:** Astronomy data files transferred to the consumer S3 bucket, ready for processing by the UC3 application backend.

4.2.16.4 Documentation and Maintenance

Documentation is provided at:

- **README.md:** Overview of the EDC connector architecture, directory structure, deployment instructions, configuration, EDC API usage, and transfer workflow.
- **deployment/helm/README.md:** Helm chart deployment steps including Vault and EDC installation.
- **deployment/README.md:** Deployment options (Terraform, Helm, Kind).
- **jsons/:** EDC API request templates with documentation of each request type.

The UC3 data connector integration is maintained by Red Hat Waterford Emerging Technologies (consumer connector deployment, transfer trigger, Kubernetes integration) and IONOS (upstream EDC IONOS S3 extensions).

4.2.17 Data Consumer Connector UC1

4.2.17.1 Functional Description

The UC1 Data Consumer Connector is the consumer-side Eclipse Dataspace Connector (EDC) instance for UC1. It operates symmetrically with the UC1 Data Provider Connector (4.2.8.1) to form the data access chain between the data-source domain and the edge processing domain. The consumer connects to the provider's IDS

Dataspace Protocol (DSP) endpoint, queries the available asset catalogue, performs contract negotiation, and initiates an HTTP PUSH data transfer to receive the IoT sensor stream.

The component handles the full consumer-side EDC lifecycle:

- **Catalogue Request:** The consumer queries the provider's protocol endpoint to discover available assets and their associated contract offers.
- **Contract Negotiation:** The consumer submits a ContractRequest referencing the discovered policy offer for the target asset (uc1-stream). The EDC framework exchanges DSP messages with the provider until the negotiation reaches FINALIZED state and a contractAgreementId is established.
- **Data Transfer Initiation:** Once a contract agreement is in place, the consumer submits a TransferRequest specifying the destination URL where the provider's data plane should push the asset content. In the UC1 deployment, the destination is the edge-side data ingestion endpoint (the AMQP broker or the HTTP-AMQP bridge).
- **Transfer Monitoring:** The consumer tracks transfer process state through the EDC Management API until the transfer completes.

A Python helper (consumer_helper.py) wraps all Management API calls, providing methods for request_asset, negotiate_contract, get_negotiation_status, and start_transfer. A reference automation script (2-start-transfer.py) demonstrates the full consumer-side workflow from catalogue request to transfer start, targeting the provider at http://ds.uc1.ac3.sparkworks.net:18182/protocol.

Additionally, the http-amqp-logger sidecar service (sparkworks/ac3-amqp-http-request-logger) acts as the HTTP transfer destination. It receives the HTTP-PUSHed data from the EDC data plane and forwards it to the RabbitMQ AMQP broker (exchange data), bridging the EDC HTTP transfer into the internal AMQP message bus.

4.2.17.2 Repository Information

- **Repository name:** `data-management`
- **URL:** https://github.com/EU-AC3/data-management
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** main

4.2.17.3 Integration in AC3

Use Case Integration

The UC1 Data Consumer Connector is integrated in UC1 only.

Deployment Domain

The consumer connector is deployed in the Edge domain (Edge-LMS-UC1). It runs as part of the edge processing stack deployed by LiSO, receiving sensor data from the data-source domain through the EDC data transfer channel.

Interaction with other AC3 software modules

- **UC1 Data Provider Connector (4.2.15):** The consumer connects to the provider's DSP endpoint at http://ds.uc1.ac3.sparkworks.net:18182/protocol to negotiate a contract and initiate transfer of the uc1-stream asset.

- **UC1 Message Broker (4.2.21):** The HTTP-AMQP bridge forwards incoming HTTP-PUSHed sensor data to the data exchange on the RabbitMQ broker. This injects the IoT data stream into the internal AMQP-based processing pipeline.
- **UC1 Data Mappers (4.2.18):** The mapper consumes messages from the broker's data queue and processes the data forwarded by the consumer connector.
- **Catalogues (4.2.3):** The consumer service is registered in the AC3 Data and Service Catalogue as a service entry (ac3: ServiceShape), allowing it to be referenced and included in application descriptors generated by the OSR.
- **LiSO (LCM) (4.2.6):** The consumer connector is deployed as a microservice by LiSO as part of the UC1 application descriptor, alongside the message broker and data mapper.

4.2.17.4 Documentation and Maintenance

Documentation location

- README.md in the data-consumer-conector-uc1 repository: Overview, deployment instructions, and usage guidance.
- edc/http-http/images/Readme.md in the data-management repository: Image build procedures for the consumer and HTTP-AMQP logger.
- edc/http-http/scripts/Readme.md: Documents the consumer-side workflow including catalogue request, contract negotiation, and transfer initiation.
- consumer_helper.py: Python helper with documented methods for each EDC Management API interaction.

Maintainer

The UC1 Data Consumer Connector is developed and maintained by Spark Works (SPW) as part of the AC3 consortium.

Issue tracking mechanism

GitHub repository issue tracker on the data-consumer-conector-uc1 and data-management repositories, supplemented by consortium-internal coordination channels.

4.2.18 UC1 Data Mappers Components

4.2.18.1 Functional Description

The UC1 Data Mapper is a Java Spring Boot microservice that acts as the normalization and transformation layer within the UC1 edge data pipeline. It consumes raw IoT sensor readings delivered to a RabbitMQ input queue, parses and normalises the data, and re-publishes the transformed measurements to an output queue for downstream consumption by the UC1 data manipulation services.

The mapper performs the following processing steps on each incoming message:

1. **JSON parsing:** Incoming messages are parsed as generic JSON objects. The mapper does not require a fixed input schema; it dynamically iterates over all JSON fields, allowing it to handle sensor payloads from any device type.
2. **Timestamp extraction:** The timestamp field is extracted from the JSON payload. If absent, the current system time is used.
3. **Field filtering:** Non-measurement fields (sensorid, timestamp) are skipped. Only numeric fields are processed.
4. **Field name transformation:** A configurable mapping applies name transformations to selected fields

© AC3 2023

(e.g. iaqAccuracy → iaq_accuracy), normalising internal field names to external sensor naming conventions.

5. **Measurement publishing:** Each extracted numeric reading is published to the output RabbitMQ exchange in the format <uri>/<sensorName>,<value>,<timestamp>, where <uri> is derived from the AMQP routing key of the incoming message.
6. **Optional CSV logging:** A configurable file-writing mode appends measurements to a CSV file on disk, useful for debugging and data backup during integration testing.

The mapper exposes two ports: a Spring Boot Actuator health and metrics endpoint (port 5026) and a REST API endpoint (port 8026). Prometheus metrics are published via the Actuator interface, including mapper.input.messages, mapper.output.messages, and mapper.mapped.latency (tagged by device and sensor name), enabling operational monitoring of the mapping throughput and processing latency.

Within AC3, the mapper is the intermediary between the raw IoT data stream (delivered by the Data Consumer Connector via the message broker) and the AI/ML processing services. It ensures that all downstream consumers receive data in a uniform, device-agnostic format.

4.2.18.2 Repository Information

- **Repository name:** data-management
- **URL:** https://github.com/EU-AC3/data-management
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** master

4.2.18.3 Integration in AC³

The UC1 Data Mapper is integrated in **UC1** only.

Deployment domain

The mapper is deployed in the **Edge domain** (Edge-LMS-UC1), co-located with the message broker and data manipulation services to minimize inter-service latency in the processing pipeline.

Interaction with other AC³ software modules

- **UC1 Message Broker (4.2.8.7):** The mapper consumes from the broker's data queue (where the consumer connector delivers raw IoT data) and publishes to the data-mapped exchange, from which the data-mapped-ml1 and data-mapped-ml2 queues fan out the normalised measurements to the respective manipulator instances.
- **UC1 Data Manipulator (4.2.8.5):** The two manipulator services (ac3-ml1 anomaly detection and ac3-ml2 forecasting) consume from data-mapped-ml1 and data-mapped-ml2 respectively, receiving the normalized measurement format produced by the mapper.
- **UC1 Data Consumer Connector (4.2.8.3):** The consumer connector delivers the raw IoT stream to the data exchange on the broker; the mapper is the first processing step after data ingestion.
- **Catalogues (4.2.3):** The mapper is registered as a service entry in the AC3 Data and Service Catalogue, making it discoverable and referenceable in application descriptors.

© AC³ 2023

- **LiSO (LCM):** The mapper is deployed as part of the UC1 application descriptor managed by LiSO.

4.2.18.4 Documentation and Maintenance

Documentation location

- **README.md in the data-mappers-uc1 repository:** Component overview, deployment instructions, and configuration reference.
- **mapper/ directory in the data-management repository:** Maven project source, Dockerfile, build.sh, and Spring Boot application configuration (application.yml, application-docr.yml).
- **Inline source documentation in RabbitService.java:** Javadoc on the message processing logic, field extraction, transformation, and metrics recording.
- **application.yml:** Documents all configurable properties with their default values and descriptions.

Maintainer

The UC1 Data Mapper is developed and maintained by Spark Works (SPW) as part of the AC^3 consortium.

Issue tracking mechanism

GitHub repository issue tracker on the data-management repository, supplemented by consortium-internal coordination channels.

4.2.19 Data Manipulator UC1

4.2.19.1 Functional Description

The UC1 Data Manipulator implements the AI/ML processing layer of the UC1 edge data pipeline. It consists of two independent Python microservices that consume normalised sensor measurements from RabbitMQ, apply pre-trained machine learning models, and store the resulting predictions and classifications in a MySQL database. Both services expose Prometheus metrics and are designed for deployment close to the IoT data source in the edge domain.

Service 1 — Anomaly Detection (ac3-ml1, data_manipulator_uc1)

The first manipulator service performs real-time anomaly detection on incoming sensor readings using pre-trained Isolation Forest models. It determines whether each sensor reading is in a normal or abnormal state and computes a confidence score.

The service loads a set of pre-trained models at startup, one per sensor type and one composite model:

Table 3: Anomaly Detection ML models trained

Model file	Sensor / scope
if_full_best.pkl	Composite model across all sensors
if_temperature_best.pkl	Temperature

if_humidity_best.pkl	Humidity
if_iaq_best.pkl	Indoor Air Quality (IAQ)
if_co2_best.pkl	CO_2
if_lux_best.pkl	Light (lux)
if_motion_best.pkl	Motion
if_bat_best.pkl	Battery level
if_vibration_best.pkl	Vibration

For each incoming measurement the service: parses the routing key to extract device and sensor identifiers; applies the relevant Isolation Forest model; classifies the reading as normal or abnormal based on the model's decision boundary; persists the result (device, sensor, value, timestamp, processing time, anomaly score) to MySQL via db_operations.py; and exports Prometheus metrics ml_processing_time, ml_processing_state, and ml_processing_score labelled by device and sensor type.

Service 2 — Time-Series Forecasting (ac3-ml2, data_manipulator_uc1-2)

The second manipulator service performs time-series forecasting using pre-trained sequence prediction models. It generates forward predictions for each supported sensor type.

Table 4: Forecasting ML models trained

Model file	**Sensor**
sqnc_temperature_model.pkl	Temperature
sqnc_humidity_model.pkl	Humidity
sqnc_iaq_model.pkl	IAQ
sqnc_co2_model.pkl	CO_2
sqnc_motion_model.pkl	Motion
sqnc_lux_model.pkl	Light (lux)
sqnc_bat_model.pkl	Battery level

For each incoming measurement sequence the service applies the corresponding sequence model to produce forecast predictions with associated uncertainty estimates, persists the results to MySQL, and exports Prometheus metrics (ml_predict_processing_time, ml_predict_processing_state, ml_predict_processing_score).

© AC3 2023

Both services share a common database connection module (db_operations.py) that implements robust connection management: automatic connection testing and reconnection, configurable timeouts (connect: 10 s, read/write: 30 s), retry logic on write failures, and UTF-8 charset support.

4.2.19.2 Repository Information

- **Repository name:** use-case-1
- **URL:** https://github.com/EU-AC3/use-case-1
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** main

4.2.19.3 Integration with AC3

The UC1 Data Manipulator is integrated in **UC1** only.

Deployment domain

Both manipulator services are deployed in the **Edge domain** (Edge-LMS-UC1), enabling near-real-time anomaly detection and forecasting close to the IoT data source. This placement reduces communication overhead and supports low-latency decision-making consistent with the UC1 architecture.

Interaction with other AC3 software modules

- **UC1 Message Broker (4.2.8.7):** The manipulator services consume from the data-mapped-ml1 and data-mapped-ml2 queues respectively, which are fan-out bindings of the data-mapped exchange populated by the mapper. Processed results are published to the data-processed exchange.
- **UC1 Data Mapper (4.2.8.4):** The mapper is the upstream producer of normalised measurements consumed by both manipulator services.
- LiSO (LCM): Both manipulator containers are instantiated and lifecycle-managed by LiSO as part of the UC1 application deployment.
- **Catalogues (4.2.3):** Both manipulator services are registered as service entries in the AC3 catalogue with their container image references, resource requirements, exposed ports, and environment variable definitions, enabling automated descriptor generation by the OSR.
- Resource Exposure Framework / Monitoring: Prometheus metrics exported by both services are scraped by the monitoring layer, providing runtime observability for the AI/ML processing pipeline.

4.2.19.4 Documentation and Maintenance

Documentation location

- **README.md in the data-manipulator-uc1 repository:** Component overview, service descriptions, database connection management, configuration reference, and deployment instructions.
- **application/manipulator/ in the use-case-1 repository:** Source files (ac3-ml1.py, ac3-ml2.py), shared modules (db_operations.py, monitoring.py), requirements, and Dockerfiles (ac3-ml1.dockerfile, ac3-ml2.dockerfile).
- **application/models/ in the ac3-use-case-1 repository:** Pre-trained model files (.pkl, .json) for all supported sensor types and both service variants.
- **Inline documentation in ac3-ml1.py and ac3-ml2.py**: Model loading, message processing logic, Prometheus instrumentation, and database persistence patterns.

Maintainer

The UC1 Data Manipulator is developed and maintained by Spark Works (SPW) as part of the AC^3 consortium.

Issue tracking mechanism

GitHub repository issue tracker on the UC1 repository, supplemented by consortium-internal coordination channels.

4.2.20 Data Manipulator UC3

4.2.20.1 Functional Description

The UC3 Manipulator is the core data processing orchestration component for the UC3 astronomy application. It is a Go-based backend that manages the distribution, execution, and result collection of astronomical data processing jobs across a distributed, event-driven architecture. The manipulator is not a single service but a set of cooperating microservice roles (producer, processor, receiver, aggregator, HTTP server) that run as containers within multi-container pods.

The architecture is built around two RabbitMQ message queues that handle bidirectional communication:

- **producer_to_processor_queue:** The producer retrieves astronomy data files from an S3 bucket, batches them according to a configurable job size, and publishes batches as messages to this queue.
- **processor_to_producer_queue:** After processing, results are published back to this queue for the producer-side receiver to collect, aggregate, and upload to the output S3 bucket.

The manipulator supports three astronomical processing applications, each with a different message format tailored to the data characteristics:

- **Starlight (stellar population synthesis):** Uses standard text-based messages with file content embedded directly in the RabbitMQ message body.
- **pPXF (penalized pixel-fitting for stellar kinematics)**: Uses binary messages to preserve .fits file integrity during transport through RabbitMQ.
- **Voronoi binning (spectral data preprocessing):** Uses S3 reference messages where only the S3 metadata is sent via RabbitMQ and the processor downloads files directly from S3, as data cube files are too large for the message broker.

The manipulator exposes a REST API on port 8080 for dataset management and job control:

Table 5: Data Manipulators Endpoints

API Endpoint	Description
GET /api/datasets	List available datasets in S3
GET /api/datasets/files	List files within a dataset
POST /api/datasets/upload	Upload files to a dataset
POST /api/queue/trigger	Trigger processing for a specific dataset
POST /api/admin/restart-rabbitmq	Restart queue infrastructure

© AC^3 2023

A React-based frontend (deployed as a separate microservice) provides a web interface for astronomers to browse datasets, trigger processing runs, and monitor job progress.

The system uses Redis for real-time metrics aggregation. An aggregator container collects per-job metrics (processing time, file sizes, queue depths) and maintains running statistics that are exported to Prometheus via a metrics endpoint on port 9090. These metrics feed into the horizontal autoscaling system (Section 4.2.6.5).

4.2.20.2 Repository Information

- **Repository:** use-case-3(subdirectory astroapp/)
- **URL:** https://github.com/EU-AC3/use-case-3
- **Release model:** Internal
- **Repository visibility:** Private
- **License:** Apache License Version 2.0
- **Main branch:** main
- **Language:** Go (backend), Python (Starlight, pPXF, Voronoi applications), TypeScript/React (frontend)

Although the repository is private, the License reported above defines the legal conditions applicable to authorised users who are granted access to the code.

The repository contains:

Table 6: Data Manipulator Repository Architecture UC3

Module	Description
astroapp/application/	Go backend source, Kubernetes manifests, Makefile, Dockerfiles
astroapp/starlight/	Starlight processing application and Dockerfile
astroapp/ppxf/	pPXF processing application and Dockerfile
astroapp/voronoi/	Voronoi binning application and Dockerfile
gui	Front end application and Dockerfile

4.2.20.3 Integration in AC³

Use Case Integration

The manipulator is the central application component for UC3. It connects to the data management layer (EDC connectors provide data to S3), the LCM layer (Maestro handles deployment via ACM), the networking layer (the AC³ Network Operator exposes RabbitMQ across clusters via Skupper), and the monitoring/autoscaling layer (metrics feed the horizontal autoscaling system).

The manipulator's microservices are organised into producer components (producer, RabbitMQ, data connector, frontend, Redis) and processor components (receiver-processor, Starlight, pPXF, Voronoi). The AC³ Network Operator ensures connectivity between these components when they are distributed across the infrastructure.

Deployment Domain

The manipulator spans both cloud and edge domains in the UC3 testbed. The producer components run on one OpenShift cluster and the processor components run on another, with Skupper providing the inter-cluster

© AC³ 2023

connectivity.

Inter-Component Interaction

- **Upstream:** Receives astronomy data from the EDC S3 consumer bucket. Deployed to the cluster via Maestro and ACM. Network connectivity managed by the AC³ Network Operator.
- **Internal:** Producer communicates with Processor via RabbitMQ (TCP port 5672). Frontend communicates with the HTTP server (port 8080). Aggregator communicates with Redis (port 6379).
- **Downstream:** Exports processing metrics to Prometheus (port 9090) for the horizontal autoscaling system. Writes processed results to S3 output buckets. An OpenShift Route (uc3-backend-api) exposes the API externally with edge TLS termination.

4.2.20.4 Documentation and Maintenance

Documentation is provided at:

- **astroapp/README.md:** Architecture overview, message type descriptions, directory structure, build/deploy instructions, deployment file inventory, API endpoint reference, scientific application descriptions, and configuration reference.
- **astroapp/application/deployments/README.md:** Deployment file descriptions.

The manipulator is developed and maintained by Red Hat Waterford Emerging Technologies.

4.2.21 Message Broker UC1

4.2.21.1 Functional Description

The UC1 Message Broker provides the internal AMQP message bus for the UC1 edge data processing pipeline. It is a pre-configured RabbitMQ instance that decouples the data ingestion layer (Data Consumer Connector) from the data transformation and AI/ML processing layers (Data Mapper and Data Manipulator), enabling asynchronous, durable, and fan-out message routing between all pipeline components.

The broker is delivered as a custom Docker image built on the official rabbitmq:management-alpine base, extended with a pre-loaded configuration that defines all exchanges, queues, users, permissions, and broker settings required for the UC1 application. This eliminates the need for post-deployment manual configuration and ensures that the broker is ready to serve pipeline components immediately on startup.

Message topology

The broker defines the following exchanges and queues:

Table 7: Exchanges used to route messages during processing of UC1

Exchange	Description
data	Receives raw IoT data from the HTTP-AMQP bridge; routes to data queue
data-mapped	Receives normalised measurements from the mapper; fans out to data-mapped, data-mapped-ml1, and data-mapped-ml2 queues

data-processed	Receives processed results from the manipulator services; routes to data-processed queue
commands	Carries operational commands to pipeline components (e.g., mapper control)

Table 8: Queue Bindings used to route messages during processing of UC1

Queue	Bound exchange	Consumer
data	data	Data Mapper (4.2.8.4) — raw IoT data input
data-mapped	data-mapped	General mapped data sink
data-mapped-ml1	data-mapped	Data Manipulator ML1 — anomaly detection (4.2.8.5)
data-mapped-ml2	data-mapped	Data Manipulator ML2 — forecasting (4.2.8.5)
data-processed	data-processed	Processed result sink
commands	commands	Command consumers (mapper, manipulators)

The fan-out binding from data-mapped to both data-mapped-ml1 and data-mapped-ml2 enables parallel processing of each normalized measurement by both the anomaly detection and forecasting services without duplicating upstream work.

User and permission model

The broker defines dedicated users for each pipeline component, each with full configure, write, and read permissions on the default vhost:

Table 9: Per-service account creation for data exchange in UC1

User	Role	Pipeline Function
uc1	Administrator	Operational management
connector	Management	Data Consumer Connector — publishes to data exchange
mapper	Management	Data Mapper — consumes from data, publishes to data-mapped
ml1	Management	Manipulator ML1 — consumes from data-mapped-ml1, publishes to data-processed

© AC3 2023

ml2	Management	Manipulator ML2 — consumes from data-mapped-ml2, publishes to data-processed

Enabled plugins

- **rabbitmq_web_stomp:** Exposes the broker over WebSocket/STOMP, enabling browser-based or web-tier clients to connect directly.
- **rabbitmq_prometheus:** Exposes native Prometheus metrics for broker-level observability (queue depths, message rates, connection counts, channel states).

A Docker HEALTHCHECK is configured using rabbitmq-diagnostics -q ping to allow container orchestrators to detect broker readiness before routing traffic to dependent services.

4.2.21.2 Repository Information

- **Repository name:** data-management
- **URL:** https://github.com/EU-AC3/data-management
- **Release model:** OSS
- **Repository visibility:** Public
- **License:** Apache License Version 2.0
- **Main branch:** master

4.2.21.3 Integration in AC³

Integrated in which UC(s)

The UC1 Message Broker is integrated in **UC1** only.

Deployment domain

The message broker is deployed in the **Edge domain** (Edge-LMS-UC1), centralising all intra-pipeline message routing at the edge. Co-location with the mapper and manipulator services eliminates cross-domain latency for the internal data path.

Interaction with other AC3 software modules

- **UC1 Data Consumer Connector (4.2.8.3):** The HTTP-AMQP bridge (ac3-amqp-http-request-logger) publishes incoming EDC-transferred data to the data exchange using the connector AMQP user.
- **UC1 Data Mapper (4.2.8.4):** The mapper consumes from the data queue and publishes normalised measurements to the data-mapped exchange using the mapper user.
- **UC1 Data Manipulator (4.2.8.5):** The ML1 and ML2 services consume from data-mapped-ml1 and data-mapped-ml2 respectively using the ml1 and ml2 users, and publish results to the data-processed exchange.
- **LiSO (LCM):** The broker is instantiated and managed as part of the UC1 application descriptor executed by LiSO.
- **Catalogues (4.2.3):** The broker is registered as a service entry in the AC3 catalogue with its container image, port definitions, and environment configuration, enabling reference in OSR-generated application descriptors.
- Resource Exposure Framework / Monitoring: The RabbitMQ Prometheus plugin exposes broker metrics at a Prometheus scrape endpoint, contributing queue-level and connection-level observability data to the AC3 monitoring plane.

4.2.21.4 Documentation and Maintenance

© AC³ 2023

Documentation location

- **README.md in the data-management repository:** Component overview, deployment instructions, and configuration reference.
- **data-management/rabbit/ in the data-management repository:** Dockerfile, definitions.json (full exchange, queue, user, binding, and permission definitions), rabbitmq.conf (port and definitions-import configuration), and build.sh.
- **The definitions.json file is self-documenting:** it explicitly declares all broker objects (users, permissions, exchanges, queues, bindings), making the full message topology visible without requiring additional documentation.

Maintainer

The UC1 Message Broker is developed and maintained by Spark Works (SPW) as part of the AC^3 consortium.

Issue tracking mechanism

GitHub repository issue tracker on the message-broker-uc1 and data-management repositories, supplemented by consortium-internal coordination channels.

© AC3 2023

5 Final Software Integration per Use Case

5.1 Use Case 1 – Final Integrated State

5.1.1 Deployed Software modules and Operational Status

The following section provides an overview of the software modules deployed within the use case environment. These modules represent the core application components, supporting services, and infrastructure elements required to enable the execution, orchestration, and monitoring of the system.

Table 10: UC1 deployed software modules and operational status.

Software Module	Repository	Status
GUI	https://github.com/EU-AC3/gui-core-uc1-uc2-uc3	Operational
OSR	https://github.com/EU-AC3/osr-core-uc1-uc2-uc3	Operational
Data and Service Catalogue	https://github.com/EU-AC3/catalogue-application-template-uc1	Operational
Edge LMS - UC1	https://github.com/EU-AC3/lms-edge-uc1	Operational
Cloud LMS - UC1	https://github.com/EU-AC3/lms-cloud-uc1	Operational
Data Source LMS - UC1	https://github.com/EU-AC3/lms-datasource-uc1	Operational
LiSO	https://github.com/EU-AC3/lcm-liso-uc1-uc2	Operational
Resource Exposer	https://github.com/EU-AC3/resource-exposer-core-uc1-uc2-uc3	Operational
Resource Discovery	https://github.com/EU-AC3/discovery	Operational
Migration Algorithm UC1	https://github.com/EU-AC3/migration-algorithm-uc1	Operational
AI-based Application Profile	https://github.com/EU-AC3/ai-application-profile-core-uc1-uc2-uc3	Operational
Data Provider Connector UC1	https://github.com/EU-AC3/data-management	Operational
Data Consumer Connector UC1	https://github.com/EU-AC3/data-management	Operational
Data Mappers	https://github.com/EU-AC3/data-management	Operational
Data Manipulator UC1	https://github.com/EU-AC3/use-case-1/tree/main/application/manipulator	Operational
Message Broker UC1	https://github.com/EU-AC3/data-management	Operational

© AC3 2023

5.1.2 Short description of integration topology (Domain distribution (Edge / Cloud / Data Source))

The UC1 final integrated topology follows the AC3 continuum architecture and combines three execution domains. The domains are the Data Source, Edge, and Cloud with a set of cross-domain CECCM components that support application definition, orchestration, monitoring, and runtime adaptation. In this topology, the IoT sensing infrastructure is in the data-source domain, the latency-sensitive application microservices are primarily executed in the edge domain, and complementary or overflow services are supported by the cloud domain. The GUI, OSR, Data and Service Catalogue, LiSO, Migration Algorithm UC1, AI-based Application Profile, and Resource Exposer collectively coordinate the lifecycle of the application across the continuum. This organization is consistent with the UC1 architecture and functional mapping described in D5.1 [1] and with the integration workflow detailed in D5.2 [2]. Figure 2 illustrates the overall UC1 topology based on the architecture defined in D5.1 [1].

The Data Source Domain corresponds to the monitored smart-building environment and contains the physical sensors and local collection nodes that generate the UC1 data stream. In software terms, this domain is represented by Data Source LMS - UC1 together with the Data Provider Connector UC1, which exposes the IoT stream as a managed data source for the rest of the AC3 chain. The Data and Service Catalogue complement this domain by storing the corresponding data-source metadata and service descriptions, allowing the dataset and its required connector services to be discovered and referenced during application composition.

The Edge Domain is the primary execution environment for the UC1 operational pipeline and is implemented through Edge LMS - UC1. It hosts the latency-sensitive microservices that process the incoming building data close to the source, notably the Data Consumer Connector UC1, Message Broker UC1, Data Mappers, and Data Manipulator UC1, which implement the anomaly-detection and forecasting logic of the application. This edge-side placement is central to UC1 because it reduces communication overhead, improves reaction time, and supports near-real-time monitoring and decision-making. The Resource Exposer and monitoring-related functions also contribute by exposing infrastructure and application KPIs to the management plane.

The Cloud Domain is implemented through Cloud LMS - UC1 and complements the edge environment by offering centralized computing capacity, persistent service hosting, and a target environment for workload redistribution when edge resources are insufficient or when broader operational visibility is required. In UC1, the cloud domain is therefore not merely a backup site, but an active part of the continuum that can host selected services and support the migration of application components from edge to cloud when SLA preservation or resource constraints require it.

Across these three domains, the user-plane and management-plane components form the control chain of the UC1 deployment. The GUI provides the entry point for the application developer, the OSR translates the application definition into a deployment-ready descriptor using information retrieved from the Data and Service Catalogue, and LiSO deploys and manages the application across the Edge LMS - UC1 and Cloud LMS - UC1 environments. The Migration Algorithm UC1, AI-based Application Profile, and the monitoring/Resource Exposer functions provide the intelligence needed for placement adaptation, resource-aware decisions, and zero-touch operation across the continuum. In this sense, the final UC1 topology should be understood as a coordinated AC3 software chain rather than only a three-domain infrastructure layout.

© AC3 2023

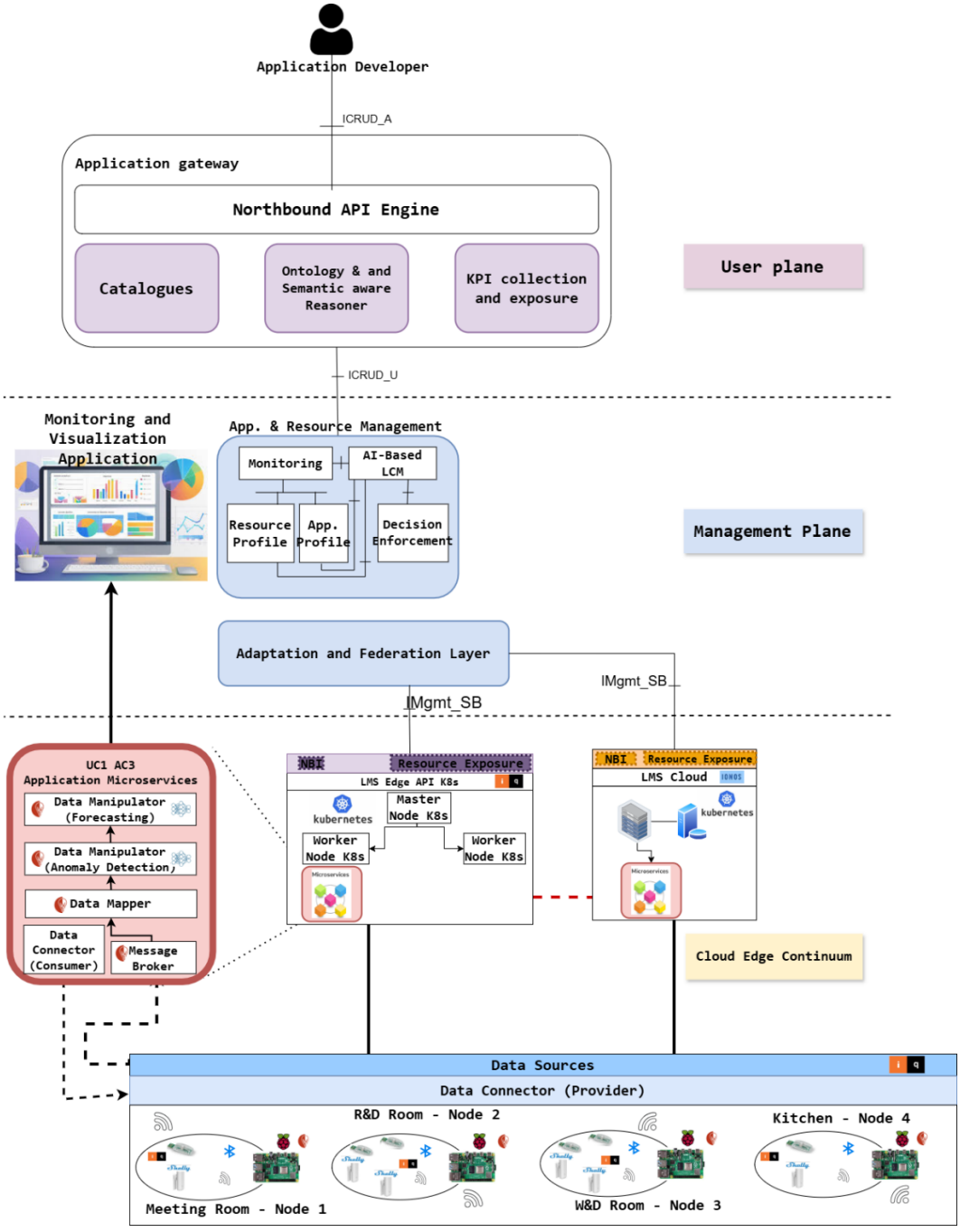

Figure 2: UC1 architecture following the AC³ general architecture paradigm [1].

End-to-end, the operational flow starts with data generation in the Data Source Domain, continues through the Data Provider Connector UC1 and Data Consumer Connector UC1, passes via the Message Broker UC1 and Data Mappers to the Data Manipulator UC1 for processing, and is then exposed to visualization, monitoring, and control functions. This arrangement combines the UC1-specific data path with the AC3 common orchestration and management components, thereby realizing the final integrated state of UC1 across Data Source, Edge, and Cloud

© AC³ 2023

5.1.3 UC1 Execution Domains and Supporting LMS Environments

The UC1 deployment relies on three supporting execution domains: the Edge Domain, the Cloud Domain, and the Data Source Domain. These domains are not reported as standalone AC3 software components. Instead, they provide the runtime environments and operational context in which the UC1 software modules are deployed, executed, and integrated.

This distinction is important because the LMS environments are based on existing Kubernetes-based infrastructure and therefore should not be interpreted as software developed by AC3. Their role in D5.4 is to document the final UC1 deployment setup and to show where the AC3 software modules are executed across the cloud-edge-data-source continuum.

5.1.3.1 UC1 Edge Domain

The UC1 Edge Domain provides the edge-side execution environment used to host latency-sensitive UC1 microservices close to the IoT data source. It supports local execution of data-management and processing components, reducing communication overhead and enabling faster processing of the sensor data produced in the monitored environment.

In the final UC1 integrated setup, the Edge Domain is used to host software modules such as the Data Consumer Connector, Message Broker, Data Mapper, and Data Manipulator services. These components operate close to the data source and form the main edge-side processing pipeline of UC1.

A repository is maintained to provide deployment evidence and reproducibility information for this environment:

- **Repository name:** lms-edge-uc1
- **URL:** https://github.com/EU-AC3/lms-edge-uc1
- **Purpose:** UC1 edge-domain deployment evidence and configuration documentation
- **Maintainer:** IQUADRAT

The repository is intended to support review, traceability, and reproducibility of the UC1 edge environment. Low-level infrastructure details, scripts, and configuration files are maintained in the repository documentation rather than repeated in this deliverable.

5.1.3.2 UC1 Cloud Domain

The UC1 Cloud Domain provides the cloud-side execution environment used to host UC1 services that require centralised execution, persistent availability, or coordination with other components of the AC3 framework. It complements the Edge Domain by supporting workload distribution and providing an execution target for services that do not need to remain close to the physical data source.

In the final UC1 deployment, the Cloud Domain supports the cloud-side operation of lifecycle-managed services, monitoring-related functions, and components involved in orchestration and coordination. It also provides the context for possible workload relocation between edge and cloud when required by the UC1 integration scenario.

A repository is maintained to provide deployment evidence and reproducibility information for this environment:

- **Repository name:** lms-cloud-uc1
- **URL:** https://github.com/EU-AC3/lms-cloud-uc1
- **Purpose:** UC1 cloud-domain deployment evidence and configuration documentation
- **Maintainer:** IQU

© AC3 2023

As with the Edge Domain, this repository documents the deployed environment and supports reproducibility. Detailed infrastructure configuration is kept in the repository documentation and is not reproduced in D5.4.

5.1.3.3 UC1 Data Source Domain

The UC1 Data Source Domain implements the source-side environment responsible for collecting and exposing the real-time IoT data stream used by the UC1 smart sensing and monitoring application. This domain is deployed on Raspberry Pi devices installed in the IQU offices and connected to the physical sensing infrastructure.

The Data Source Domain interfaces with the following sensor types:

- Sensirion SCD41 sensors, used for environmental measurements such as CO_2 concentration, temperature, and humidity.
- Shelly Motion 2 sensors, used for motion and occupancy-related measurements.

The role of this domain is to acquire sensor data, normalise it, and make it available to the downstream UC1 data-management chain. It therefore acts as the bridge between the physical sensing infrastructure and the AC3 software ecosystem.

A repository is maintained to document the UC1 data-source software stack:

- **Repository name:** lms-datasource-uc1
- **URL:** https://github.com/EU-AC3/lms-datasource-uc1
- **Purpose:** UC1 data-source domain software and deployment documentation
- **Maintainer:** IQU

Unlike the Edge and Cloud Domains, this repository includes source-side data acquisition software. However, in this deliverable it is presented as part of the UC1 deployment environment rather than as a separate CECCM software component, because its purpose is specific to the UC1 data-source setup.

Role in the UC1 Integrated Architecture

Together, the UC1 Edge, Cloud, and Data Source Domains provide the runtime foundation for the final UC1 software integration. The Data Source Domain generates the live IoT stream, the Edge Domain executes the main local processing pipeline, and the Cloud Domain provides complementary execution and coordination capabilities.

This organisation supports the final UC1 integration objectives by enabling local data processing near the source, reducing unnecessary data transfer, and maintaining the possibility of distributing or relocating workloads between edge and cloud environments. The detailed deployment and configuration material for these domains is maintained in the corresponding repositories and README files.

5.1.4 Adjustments and Configuration

Several integration adjustments were required to reach the final UC1 software-integrated state:

- **Separation into Data Source, Edge, and Cloud domains:** The final UC1 deployment was intentionally structured across three distinct domains rather than as a single consolidated environment. This adjustment was necessary to preserve the AC3 cloud-edge continuum model and to reflect the actual operational roles of the software chain: data generation in the Data Source LMS - UC1 domain, low-latency processing in Edge LMS - UC1, and complementary execution capacity in Cloud LMS - UC1.
- **Adoption of a common Kubernetes-based LMS model for edge and cloud execution:** Both execution domains were aligned around Kubernetes-based LMS environments in order to provide a uniform deployment and lifecycle-management substrate for UC1 microservices. This common execution model

© AC3 2023

simplifies descriptor translation, deployment consistency, and runtime management across domains, while also making coordinated placement and migration decisions more tractable for the orchestration chain.

- **Intent-driven orchestration through GUI, OSR, Data and Service Catalogue, and LiSO:** A key integration adjustment was the consolidation of the deployment flow around the AC3 control chain. Instead of deploying UC1 services as a standalone application stack, the final integration uses the GUI as the application entry point, the OSR to generate the application descriptor, the Data and Service Catalogue to provide reusable service and data descriptions, and LiSO to translate the resulting AppD into deployable resource objects for the target LMS. This turns UC1 into a CECCM-managed deployment rather than a manually assembled microservice bundle.
- **Modularization of the data path into deployable AC3 data-management components:** The UC1 processing chain was decomposed into separate software modules so that each function in the data path could be independently described, deployed, and maintained. The resulting flow comprises the Data Provider Connector UC1, Data Consumer Connector UC1, Message Broker UC1, Data Mappers, and Data Manipulator UC1. This modularization was also an integration requirement, since it allowed the data pipeline to be represented through catalogue entries, included in the OSR-generated descriptor, and deployed across the target LMS environments through the LCM chain.
- **Explicit support for cloud-side overflow processing and service continuity: The** cloud domain was integrated as an active execution environment rather than a passive backup. In UC1, this was necessary because the edge domain is preferred for low-latency operation, but it may not always be sufficient under high data volumes or tighter resource constraints. The cloud-side LMS therefore provides complementary computing capacity and supports continuity of service when processing must be shifted away from the edge.
- **Migration-oriented runtime model supported by LiSO and the Migration Algorithm UC1:** The final UC1 integration preserves migration as a built-in runtime capability rather than treating deployment as static placement. LiSO remains responsible for lifecycle management and decision enforcement, while the Migration Algorithm UC1 contributes the logic for deciding when microservices should be moved between execution environments. This is central to the AC3 value proposition in UC1, since the application is expected to adapt to changing resource conditions, latency constraints, and workload characteristics across edge and cloud domains.
- **Integration of monitoring, Resource Exposer, and AI-based Application Profile into the deployment logic:** Monitoring, resource profiling, and application profiling were integrated as management-plane inputs that support orchestration and runtime adaptation decisions. The inclusion of the Resource Exposer and the AI-based Application Profile ensures that runtime information from the deployed environments can be exposed to the management layer and used to support placement, migration, and zero-touch operation.

Overall, these adjustments transformed UC1 from a set of locally deployable application components into a CECCM-managed, domain-distributed software chain in which application definition, data access, microservice deployment, runtime control, and migration support are coherently integrated across the Data Source, Edge, and Cloud environments.

5.1.5 Deployment and Replication Guidelines

UC1 deployment follows a domain-distributed, descriptor-driven approach in which the data-source environment, the Edge LMS - UC1, and the Cloud LMS - UC1 are combined through the AC^3 orchestration chain rather than deployed as an isolated application stack. In practical terms, replication of UC1 requires reproducing both the three-domain runtime structure and the software control path formed by the GUI, OSR, Data and

Service Catalogue, LiSO, and the UC1 data-management modules. This deployment logic is consistent with the UC1 deployment flow described in D5.1 [1] and D5.2 [2], where the application is defined through the CECCM, translated into an application descriptor, and then deployed to the appropriate LMS environment.

- **Platform layer: The execution substrate of UC1 is formed by two Kubernetes-based LMS environments:** Edge LMS - UC1 and Cloud LMS - UC1. The edge LMS provides the preferred runtime location for low-latency execution close to the IoT source, while the cloud LMS provides centralized execution capacity and supports overflow processing or service continuity when edge-side resources are insufficient. The Data Source LMS - UC1 domain remains separate from these execution environments and must be connected to them through the UC1 data-access chain.
- **User and descriptor layer:** Deployment begins with application definition through the GUI, which is the entry point for the developer to define the UC1 application components, dependencies, and SLA-related requirements. The OSR then processes this information, queries the relevant catalogued service and data descriptions from the Data and Service Catalogue, and generates the application descriptor required by the orchestration chain. Reproducing UC1 therefore requires not only the deployment artefacts, but also the descriptor-generation workflow that binds together application logic, deployment requirements, and data dependencies.
- **LCM layer:** LiSO acts as the lifecycle-management component responsible for taking the descriptor generated by the OSR and deploying the application to the appropriate LMS domain. LiSO interacts with the edge and cloud LMS environments and manages the lifecycle of the deployed services, including placement, runtime adaptation, and migration support. For replication, LiSO or an equivalent LCM path must therefore be able to consume the UC1 descriptor and target both LMS domains consistently.
- **Data layer:** The UC1 data path is established through a modular chain consisting of the Data Provider Connector UC1, Data Consumer Connector UC1, Message Broker UC1, Data Mappers, and Data Manipulator UC1. The provider connector exposes the IoT stream from the data-source side, the consumer connector receives the data into the execution domain, the message broker distributes it internally, the mapper transforms it into the required internal format, and the manipulator performs the core application processing. Replication therefore requires not only the deployment of these modules, but also the consistent configuration of their interfaces, endpoints, ports, queues, and environment variables so that the end-to-end data flow remains intact.
- **Migration and runtime adaptation:** The UC1 deployment model is not static. The Migration Algorithm UC1 and the wider AI-based LCM chain support runtime relocation of microservices between edge and cloud according to resource conditions, latency requirements, and application behaviour. For this reason, faithful replication of UC1 should preserve both execution environments and the monitoring/Resource Exposer inputs needed for placement adaptation. Replicating only the edge side or only the cloud side would reproduce only a partial setup, not the intended final integrated state.

For practical replication, the minimum requirements are the following:

- A Data Source LMS - UC1 domain capable of generating or exposing the UC1 sensor stream and interfacing it through the provider-side connector chain.
- An edge Kubernetes environment (Edge LMS - UC1) able to host the latency-sensitive UC1 processing modules and act as the primary LMS execution domain.
- A cloud Kubernetes environment (Cloud LMS - UC1) able to host complementary services and receive migrated or overflow workloads when required.
- A descriptor-generation and orchestration path including the GUI, OSR, Data and Service Catalogue, and LiSO, so that deployment follows the AC3-managed workflow rather than ad hoc manual placement.

© AC3 2023

- The UC1 data-management software modules required to recreate the operational pipeline from data acquisition to processing and message exchange.

The UC1 LMS repositories for edge and cloud provide the technical evidence and configuration structure needed to reproduce the two main Kubernetes execution domains, while the UC1 software repositories for connectors, mapper, manipulator, and broker provide the deployable application elements needed to reconstruct the processing pipeline itself. Taken together, these artefacts allow the UC1 final integrated state to be replicated as a continuum-based software deployment.

5.2 Use Case 2 – Final Integrated State

5.2.1 Deployed Software Modules and Operational Status

The following section provides an overview of the software modules deployed within the use case environment. These modules represent the core application components, supporting services, and infrastructure elements required to enable the execution, orchestration, and monitoring of the system.

Software Module	Repository	Status
GUI	https://github.com/EU-AC3/gui-core-uc1-uc2-uc3	Operational
OSR	https://github.com/EU-AC3/osr-core-uc1-uc2-uc3	Operational
LMS Networking UC2	https://github.com/EU-AC3/lms-networking-uc2/	Operational
Edge LMS - UC2	https://github.com/EU-AC3/lms-edge-uc2	Operational
Cloud LMS - UC1	https://github.com/EU-AC3/lms-cloud-uc2	Operational
Sdwan controller	https://github.com/EU-AC3/sdwan_controller	Operational
LiSO	https://github.com/EU-AC3/lcm-liso-uc1-uc2	Operational
Resource Exposer	https://github.com/EU-AC3/resource-exposer-core-uc1-uc2-uc3	Operational
Resource Discovery	https://github.com/EU-AC3/discovery	Operational
Migration Algorithm UC2	https://github.com/EU-AC3/migration-algorithm-uc2	Operational
AI-based Application Profile	https://github.com/EU-AC3/ai-application-profile-core-uc1-uc2-uc3	Operational

5.2.2 Short description of integration topology (Domain distribution (Edge / Cloud / Data Source))

UC2 demonstrates a smart monitoring system integrating Unmanned Aerial Vehicles (UAVs), IoT devices, AI-based video analytics, and edge computing to support urban surveillance, traffic monitoring, and environmental

sensing. The application follows a microservice-based architecture deployed across the cloud–edge–far-edge continuum, enabling real-time video processing close to data sources while maintaining centralized cloud services.

The UC2 testbed spans two main locations: the IONOS cloud infrastructure in Germany and the EUR edge and far-edge infrastructure in France, interconnected through SD-WAN networking. Each infrastructure hosts software components according to its role and resource requirements.

The IONOS cloud region is divided into multiple clusters. A management cluster hosts orchestration components, including the AI-based Lifecycle Manager (LiSO) and the SD-WAN controller, responsible for application lifecycle management and network configuration. A cloud cluster runs Kubernetes and hosts components such as the Local Management System (LMS), the VIM adaptation agent, and user-facing services including the frontend and gateway. These services are internet-accessible but do not require low-latency edge connectivity.

The EUR edge region consists of a K3s edge cluster hosting orchestration components and several application microservices requiring proximity to the data sources. Due to its higher resource capacity, this node also hosts the database backend and additional processing services. An SD-WAN edge node ensures connectivity with the cloud and other clusters.

The far-edge environment is represented by a UAV-mounted NVIDIA Jetson device running a single-node K3s cluster connected through a 5G base station. This node hosts image and video processing microservices that perform real-time AI inference directly on video streams using the Jetson GPU. Deploying these services at the far edge reduces latency and minimizes data transfer across the infrastructure.

Together, these infrastructures form a distributed environment where management and user-facing services run in the cloud, data processing and storage components operate at the edge, and real-time AI analytics execute on far-edge UAV devices. The overall deployment setup and interactions between infrastructures are illustrated in Figure 3.

© AC3 2023

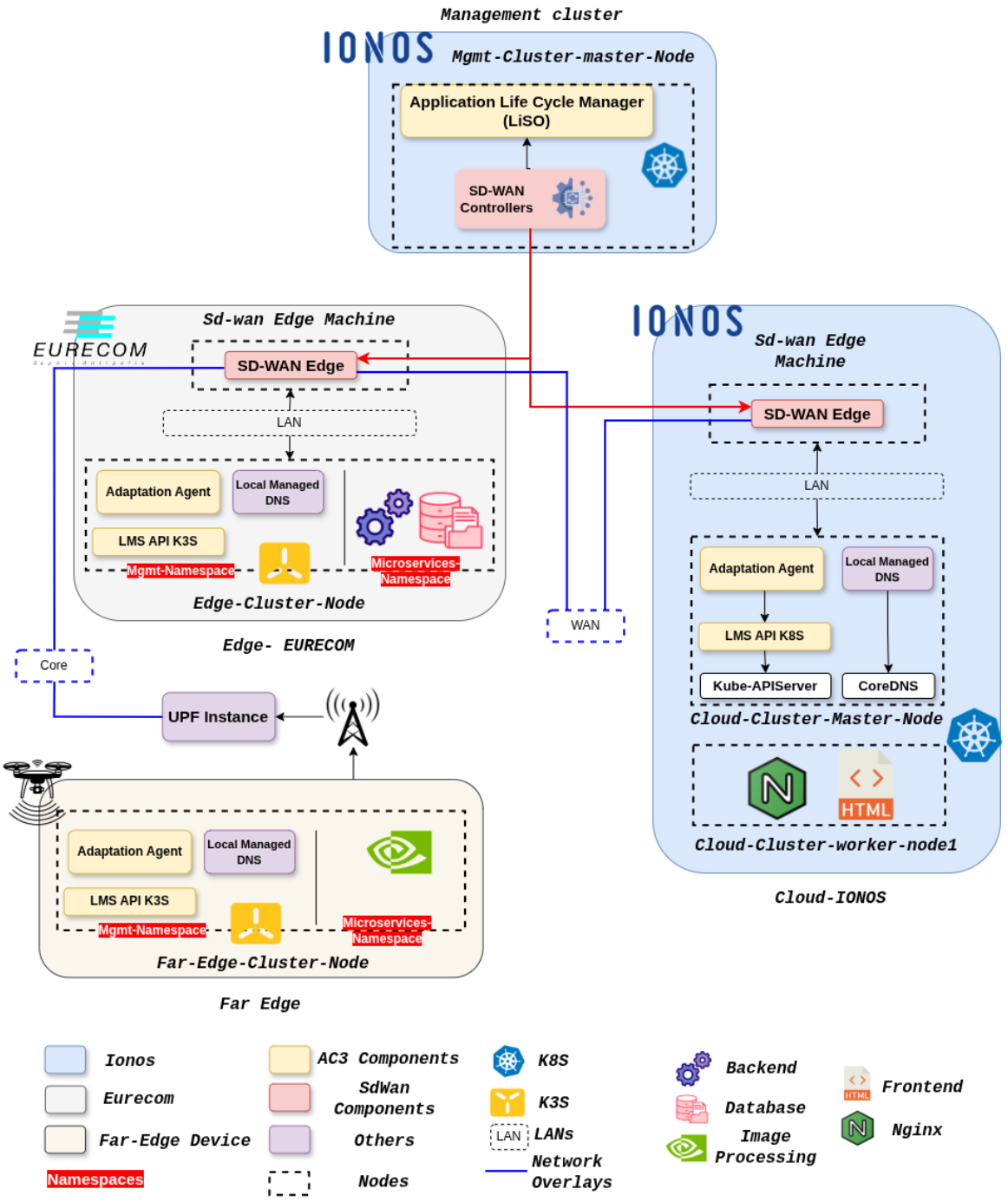

Figure 3: UC2 testbed architecture showing the distributed cloud–edge–far-edge setup [2].

5.2.3 Adjustments and configurations

UC2 spans multiple domains: a cloud domain geographically located in Germany, along with edge and far-edge domains located in France. All three domains are managed by LiSO at the LCM level. The runtime infrastructures are handled through LiSO's LMS plugins, which are specialized software components designed to interface with the underlying infrastructure technologies.

© AC³ 2023

The three domains are interconnected using EURECOM's SD-WAN controller, which enables programmatic network management. Through SD-WAN edge plugins, the controller can create, configure, and delete overlay networks dynamically in response to LiSO requests.

A repository is maintained to provide the software implementation, deployment evidence, and reproducibility information for this component:

- **Repository name:** lcm-liso-uc1-uc2
- **URL:** https://github.com/EU-AC3/lcm-liso-uc1-uc2
- **Purpose:** UC2 Domains management, configuration and connexion
- **Maintainer:** EUR

5.2.4 Adjustments and Configuration

Several adjustments and configurations were introduced during the final integration phase of UC2.

- **Integration of the post-migration feature into LiSO:** To enforce the decisions produced by the proactive migration algorithm, a post-migration mechanism was integrated into LiSO. This feature enables microservice migration with minimal service interruption and reduced migration time.
- **Persistent-volume management in LiSO and LMS plugins:** To support the handling of application data streams and prediction models stored in the UC2 databases, persistent-volume management capabilities were added to LiSO and its LMS plugins. New LiSO endpoints were implemented to manage Kubernetes Persistent Volumes (PVs) and Persistent Volume Claims (PVCs), particularly for hostPath-based storage configurations.
- **RTSP-based streaming support:** To support drone-specific camera systems, the application streaming layer was adapted to use the RTSP protocol. This modification enabled compatibility with real-time video streams produced by the targeted camera devices.
- **Continuous learning and fine-tuning:** To improve the accuracy of the proactive migration predictions, the AI model was retrained using infrastructure-specific data collected from the far-edge environment. A continuous fine-tuning service was proposed to continuously adapt and enhance the prediction model based on runtime observations and updated infrastructure behaviour.

5.2.5 Deployment and Replication Guidelines

UC2 deployment follows a domain-distributed, descriptor-driven approach in which the Cloud LMS - UC2, the Edge LMS - UC2, and the Far Edge LMS - UC2 are combined through the AC3 orchestration chain rather than deployed as isolated application stacks. In practical terms, replication of UC2 requires reproducing runtime structure and the software control path formed by the GUI, OSR, LiSO, and the underlying infrastructure plugins.

- **Platform layer:** The execution substrate of UC2 is formed by three Kubernetes-based LMS environments: Cloud LMS - UC2, Edge LMS - UC2, and Far Edge LMS - UC2. The cloud LMS provides centralized execution capacity, while the edge and far-edge LMS domains provide low-latency execution closer to the operational environment (specifically for the drone object detection model). The three domains are geographically distributed across Germany and France and are interconnected through EURECOM's SD-WAN infrastructure.
- **User and descriptor layer:** Deployment begins with application definition through the GUI, which acts as the entry point for developers to define the UC2 application components, dependencies, and SLA-related requirements. The OSR processes this information and generates the application descriptor required by the orchestration chain. Reproducing UC2 therefore requires not only the deployment artefacts, but also

© AC3 2023

the descriptor-generation workflow that binds together application logic, deployment requirements, and infrastructure constraints.

- **LCM layer:** LiSO acts as the lifecycle-management component responsible for consuming the descriptor generated by the OSR and deploying the application across the appropriate LMS domains. All three domains are managed by LiSO at the LCM level. Runtime infrastructure management is performed through LiSO LMS plugins, which are specialized software components designed to interface directly with the underlying infrastructure technologies. LiSO manages service lifecycle operations including deployment, placement, runtime adaptation, scaling, and migration support across the cloud, edge, and far-edge environments.
- **Network orchestration layer:** The three LMS domains are interconnected through EURECOM's SD-WAN controller, which enables programmatic network management between geographically distributed sites. Through dedicated SD-WAN edge plugins, the controller dynamically creates, configures, and removes overlay networks according to requests issued by LiSO. This integration allows the orchestration layer to coordinate both compute and network resources as part of the same deployment workflow.
- **Migration and runtime adaptation:** The AI-driven lifecycle-management chain supports runtime relocation of microservices between cloud, edge domains according to resource availability. The migration algorithm should be deployed at the same time with liso and the other infrastructure components such us the resource discovery and the exposer.

Instructions on how to reproduce the setup are in the EU-AC3/use-case-2 repository.

5.3 Use Case 3 – Final Integrated State

5.3.1 Deployed Software and modules and Operational Status

The following section provides an overview of the software modules deployed within the use case environment. These modules represent the core application components, supporting services, and infrastructure elements required to enable the execution, orchestration, and monitoring of the system.

Table 11: List of software modules deployed for UC3

Software Module	Repository	Status
UC3 Backend	https://github.com/EU-AC3/use-case-3	Operational
Starlight Application	https://github.com/EU-AC3/use-case-3	Operational
pPXF Application	https://github.com/EU-AC3/use-case-3	Operational
Voronoi Application	https://github.com/EU-AC3/use-case-3	Operational
Frontend GUI	https://github.com/EU-AC3/use-case-3	Operational
RabbitMQ	https://github.com/EU-AC3/use-case-3	Operational
Redis	https://github.com/EU-AC3/use-case-3	Operational
Consumer Connector	https://github.com/EU-AC3/data-provider-conector-uc3	Operational
Transfer Trigger	https://github.com/EU-AC3/data-provider-conector-uc3	Operational

© AC3 2023

HashiCorp Vault	https://github.com/EU-AC3/data-provider-conector-uc3	Operational
HScaler Predictor Client	https://github.com/EU-AC3/use-case-3	Operational
HScaler Model API	https://github.com/EU-AC3/use-case-3	Operational
Prometheus Adapter	https://github.com/EU-AC3/use-case-3	Operational
AC3 Network Operator	https://github.com/EU-AC3/lms-networking-uc3	Operational
P2CODE Scheduler	https://github.com/rh-waterford-et/p2code_scheduler_operator	Operational
MaestroLCM	UBITECH-managed	Operational

5.3.2 Short description of integration topology (Domain distribution (Edge / Cloud / Data Source))

The UC3 testbed spans six environments, comprising four OpenShift clusters and two external services:

- **OSR:** Deployed externally by FIN on the ION cloud infrastructure. It provides the developer-facing GUI and dispatches application descriptors to Maestro via the Translator API.
- **LCM Cluster:** A Kubernetes cluster that hosts the Maestro LCM platform on UBITECH infrastructure. Maestro runs here as an independent service, receiving deployment requests from the OSR and submitting them to ACM for deployment to the application cluster.
- **ACM Hub Cluster:** A single-node OpenShift cluster running RHAT Advanced Cluster Management. The ACM Hub manages the two application clusters, distributes ManifestWork resources, and hosts the AC^3 Network Operator and P2CODE Scheduler operator. The Prometheus/Thanos federation layer also operates from this cluster, aggregating metrics from the managed clusters.
- **Application Clusters (2x managed clusters):** Two OpenShift clusters, each with 24 vCores, 64 GB RAM, and 200 GB SSD where the application microservices are deployed.
- **UCM Cluster:** Hosts the IONOS-managed EDC provider connector and Vault instance that serve as the data source. S3 object storage holds the raw astronomy datasets. The transfer trigger on the producer cluster initiates contract negotiation and data transfer from this environment.

Inter-cluster connectivity between the producer and processor clusters is provided by Skupper, managed by the AC^3 Network Operator running on the ACM Hub. RabbitMQ on the producer cluster is exposed to the processor cluster through the Skupper virtual application network, enabling processor pods to consume jobs from the central event queue.

Data flows from the UCM provider (ION S3) through the EDC connector to the RHT consumer S3 bucket, from where the UC3 backend retrieves it for processing.

© AC3 2023

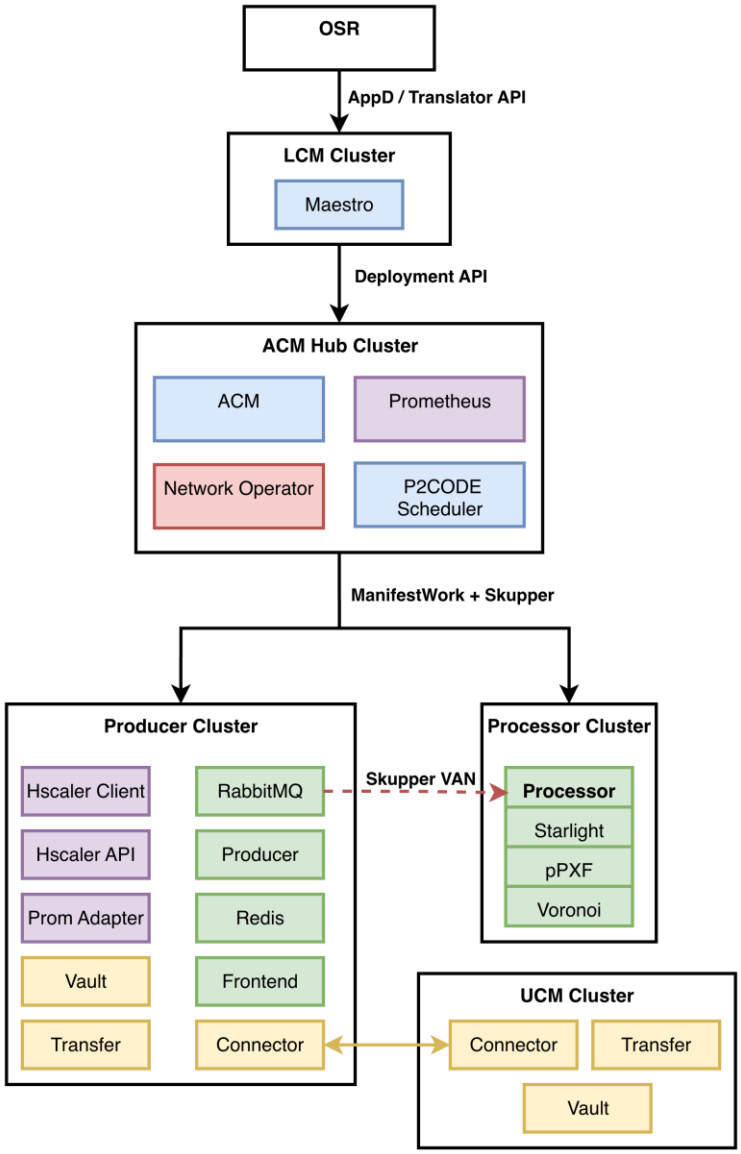

Figure 4: UC3 Topology

5.3.3 UC3 Execution Domains and Supporting LMS Environments

The UC3 Multi-Cluster Networking Domain provides the networking support required to connect UC3 microservices deployed across separate OpenShift clusters. It enables distributed application components to communicate as part of the same UC3 processing workflow.

In the final UC3 integrated setup, this domain is supported by the AC³ Network Operator, which automates inter-cluster communication using Skupper. Its main role is to expose RabbitMQ across cluster boundaries, allowing processor pods running on remote clusters to consume jobs from the central event queue.

A repository is maintained to provide the software implementation, deployment evidence, and reproducibility information for this component:

- **Repository name:** lms-networking-uc3
- **URL:** https://github.com/EU-AC3/lms-networking-uc3
- **Purpose:** UC3 inter-cluster networking support and deployment documentation
- **Maintainer:** Red Hat Waterford Emerging Technologies

The repository is intended to support review, traceability, and reproducibility of the UC3 networking setup. Low-level deployment details, scripts, and configuration files are maintained in the repository documentation rather than repeated in this deliverable.

5.3.4 Adjustments and Configuration

Several integration adjustments were made during the final integration phase:

- **Skupper Service Exposure Strategy**: The AC^3 Network Operator implements two exposure strategies (direct annotation vs. proxy/ExternalName) to handle services deployed by ACM ManifestWork resources, which cannot be directly annotated due to owner reference constraints. This was identified during integration testing when ACM-deployed RabbitMQ services could not be annotated with Skupper proxy annotations.
- **Multi-Container Pod Design:** The UC3 backend uses a single Go binary with multiple operational modes rather than separate images per microservice role. This simplifies image management and ensures consistency. Producer and processor pods run multiple containers (4 containers in the producer pod, 2 in the processor pod) sharing a persistent volume.
- **Application-Specific Message Formats:** The RabbitMQ message format was adapted per processing application. Starlight uses text messages, pPXF uses binary messages (to preserve .fits file integrity), and Voronoi uses S3 reference messages (to avoid transmitting large data cubes through the message broker).
- **Prometheus Adapter for External Metrics:** The standard OpenShift monitoring stack does not natively expose application metrics to the HPA via the External Metrics API. A dedicated Prometheus Adapter deployment was configured with custom external rules to bridge predicted_job_time_avg from the HScaler predictor to the Kubernetes external.metrics.k8s.io API.
- **EDC Vault Integration:** The EDC connector requires HashiCorp Vault for managing temporary S3 transfers credentials. Vault is deployed as a co-located pod in development mode with a static root token for the testbed environment.
- **OpenShift Security Context:** The UC3 processing applications (Starlight, pPXF) require privileged security contexts due to their legacy codebases. A dedicated service account (starlight-sa) with a privileged SCC RoleBinding is configured to permit this.

5.3.5 Deployment and Replication Guidelines

UC3 deployment follows a layered approach:

- **Platform Layer:** OpenShift clusters are provisioned with the Prometheus Operator, ACM agent, and Skupper prerequisites. The AC^3 Network Operator and P2CODE Scheduler are deployed on the ACM Hub via make install && make deploy.
- **LCM Layer:** Maestro is deployed on the LCM cluster. The OSR is configured with the MAESTRO_ENDPOINT pointing to the Maestro Translator API.
- Networking Layer: A MultiClusterNetwork custom resource is created on the ACM Hub to establish Skupper links between the producer and processor namespaces. The operator automatically manages token generation, distribution, and service exposure.
- **Application Layer:** Application deployment can proceed via two paths:

© AC3 2023

- **Automated (OSR/Maestro):** The developer defines an application profile in the GUI. The OSR generates an AppD, which MAESTRO translates into Kubernetes manifests and deploys to the target cluster via ACM.
- **Manual:** The deploy.sh script in astroapp/application/deployments/ applies all Kubernetes manifests in the correct order (namespace, volumes, RabbitMQ, Redis, producer, processor, frontend, Vault, EDC connector, transfer trigger, HScaler components, HPA, Prometheus Adapter).

- **Data Layer:** The EDC provider connector is configured at the UCM site. The consumer connector and transfer trigger are deployed on the producer cluster. Contract policies are established via the EDC management API.

For replication, all UC3-specific components are containerised and available from the listed registries.

5.4 Cross-Use Case Software Reuse

This subsection complements the per-use-case integration view by summarising the degree of software reuse across UC1, UC2, and UC3. The objective is to distinguish between software modules that constitute a shared AC^3 backbone and modules that have been implemented specifically for the needs of an individual use case.

Table 44 presents the mapping between software modules and the use cases in which they are deployed.

Table 12: Cross-Use Case Reuse Matrix

Software Module and Executions Domains	UC1	UC2	UC3
GUI	✓	✓	✓
OSR	✓	✓	✓
Data and Service Catalogue	✓		✓
Edge LMS - UC1	✓		
Cloud LMS - UC1	✓		
Data Source LMS - UC1	✓		
Edge LMS- UC2		✓	
Cloud LMS- UC2		✓	
Networking LMS- UC2		✓	
Networking LMS - UC3			✓
Resource Exposer	✓	✓	✓
Resource Discovery	✓	✓	✓

© AC3 2023

LiSO	✓	✓	
MAESTRO			✓
Migration Algorithm UC1	✓		
Migration Algorithm UC2		✓	
Horizontal Autoscaling Components			✓
Scheduler (P2CODE) UC3			✓
AI-based Application Profile	✓	✓	✓
AI-based Resource Predictor			✓
Data Provider Connector UC1	✓		
EDC Connector for IONOS S3 UC3			✓
Data Consumer Connector UC1	✓		
Data Mappers	✓		
Data Manipulator UC1	✓		
Data Manipulator UC3			✓
Message Broker UC1	✓		

The matrix shows that AC^3 relies on a compact shared software backbone reused across all three use cases. Namely the GUI, OSR, Data and Service Catalogue, Resource Exposer and Resource Discovery, and AI-based Application Profile, while the remaining modules are predominantly UC-specific implementations, confirming a modular design that combines common CECCM capabilities with targeted adaptations for each use case.

5.4.1 Identification of Reused Modules

The matrix shows that a limited but important set of software modules is reused across multiple use cases. In particular, the following components are common to UC1, UC2, and UC3:

- GUI
- OSR
- Data and Service Catalogue
- Resource Exposure
- Resource Discovery
- AI-based Application Profile

© AC3 2023

These modules represent the main cross-use-case software backbone of AC^3 and correspond to functions that are not tied to a single deployment scenario but instead support the overall operation of the CECCM across the project.

In addition to this fully shared backbone, the matrix also highlights partial reuse across subsets of use cases. For example, LiSO is reused in UC1 and UC2. This indicates that software reuse in AC^3 is not limited to universally shared components, but also includes selected modules reused where functional requirements are sufficiently similar.

5.4.2 UC-Specific Implementations

Alongside the shared backbone, the matrix confirms that a significant part of the software portfolio remains use-case-specific. This applies to:

- LMS implementations, such as Edge LMS - UC1, Cloud LMS - UC1, Cloud LMS - UC2, and Networking LMS - UC3.
- Use-case-specific orchestration and decision modules, such as MAESTRO, Horizontal Autoscaling Components, and Scheduler (P2CODE) UC3.
- Data handling components, including Data Provider Connector UC1, Data Consumer Connector UC1, Data Mappers, Data Manipulator UC1/UC3, and Message Broker UC1, are tailored to the application and data-flow requirements of a specific UC.

This distribution reflects the intended AC^3 design approach: common framework-level capabilities are reused where possible, while specialised software is introduced where required by the characteristics of the corresponding use case.

Overall, the reuse analysis confirms that AC^3 does not consist of three isolated software stacks. Instead, it combines a shared set of reusable CECCM components with UC-specific implementations, resulting in a software ecosystem that is both modular and adaptable to heterogeneous deployment scenarios.

© AC3 2023

6 Software Governance and Post-Project Sustainability

This section defines the governance framework applied to the software assets delivered within AC^3. It addresses repository ownership, release model, repository visibility, licensing, access procedures, maintenance responsibility, and post-project continuity. This governance perspective complements the technical reporting provided in Sections 4 and 5 by clarifying how the delivered software is managed both during the project handover phase and after the formal end of the project.

The AC^3 software governance model is decentralised. The AC^3 GitHub organization provides a common project-level entry point and supports traceability across the software portfolio, but technical responsibility remains with the partner that developed or maintains each component. This approach reflects the multi-partner nature of the project and allows each organisation to align software release decisions with its exploitation strategy, internal policies, and intellectual property considerations.

6.1 Repository Ownership Model

Each software module developed within AC^3 is owned and maintained by the partner responsible for its implementation. Although many repositories are grouped under the common AC^3 GitHub organization, this does not imply centralised technical ownership of all software assets. Responsibility for each repository remains with the corresponding component owner.

Each responsible partner retains autonomy over:

- source-code maintenance;
- release planning and versioning strategy;
- branch and tag management;
- issue handling and bug fixing;
- selection of the applicable License;
- repository visibility;
- future publication or exploitation decisions;

The AC^3 GitHub organization serves as a common entry point for the software assets reported in this deliverable. It supports discoverability and traceability for reviewers, consortium partners, and authorised stakeholders. However, it does not impose a single uniform release model, License, or visibility policy across all repositories.

For this reason, each detailed component report in Section 4 includes a “Repository Information” block identifying the repository name, URL, release model, repository visibility, License, and main branch. This ensures that ownership and access conditions are clear at the component level.

6.2 Release Model, Repository Visibility and License Policy

The AC^3 software portfolio follows a mixed release model. Some components are released as open-source software, while others remain under restricted access due to intellectual property considerations, exploitation plans, partner-specific policies, or dependency on external software governance.

To avoid ambiguity, AC^3 distinguishes between three separate concepts:

- **Release model:** indicates whether the software is released as OSS, internal software, external OSS, or partner-managed OSS.
- **Repository visibility:** indicates whether the repository is public, private, access-controlled, or externally hosted.

© AC3 2023

- **License:** defines the legal conditions under which the software may be used, modified, and redistributed by users who have access to it.

This distinction is important because repository visibility and License are independent. A private repository may still contain software released under an open-source License, provided that the License applies to authorised users or to a later public release. Conversely, a public repository remains subject to the License terms selected by the responsible partner.

Public repositories are accessible without restriction through the AC³ GitHub organization or through the external repository indicated in the component description. Private or access-controlled repositories require GitHub authentication and explicit authorisation. Access to restricted repositories for reviewers, project officers, or authorised stakeholders is coordinated through the project coordinator and the responsible component owner.

The License reported for each component reflects the information provided by the responsible partner. Where a component is governed by a partner-specific or custom License, the corresponding component description identifies this explicitly. Where a component builds on an external open-source project, the external repository and its applicable License are reported in the relevant component section.

6.3 Access Procedure for Restricted Repositories

For repositories marked as private, access is not granted automatically. Authorised stakeholders requiring access should submit a request through the project coordinator or directly through the responsible component owner, depending on the access procedure agreed within the consortium.

Access requests should identify:

- the repository or component requested;
- the requesting organisation and user account;
- the purpose of the access request;
- the expected access duration, where applicable.

The responsible partner may grant access through GitHub permissions or provide alternative review access, depending on the repository governance model and the sensitivity of the component. Access may be limited to read-only permissions where appropriate.

This procedure ensures that restricted repositories remain available for review and technical traceability while preserving partner-specific exploitation and intellectual property constraints.

6.4 Issue Tracking and Maintenance Responsibility

Maintenance responsibility remains decentralised. Each component owner is responsible for maintaining its own repository, including source-code updates, documentation updates, issue tracking, and bug fixing, where applicable.

For public repositories, GitHub issue trackers may be used to report bugs, request clarifications, or suggest improvements. For private repositories, issue handling may be managed either through the repository issue tracker or through consortium-internal coordination channels. The specific mechanism depends on the repository owner and the access policy applied to the component.

The component-level “Documentation and Maintenance” subsections in Section 4 identify the maintainer and, where applicable, the issue tracking mechanism. This ensures that each software asset has an identifiable responsible organisation after project completion.

© AC³ 2023

6.5 Post-Project Continuity

After completion of the AC^3 project, responsibility for software maintenance remains with the respective repository owners. No centralised maintenance body is established at the project level. The AC^3 GitHub organization will continue to serve as a stable reference point for the repositories hosted under it, subject to the access and maintenance decisions of the responsible partners.

Public repositories are expected to remain accessible through the AC^3 GitHub organization or through the external repositories indicated in this deliverable. Private repositories may follow different post-project paths depending on partner decisions, including:

- remaining internal;
- being released publicly at a later stage;
- being maintained for further research use;
- being integrated into commercial or partner-managed products;
- being superseded by newer versions under partner governance.

This decentralised continuity model ensures that software sustainability is aligned with partner strategies while preserving traceability of the AC^3 technical outputs. The final repository states reported in this deliverable therefore constitute the WP5 software handover snapshot, while future evolution remains under the responsibility of the relevant repository owners.

6.6 External and Partner-Managed Repositories

Some AC^3-relevant components are hosted outside the AC^3 GitHub organization, either because they are maintained by external open-source communities or because they are governed by a responsible partner through an external repository. These components are still reported in D5.4 when they are part of the implemented AC^3 software stack or are required to reproduce the final integrated use-case deployments.

For such components, the repository URL, release model, repository visibility, License, and maintainer are reported in the corresponding component section. The AC^3 consortium does not control the governance of external repositories, but their inclusion in this deliverable ensures traceability between the AC^3 implementation and the software assets used in the final integrated deployments.

© AC3 2023

7 Conclusions

This deliverable provides the final WP5 software-facing report on the AC3 components and testbed integration activities. In line with the Grant Agreement description of D5.4, it consolidates the implemented CECCM software components, their repository-level availability, the responsible partners, the applicable access and licensing conditions, and their role in the final use-case deployments.

The report confirms that the AC3 software portfolio covers the principal functional areas required by the project architecture, including the Application Gateway, OSR, catalogue and data-management services, Local Management System components, resource exposure and discovery functions, zero-touch lifecycle-management mechanisms, migration support, autoscaling, predictive resource-management functions and use-case-specific data-processing components. These assets collectively support the operational realisation of the CECCM across UC1, UC2 and UC3.

From an implementation perspective, the final integrated state demonstrates that the software modules have moved beyond isolated component development and have been organised into deployable use-case configurations. UC1 validates the IoT data-management and edge/cloud processing pipeline, UC2 demonstrates cloud-edge-far-edge orchestration for UAV-based smart monitoring, and UC3 validates multi-cluster execution, networking, data transfer and autoscaling support for data-intensive astronomy workloads.

The deliverable also clarifies the post-project software governance model. The AC3 GitHub organization provides a stable entry point for the software assets where possible, while partner-specific repositories, restricted-access components and selected external dependencies are governed by the responsible organisations. This mixed access model balances openness, reusability and exploitation needs while preserving traceability for reviewers and authorised stakeholders.

Overall, D5.4 closes the WP5 software reporting chain by linking the AC3 implementation work to the final software release and integrated testbed view. It complements D5.1 [1], D5.2 [2] and D5.3 [3] by focusing on software availability, component traceability, deployment readiness and sustainability rather than repeating the planning, intermediate integration status or KPI-oriented validation results already reported elsewhere.

© AC3 2023

8 References

[1] AC³ Project, "D5.1 "Initial integration and proofs of concept plan", 03 February 2025. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2025/10/AC3_D5.1_v.1.0_final.pdf

[2] AC³ Project, "D5.2 "Report on Integration the CECCM", 03 November 2025. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2025/10/AC3_5.2_Review_Final_v1.0_TM.pdf

[3] AC³ Project, "D5.3 "Results and Achievements", 03 April 2026. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2026/06/D5.3_vfinal-1.pdf

[4] AC³ Project, "Description of Action (DoA)", 10 September 2025.

[5] AC³ Project, "D2.1 "1st Release of the CECC framework and CECCM", 31 August 2023. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/02/AC3_D2.1.pdf

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

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

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

[9] AC³ Project, "D3.4"," Final Report on data management for applications in CECC" 28 June 2024. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2026/06/AC3_D3.4_v1.0.pdf

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

[11] AC³ Project, "D4.2 "Initial Report on the resource management procedures over CECC", 30 June 2024. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2026/06/AC3_Deliverable_4_2_Version_12_to_submit-1.pdf

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

[13] P2CODE Project [Online]. Available: https://p2code-project.eu/

© AC³ 2023