



D.3.1 Report on the application LCM in the CEC - Initial

Document Summary Information

| | | | |
|-----------------------------|---|-------------------------------|-------------------|
| Project Identifier | HORIZON-CL4-2022-DATA-01. Project 101093129 | | |
| Project name | Agile and Cognitive Cloud-edge Continuum management | | |
| Acronym | AC ³ | | |
| Start Date | January 1, 2023 | End Date | December 31, 2025 |
| Project URL | www.ac3-project.eu | | |
| Deliverable | D.3.1 Report on the application LCM in the CEC - Initial | | |
| Work Package | WP3 | | |
| Contractual due date | 30/06/2024 | Actual submission date | 30/06/2024 |
| Type | | Dissemination Level | PU |
| Lead Beneficiary | UPR | | |
| Responsible Author | Vrettos Moulos | | |
| Contributors | D. Klonidis, V. Katopodis (UBI), M. Mekki, S. Messaoudi (EUR), A. Kadouma, A. Rasheed, I. Afolabi (FIN), M. Gurdiel, K. Ramantas (IQU), E. Dritsas, S. Sengupta, A. Nikoukar (ION), G. Tsolis, J. Beredimas, P. Theocharaki, A. Kordelass (CTX)), R. Carroll (RHT), A. Ba (IBM), D. Amaxilatis, N. Tsironis (SPA), A. Podimata, G. Koulopoulos, V. Moulos (UPR) | | |
| Peer reviewer(s) | Adlen Ksentini (EUR), Christos Verikoukis (ATH) | | |



AC³ project has received funding from European Union's Horizon Europe research and innovation programme under Grand Agreement No 101093129.

Revision history (including peer reviewing & quality control)

| Version | Issue Date | % Complete | Changes | Contributor(s) |
|---------|------------|------------|---|--|
| v1.0 | 29/02/2024 | 0% | Initial Deliverable Structure | D. Klonidis (UBI), V. Moulos (UPR) |
| V1.1 | 01/03/2024 | 20% | Editing of Section 3 | V. Moulos (UPR) |
| V1.2 | 04/03/2024 | 24% | Restructuring Chapters | D. Klonidis (UBI) |
| V1.3 | 14/03/2024 | 30% | Editing Data models | D.Amaxilatis (SPA) |
| V1.4 | 04/04/2024 | 35% | Proofreading | D. Klonidis (UBI) |
| V1.5 | 11/04/2024 | 45% | Editing Section 4 | A.Kadouma (FIN) |
| V1.6 | 16/04/2024 | 50% | Editing Section 4 | M.Gurdiel(QUA) |
| V1.6.1 | 22/04/2024 | 55% | Editing Section 4 | M.Gurdiel(QUA) |
| V1.7 | 22/04/2024 | 60% | Proofreading | M.Gurdiel (QUA), D. Amaxilatis (SPA) |
| V1.8 | 25/04/2024 | 65% | Editing Section 4 and 5 | E. Dritsas (ATH), D. Klonidis (UBI), A. Kadouma (FIN) |
| V1.9 | 29/04/2024 | 68% | Proofreading | D. Klonidis (UBI), D.Amaxilatis(SPA) |
| V1.9.1 | 02/05/2024 | 70% | Editing Section 4 | A.Kadouma (FIN), I.Afolabi (FIN) |
| V1.9.2 | 08/05/2024 | 73% | Editing Section 5 | E. Dritsas (ATH), A.Meliani (EUR) |
| V2.0 | 09/05/2024 | 80% | Editing Section 2 and Section 3 | V.Moulos (UPR), E.Dritsas (ATH), D. Klonidis (UBI) |
| V2.1 | 10/05/2024 | 81% | Editing Section 4 | A.Kadouma (FIN) |
| V2.2 | 14/05/2024 | 85% | Editing Section 2 and Section 3 | V.Moulos (UPR) |
| V3.0 | 15/05/2024 | 88% | Added Data Catalogue description and interfaces | D.Amaxilatis (SPA), S. Messaoudi (EUR), A. Melliani (EUR), J. Beredimas (CSG), Athanasios |

| | | | | |
|------|------------|------|---|---|
| | | | | Kordelas (CSG) |
| V3.1 | 20/05/2024 | 90% | Editing Section 1, Section 2, and Section 3 | V.Moulos (UPR) |
| V4.0 | 06/06/2024 | 95% | Reviewing all Sections and editing Conclusion | V. Moulos(UPR), E. Dritsas (ATH), A. Podimata (UPR), G. Koulopoulos (UPR) |
| V4.1 | 11/06/2024 | 98% | Final refinements (pre review process) | V. Moulos(UPR) |
| V4.2 | 26/06/2024 | 100% | Address the comments after the review process | V. Moulos, G. Koulopoulos (UPR), E. Dritsas (ATH) |
| V5.0 | 27/06/2-24 | 100% | Compile the last version | V. Moulos (UPR), E. Dritsas (ATH), A. Meliani (EUR) |

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

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

Without derogating from the generality of the foregoing neither the 6G-BRICKS Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

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

Table of Contents

| | |
|--|----|
| Executive Summary | 9 |
| 1 Introduction..... | 10 |
| 1.1 Overview – Purpose and objectives..... | 10 |
| 1.2 Link with other project activities..... | 10 |
| 1.3 Mapping AC ³ Outputs..... | 11 |
| 1.4 Deliverable Overview and Report Structure | 12 |
| 2 The LCM process in the overall architecture..... | 13 |
| 2.1 Main concept and targeted functionalities..... | 13 |
| 2.2 Mapping to the AC ³ architecture | 13 |
| 3 Application descriptor composition mechanism..... | 16 |
| 3.1 Overview of the implementation plan..... | 16 |
| 3.2 Initial developments | 18 |
| 3.2.1 Application descriptor model | 18 |
| 3.2.2 Application descriptor composer | 24 |
| 3.2.3 Visualisation and Interaction | 27 |
| 3.2.4 Service catalogue and data catalogues..... | 28 |
| 3.2.5 User Interfaces..... | 29 |
| 3.3 Planned implementations and extensions for final phase..... | 29 |
| 4 AI-based application profiling mechanism | 31 |
| 4.1 Overview of the implementation plan..... | 31 |
| 4.2 Initial developments | 32 |
| 4.2.1 The Application Descriptor Analysis | 33 |
| 4.2.2 Interconnection with Monitoring Module and Information Collection | 33 |
| 4.2.3 Data Processing..... | 34 |
| 4.2.4 Machine Learning (ML) Model Development..... | 34 |
| 4.2.5 Profiling and Prediction Engine: Dynamic Application Profiling Mechanism | 36 |
| 4.2.6 Profiling Metrics and Parameters..... | 36 |
| 4.2.7 Application Profile Model..... | 38 |
| 4.3 Planned implementations and extensions for final phase..... | 38 |
| 5 AI-based LCM mechanism and functions | 40 |
| 5.1 Overview of implementation plan | 40 |
| 5.2 LCM Functions and initial developments..... | 42 |
| 5.2.1 Smart placement of app micro-services to resources (Placement)..... | 42 |
| 5.2.2 Runtime supervision | 42 |
| 5.2.3 Runtime decision | 43 |
| 5.3 Architecture solution | 46 |
| 5.4 Results..... | 50 |
| 5.5 Planned implementations and extensions for final phase..... | 59 |
| 5.5.1 Current Implementation status | 59 |
| 5.5.2 Resource Exposure Component | 59 |
| 5.5.3 AI-based LCM (Lifecycle Management) | 59 |
| 6 Conclusions..... | 60 |
| 7 References..... | 61 |
| Annex I: Example of an Application Descriptor | 63 |

List of Figures

| | |
|---|----|
| Figure 1 User and Management Plane Architecture (WP3 Components) | 14 |
| Figure 2 Overall architecture design | 17 |
| Figure 3 Visual of the multi-step form interface | 25 |
| Figure 4 Upload interface | 26 |
| Figure 5 User Interface of Model Profile | 27 |
| Figure 6 AI Based Application Profile Solution Architecture | 32 |
| Figure 7 Microservices Based Applications Lifecycle..... | 40 |
| Figure 8 The LCM module and its interaction with the surrounding modules in the AC ³ architecture..... | 41 |
| Figure 9 Model’s Inputs and outputs | 44 |
| Figure 10 Transforming the initial dataset to training dataset | 45 |
| Figure 11 Fault tolerance Migration Algorithm..... | 46 |
| Figure 12 Checkpoint Object Sample | 48 |
| Figure 13 Intra-Cluster Migration Workflow. | 49 |
| Figure 14 Intra-Cluster Migration Workflow. | 50 |
| Figure 15 Benchmarking Results | 52 |
| Figure 16 the total saving time of different container sizes with different numbers of instances..... | 53 |
| Figure 17 Average time to restore container with respect to its size. | 54 |
| Figure 18 RL agent | 58 |

List of Tables

| | |
|---|----|
| Table 1 Adherence to AC ³ GA Deliverable & Tasks Descriptions..... | 11 |
| Table 2 Parameters..... | 37 |
| Table 3 Results from Different Model Instances..... | 45 |

Glossary of terms and abbreviations used

| Abbreviation / Term | Description |
|---------------------|--|
| AC ³ | Agile and Cognitive Cloud edge Continuum management |
| AES | Advanced Encryption Standard |
| AI | Artificial Intelligence |
| AMQP | Advanced Message Queuing Protocol |
| API | Application Programming Interface |
| ARIMA | Autoregressive Integrated Moving Average |
| AppD | Application Descriptor |
| CA | Certificate Authority |
| CECC | Cloud Edge Computing Continuum |
| CECCM | Cloud Edge Computing Continuum Manager |
| CI/CD | Continuous Integration (CI) and Continuous Delivery (CD) |
| CPU | Central Processing Unit |
| CRUD | Create, Read, Update, and Delete |
| DCAT-AP | DCAT Application profile for data portals in Europe |
| DDoS | Distributed Denial of Service |
| DQN | Deep Q-Networks |
| ETSI | European Telecommunications Standards Institute |
| GUI | Graphical User Interface |
| GXFS | Gaia-X Federation Services |
| HTTP | Hypertext Transfer Protocol |
| IaaS | Infrastructure as a Service |
| IDS | International Data Spaces |
| IDSA | International Data Spaces Association |
| IoT | Internet of Things |
| JSON | JavaScript Object Notation |
| LCM | Life Cycle Management |

| | |
|------|----------------------------|
| LSTM | Long short-term memory |
| ML | Machine Learning |
| MQTT | MQ Telemetry Transport |
| PaaS | Platform as a Service |
| QoS | Quality of Service |
| RBAC | Role-Based Access Control |
| SLA | Service Level Agreement |
| OWL | Web Ontology Language |
| WAF | Web Application Firewall |
| YAML | YAML Ain't Markup Language |

Executive Summary

The purpose of this deliverable is to provide a concise, yet comprehensive overview of the innovative mechanisms designed to enhance application lifecycle management in the Cloud-Edge Continuum (chapter 2). It investigates advanced methodologies and descriptor files (chapter 3-4) to enhance the Lifecycle Management (LCM) of applications within the Cloud-Edge Continuum (CEC). It specifically addresses the development and integration of Service Level Agreement (SLA) and intent models, application profile models, stateful service migration strategies, and AI-driven mechanisms for application profiling and lifecycle management.

It aims to provide details about the feasibility, potential impacts, and implementation strategies of the proposed enhancements. Ultimately, this report serves as a foundational document to guide future development initiatives, support policy-making, and foster strategic planning (chapter 5) in the context of advanced cloud-edge infrastructures. It specifically addresses the development and integration of SLA and intent models, application profile models, stateful service migration strategies, and AI-driven mechanisms for application profiling and lifecycle management.

The analysis began with a review of existing LCM practices within the CEC, identifying key challenges related to application scalability, resource allocation, and operational agility. Then explored potential solutions, focusing on the implementation of application profile models to optimize resource use and application transparency (through the detailed description of the internal structure and the contained microservice architecture). Furthermore, the developed strategies for stateful service migration to ensure continuity and integrity of applications during dynamic relocations are described. Also introduced and detailed the design of AI-based mechanisms for application profiling and lifecycle management, aiming to automate and optimize the LCM processes through intelligent decision-making.

The main conclusions drawn from this report underline the critical importance of adopting a holistic and AI-enhanced approach to application LCM in the CEC. The proposed profiles and intent models are essential for maintaining alignment between service delivery and infrastructure expectations (IaaS Provider). Application profile models emerge as crucial tools for achieving operational efficiency, while stateful service migration is vital for ensuring application resilience. AI-driven profiling and LCM mechanisms are shown to significantly contribute to the proactive management of applications, enhancing adaptability and responsiveness to changing conditions.

1 Introduction

This section provides an overview of the work reported in this deliverable, including the main purpose and the related key objectives. It also describes the link with the other project activities and related deliverables. The targeted outcomes are summarized next and mapped to the GA activities and the deliverable sections. The final subsection provides the structure of the deliverable.

1.1 Overview – Purpose and objectives

In this deliverable, the consortium's main objective is to navigate through the various components related to the user plane within the Cloud-Edge Continuum Configuration Management (CECCM) system. This includes a detailed examination of the components that interact directly with application developers. These components are the **User Interfaces** which (Deliverable 2.1, in Table 2) allows application developers to interact with the CECCM system. This set of interfaces offers the capability to develop, deploy, and manage the entire application lifecycle. User Interfaces serves as the primary gateway for developers, providing an intuitive and efficient means to control and oversee application management processes (through structured YAML and JSON files). The second is the **Application Profiles, Ontology Modeling Tools, and Application Descriptor Models** where these tools are essential for defining the structure, requirements, and intents of applications. They provide a comprehensive framework for creating detailed application descriptors that capture all necessary information for deployment and management. And finally, the **Service and Data Catalogue**: These catalogs offer a centralized repository of services and data sources available within the CECCM system. They enable developers to explore and integrate various resources, ensuring that applications are built using the best possible components and data sources.

Having these functionalities we introduce the **AI-based Lifecycle Management System (LCM)** to encompass both deployment and runtime management, which is the final objective of this deliverable. This involves using advanced AI algorithms to optimize the initial placement of microservices based on detailed application and resource profiles. These profiles include insights into traffic patterns, data intensity, and resource needs, as well as the status of available CECC resources such as computing power, network capacity, and energy consumption. The LCM ensures efficient resource utilization and adherence to Service Level Agreements (SLAs), continuously monitoring application performance and making necessary adjustments like scaling resources or migrating services to maintain optimal operation.

The aim of this deliverable is to perform an in-depth analysis of these components, providing insights and recommendations for the optimal allocation of applications and their constituent microservices. By focusing on specific characteristics, this analysis will guide the best practices for deploying and managing applications efficiently within the CECC environment. This deliverable will outline the foundational principles, targeted functionalities, and advanced mechanisms necessary to enhance the system's overall capability, ensuring that applications are not only well-defined and deployed but also intelligently managed throughout their lifecycle.

1.2 Link with other project activities

This deliverable primarily reflects the design and development activities of Tasks 3.1, 3.2, 3.4 and certain aspects of Task 3.3 related to the service migration (presenting necessary parameters for the profiles).

Initial feedback has been received from the architecture definition task in WP2 regarding the overall design and flow of information, as detailed in Deliverable D2.1 (Sections 4.8, 5.2-5.4) in M08. Further insights into the technologies supporting the targeted architecture were presented in Deliverable D2.3 (Sections 5.2 and 4.3.2) in M12.

During the first implementation phase, the main concepts in relation to the profiles have been refined and adapted to meet the AI-based Lifecycle Management System (LCM)'s overall requirements and implementation features. The advancements and adaptations presented in this deliverable will also be incorporated into the updated final architecture in Deliverable D2.2, due in M24.

1.3 Mapping AC³ Outputs

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

Table 1 Adherence to AC³ GA Deliverable & Tasks Descriptions

| AC ³ GA Component Title | AC ³ GA Component Outline | Respective Document Chapter(s) | Justification |
|--|---|--------------------------------|---|
| DELIVERABLE | | | |
| D3.1 Report on the application LCM in the CEC– Initial | | | |
| TASKS | | | |
| T3.1 SLA and intent-based models for microservice-based applications deployment | SLA model definition; Definition of microservice-based applications for automatic deployment; Translation of performance requirements/intents into corresponding policies/strategies; Extension of the application composition model to include information on data management | Section 3 | Presented the components for the user place as well as the Application & Resource Management of the AC3 Framework architecture. |
| T3.2 Application profile models | Definition and construction of the application profiles to handle the LCM of the application; Collection and monitoring of profiling information; | Section 4 | Have a unified model that cater LCM functionalities. |

| | | | |
|---|---|-------------------------|--|
| | ML algorithm to predict the application profile using the monitored data. | | |
| T3.3 Service migration of stateful microservices | Algorithms to overcome the challenges related to the migration of microservices; Achieve a trade-off between application SLA and infrastructure resources; | Section 5 | Present techniques that will assist LCM in the management place. |
| T3.1, T3.2, T3.3 with WP5 perspective | (Extensions related to overall LCM functionality and module interfacing) | Section 2 and Section 5 | Present solutions that can be integrated to the LCM |

1.4 Deliverable Overview and Report Structure

In this section, a description of the Deliverable's Structure is provided, outlining the respective Chapters and their content:

- Section 2 presents the objectives and briefly outlines the components.
- Section 3 provides a detailed analysis of the architecture and the relation between each component. Moreover, the application descriptor model is introduced, providing details on the parameters that it uses. (An example can be found in the Appendix I)
- Section 4 presents the AI-based application profiling mechanism (An example can be found in Appendix II)
- Section 5 provides the AI-based Lifecycle Management System (LCM) and the algorithms that are necessary for the functionalities that they expose.
- Finally, Section 6 concludes the gaps that need to be addressed by the project in order to implement the AC³ CECCM Framework

2 The LCM process in the overall architecture

The Lifecycle Management (LCM) system is a central component of the Cloud-Edge Continuum Configuration Management (CECCM) framework. It manages, predicts, acts, and updates according to the intents of the deploying applications. This chapter explores the process of creating and deploying applications within the CECC environment architecture (Deliverable D2.1), highlighting how the AI-based Lifecycle Management System (LCM) ensures seamless deployment and intelligent updates. Key areas of development include redefining the Service Catalog (through the Ontology and Semantic Aware Reasoner), the introduction of the AI-based LCM for deployment and runtime management, and the creation of AI-based application profiles. These initiatives collectively enhance the CECCM system's capability to manage applications efficiently, ensuring they are well-defined, optimally deployed, and intelligently managed throughout their lifecycle.

2.1 Main concept and targeted functionalities

This deliverable focuses on **three key functionalities** within the Cloud-Edge Continuum Configuration Management (CECCM) system: the application descriptor with all its requirements and the constraints of how the application can be deployed, deployment strategies and service allocation via the AI-based Lifecycle Management (LCM), and intelligent application redeployment.

Firstly, the application descriptor **encapsulates all the requirements of an application and specifies what needs to be deployed** through a human-readable medium. This comprehensive document includes critical details about the application's deployment, such as necessary resources, configurations, and deployment parameters. By reflecting the application's intents, the application descriptor ensures accurate and efficient deployment within the CECCM system, providing a structured framework for seamless deployment.

Secondly, the AI-based LCM offers **advanced deployment strategies and service allocation methods**. These strategies enable the deployment of applications into the infrastructure as a service (IaaS) of the AC³ federation. The AI-driven LCM evaluates various factors, including resource availability, network conditions, and performance metrics, to determine the optimal deployment method for each application. This ensures that applications are deployed according to their specific requirements while optimizing resource utilization within the AC³ federation.

Lastly, the AC³ system includes a critical functionality that extends beyond initial deployment: **intelligent application redeployment**. Leveraging resource profiles and application profiles, the AI-based LCM provides ongoing lifecycle management, which includes intuitive and updated functionalities for smart allocation. Continuous AI analysis allows the system to make informed decisions about reallocating resources, scaling applications, and optimizing performance. This dynamic and proactive approach ensures that applications remain efficient, responsive, and aligned with evolving operational demands, thereby enhancing the overall efficacy of the CECCM system.

2.2 Mapping to the AC³ architecture

The overall AC³ architecture is depicted in Figure 1. This is an update of the initial architecture described in D2.1. Further details on each element of the architecture will be presented in D2.2. The framework guides the application onboarding, deployment and runtime management process over the interfaced infrastructures.

The focus of the work described in the current deliverable relates to the processes of **a)** application composition and onboarding, **b)** runtime application configuration with an emphasis on service migration (which is one of the core functions of the AI-based life-cycle management module), and **c)** AI-based application profile creation and

usage. These topics are highlighted respectively in Figure 1. The three topics also define the three key development areas that must be addressed and collectively provide the application composition, onboarding, deployment, and finally, the smart management of applications during runtime.

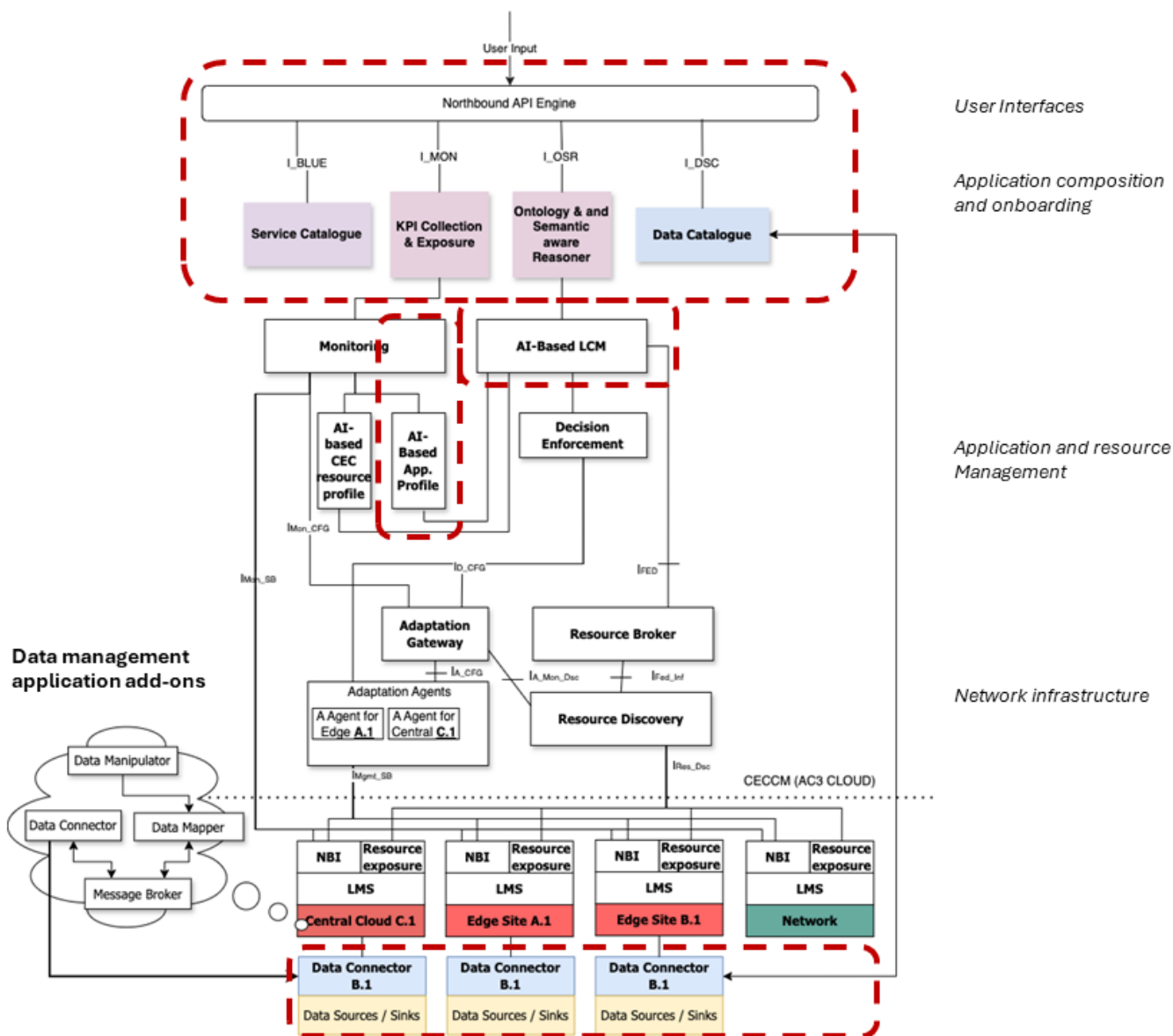


Figure 1 User and Management Plane Architecture (WP3 Components)

Firstly, the redefinition of the **Service Catalog** is crucial. This catalog acts as a comprehensive inventory of available services and resources that applications can utilize, including data sources, microservices, and other critical components. By redefining this part, as detailed in 3.2.4.1, the catalog will provide developers with a clear, organized, and accessible repository, streamlining the deployment process. This redefinition is essential

for enabling developers to efficiently integrate various elements required for their applications, ensuring that all necessary services are readily accessible and well-documented.

Secondly, expanding the functionalities of the **AI-based LCM** to include both deployment and runtime management is essential. The LCM uses sophisticated AI algorithms to determine the optimal initial placement of microservices based on detailed application profiles and CECC resource profiles. These profiles offer comprehensive insights into the application's traffic patterns, data intensity, and resource requirements, as well as the status of available CECC resources such as computing power, network capacity, and energy consumption. This ensures efficient resource utilization and adherence to Service Level Agreements (SLAs). Additionally, the LCM continuously monitors application performance, making necessary adjustments such as scaling resources or migrating services to maintain optimal operation. This proactive management capability ensures that applications remain robust, scalable, and efficient in response to dynamic operational demands.

Lastly, the creation of an (AI-based) **Application Profile** is a significant part of the advancements that the LCM is going to recommend. This profile goes beyond traditional runtime monitoring by incorporating predictive analytics, allowing the system to anticipate future needs and proactively adjust resources. This reduces the reliance on reactive runtime monitoring alone and introduces a new layer of intelligence to the CECCM system. The application profile not only tracks current performance metrics but also forecasts potential issues and opportunities for optimization (through the LCM). By leveraging these predictive insights, the system can make preemptive adjustments, such as scaling up resources before a predicted spike in demand or optimizing network routes to prevent congestion. This enhancement ensures that applications consistently meet user expectations and business requirements.

These three components ensure that applications are not only well-defined and optimally deployed but also intelligently managed throughout their lifecycle. This comprehensive approach provides a robust framework for application management in the dynamic CECC environment, enabling developers to build, deploy, and maintain high-performance applications that can adapt to evolving operational challenges and opportunities. Through these advancements, the CECCM system ensures efficient, scalable, and resilient application management, leveraging AI-driven insights to maintain optimal performance and resource utilization.

3 Application descriptor composition mechanism

The Application Descriptor composition mechanism serves as an opening element for effective application lifecycle management, especially in microservices-based applications. Application profiles, which form the core of these descriptors, provide detailed descriptions of an application's features, requirements, and specifications. Acting as blueprints for software development, these profiles interpret both functional and technical requirements, offering a comprehensive understanding of the application's purpose and capabilities. This guidance is central for the development, deployment, and monitoring processes, ensuring that applications are managed efficiently throughout their lifecycle.

The complexity of CECC applications necessitates robust application profiles due to their distributed nature. These applications often span multiple nodes, each potentially located in different geographical areas and operating on diverse hardware platforms. Therefore, a well-constructed application descriptor must encompass various critical elements: the application name and description, its consumers, the microservices that constitute it, data sources, load and traffic types, and the deployment context. By addressing these elements, the Application Descriptor Composition Mechanism ensures that every aspect of the application is meticulously documented and managed, facilitating seamless deployment and operation within the CECC environment. This chapter analyses the mechanisms behind creating these detailed application descriptors, highlighting their importance in the broader context of CECC application management.

3.1 Overview of the implementation plan

The implementation plan for the Application Descriptor focuses on creating and deploying applications within the Cloud-Edge Continuum Configuration Management (CECCM) system. Central to this process is the specification of application components and requirements, including Service Level Agreements (SLAs). The CECCM facilitates this through the User Interfaces (consuming the Northbound API), which aids in specifying the data needs of the application via the data and service catalogues (Figure 2). The onboarding process begins with translating application intents into corresponding SLAs, forming the foundation for the Application Descriptor. This descriptor is crucial as it encapsulates all necessary details to execute and deploy the application, such as resource requirements and types. Additionally, developers can specify desired policies for scaling and aspects of application management, ensuring the application is well-defined and ready for deployment.

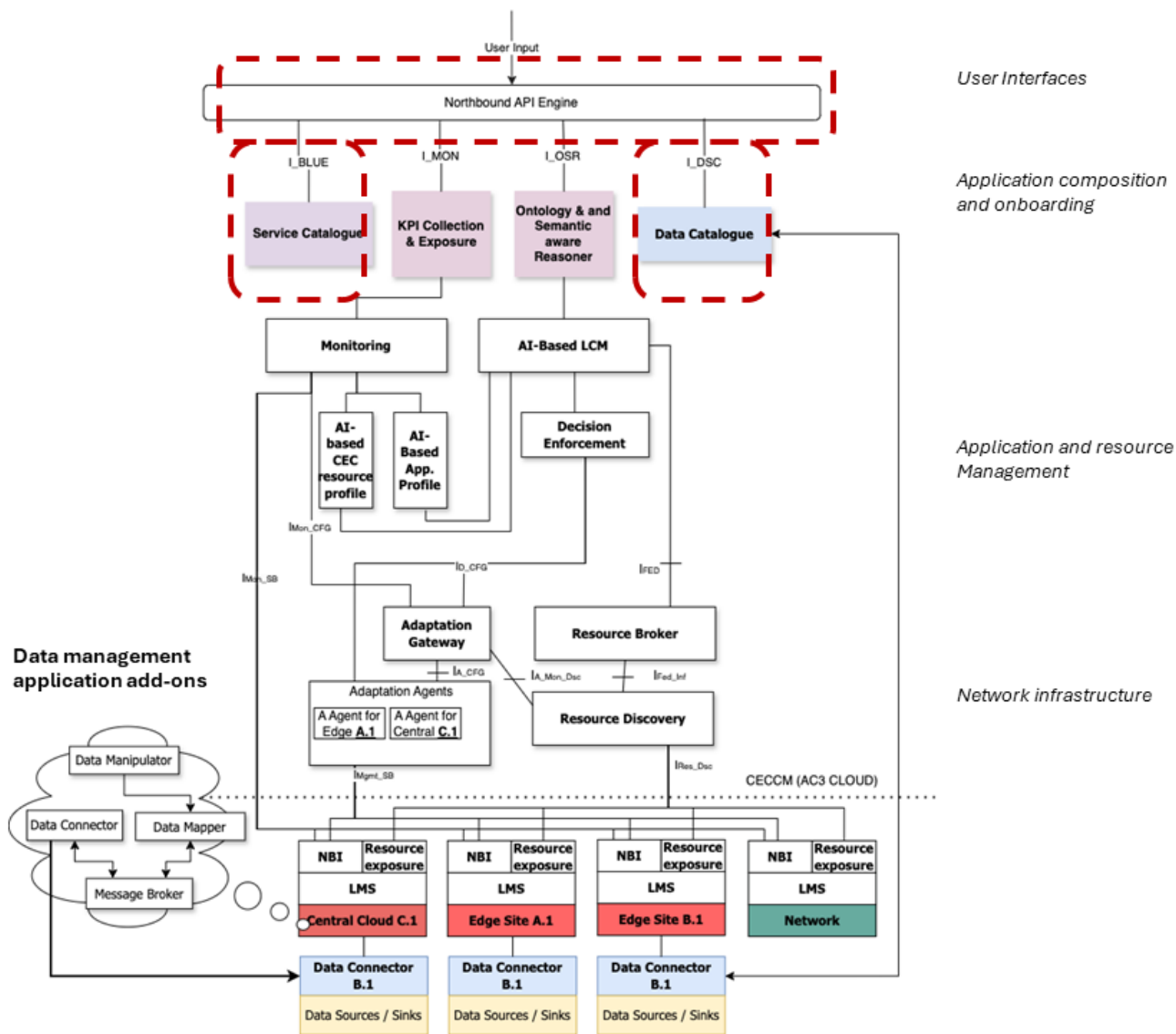


Figure 2 Overall architecture design

In addition to defining the application components, the Application Descriptor plays an important role in the ongoing management of applications. It serves as a single, comprehensive document generated by the Application Gateway, sufficient to guide the deployment of any application. The descriptor includes information about the amount and type of resources required, ensuring that the application is provisioned correctly from the outset. While developers can set initial management policies, the ultimate management and scaling policies are refined and executed by the AI-based Lifecycle Management (LCM) mechanisms. These AI-driven systems take over post-onboarding, optimizing resource allocation and performance throughout the application's lifecycle, ensuring that SLAs are met, and resources are utilized efficiently. This integrated approach guarantees that applications are not only deployed optimally but are also managed effectively in the dynamic CECC environment.

3.2 Initial developments

3.2.1 Application descriptor model

In the user plane and in detail in the application deployment and management of microservices, the Application Descriptor serves as a critical component within the Lifecycle Management (LCM) framework. This document format is specifically designed to encapsulate the essential aspects of an application's architecture and operational requirements, guiding the automated processes managed by an Ontology and Semantic-aware Reasoner (OSR).

This component utilizes the gathered data to present an innovative approach to addressing policies within a Cloud-Edge environment. By unifying the handling of application descriptors, requirements, policies, and interconnected components—essential in microservice compositions—it offers a comprehensive understanding of the extracted knowledge. The proposed approach enables real-time extraction of insights regarding the most critical policies for the application and infrastructure, thereby enhancing the effectiveness of policy management in dynamic environments.

At its core, the Application Descriptor provides a comprehensive specification of the application, including the network and compute resources needed for optimal performance. It meticulously outlines the structure and functionality of each microservice that constitutes the application, detailing the specific configurations and the interdependencies among them. This allows for a systematic and coherent assembly of the microservices, ensuring they function seamlessly as a unified entity.

The Application Descriptor serves as an input to the Ontology Semantic Reasoner (OSR). Upon submission by developers, the descriptor is transmitted to the OSR, where it is interpreted and translated into a machine-processable format. The OSR leverages semantic web technologies like ontologies and reasoners to represent the application's requirements, components, and policies defined in the descriptor. By understanding the semantic relationships and ontological structures within the application data, the OSR can reason about the policies and management rules described in the descriptor. It uses various reasoning techniques like deduction, induction, abduction, and analogy to derive new knowledge, detect dependencies between microservices, and validate the policies against the runtime conditions of the application and infrastructure. The OSR does not directly orchestrate the deployment or manage the application lifecycle. Instead, it acts as an intelligent reasoning engine that interprets and adapts the human-defined policies and intents captured in the Application Descriptor. It translates these into machine-processable ontology instances and semantic rules that can be consumed by other components responsible for deployment and lifecycle management. Thus, the Application Descriptor is not merely a static document but a pivotal element that interacts with sophisticated semantic and ontological reasoning mechanisms to facilitate efficient and intelligent application lifecycle management in modern distributed environments.

The Application Descriptor is modelled as a structured document that outlines the composition of an application, detailing its various components, behaviors, requirements, and the environment it operates within. It serves as a blueprint that guides the deployment, scaling, and management of applications across cloud-edge infrastructures, ensuring that all elements are configured and interact seamlessly as intended.

The Application Descriptor model encompasses several fields, each contributing to a comprehensive understanding of the application's design and operational context.

1. **Application:** This field serves as the identifier and descriptor of the application itself. It includes Name, Version, and Description as subfields. The Name uniquely identifies the application within the system,

the Version helps in tracking iterations and updates, and the Description offers a concise summary of the application's functionality and its intended use. This section forms the basic introduction to the application, giving stakeholders a quick overview of its purpose.

2. **Application Consumers:** The "Application Consumers" section of the application descriptor comprehensively outlines the users or systems that will interact with the application, ensuring that the application's design and functionalities cater to the specific needs and preferences of its intended users. This segment includes detailed information on the types of users, the interfaces they will use, and the functionalities specific to them, enabling tailored user experiences and secure access management.
 - a. **Profile Type:** Defines the category of users or systems interacting with the application, such as individuals, enterprises, or automated systems, helping to tailor the application to the needs of its specific user base.
 - b. **Interfaces Experiences:** Provides detailed configurations concerning the user interfaces that will be employed:
 - i. **Interface Type:** Specifies the kind of interface, such as web, mobile, or desktop, designed to ensure compatibility and usability across different platforms.
 - ii. **Endpoint:** Indicates the specific endpoint through which the interface is accessed, defining how users connect to the application.
 - iii. **Technologies:** Lists the technologies used to build and support the interface, ensuring that the application leverages the most effective tools and frameworks for user interaction.
 - c. **Consumer Specific Functionality:** Describes functionalities that are tailored to specific types of users or systems, ensuring a personalized and secure user experience:
 - i. **Authentication & Authorization:** Outlines the methods and security measures employed to verify user identities and control access, ensuring that only authorized users can perform certain operations.
 - ii. **Permissions:** Details the specific permissions assigned to different user roles, clarifying what actions each type of user can perform within the application.
 - iii. **Security Measures:** Additional security protocols implemented to safeguard user interactions and protect sensitive data.
 - iv. **Usage Patterns:** Describes typical usage scenarios and patterns, which help in optimizing the application design to better suit how consumers will use the application.
3. **Microservices:** The "Microservices" section outlines the core architectural components of the application, describing each microservice that plays a part in the system's functionality. This section provides essential information about the identity, function, and operational specifics of each microservice, enabling a deep understanding of the application's structure and the relationships between its components. Detailed specifications such as deployment characteristics, inter-service dependencies, resource needs, and scalability options ensure that each microservice can be effectively integrated and managed within the overall architecture.
 - a. **Name, ID, Service Type, Purpose:**
 - i. **Name:** The unique identifier for each microservice.
 - ii. **ID:** A system-generated unique identifier that helps in tracking and managing the microservice.
 - iii. **Service Type:** Categorizes the microservice based on its functionality (e.g., database, API gateway).
 - iv. **Purpose:** Describes the primary function or role of the microservice within the application.
 - b. **Image, Version:**

- i. Image: Specifies the container image used for deploying the microservice.
 - ii. Version: Indicates the version of the microservice, aiding in version control and updates.
 - c. API Endpoints: Details the interfaces through which the microservice communicates with other components or services:
 - i. API ENDPOINT URL: The URL where the microservice can be accessed.
 - ii. API ENDPOINT PORT: The network port used for communications.
 - iii. Purpose: Describes what the endpoint is used for.
 - iv. Method: Specifies the HTTP method (GET, POST, etc.) supported by the endpoint.
 - d. Dependencies: Lists other microservices or resources that this microservice depends on to function properly, highlighting the interconnected nature of the application components.
 - e. Resource Requirements: Details the resources needed by the microservice to operate efficiently:
 - i. Minimum Resource Requirements: The minimum resources necessary for basic operation.
 - ii. Maximum Resource Requirements: The maximum resource allocation to handle peak loads.
 - f. Operational Support Systems: Systems like monitoring and logging that support operational integrity.
 - g. Scalability & Load Balancing: Strategies and policies for scaling and load management.
 - h. Resource Management: How resources are allocated and managed.
 - i. Performance Metrics: Metrics such as Response Time Variability, Concurrent User Load, and Data Throughput that help in measuring and optimizing the performance of the microservice.
 - j. Incident Response & Compliance: Details the standards and practices in place for handling failures and ensuring compliance with service-level agreements (SLAs).
 - k. Configuration & Environment:
 - i. Environment Variables: Key-value pairs that configure the microservice's runtime environment.
 - ii. Image Pull Policies: Rules that dictate how images are pulled from repositories, important for deployment and updates.
- 4. Data Sources: The "Data Sources" section delineates the various sources of data that the application utilizes, detailing their types, characteristics, and the security protocols governing their use. This section helps understanding how data is integrated and managed within the application, ensuring that data flow is both efficient and secure. It includes comprehensive information on each data source's identity, operational details, and how it interfaces with the application, for maintaining data integrity and optimizing data-driven functionalities.
 - a. Source Name, Category, Details:
 - i. Source Name: The name assigned to the data source which identifies it within the system.
 - ii. Category: Classifies the data source based on its type or nature (e.g., hot, cold).
 - iii. Details: Provides a brief description or other relevant information about the data source that can be crucial for its utilization or management.
 - b. Broker Address, Geographical Location:
 - i. Broker Address: The network address of the data broker or service that manages access to the data source.
 - ii. Geographical Location: Specifies the physical or virtual location of the data source, including details such as:
 - 1. Country: The country where the data is stored or from which it is accessed.

2. Coordinates: Latitude and longitude details if applicable, providing precise geographical positioning.
 - c. Data Models: Outlines the structure and schema of the data being managed within the source. This includes:
 - i. Model Name: The name of the data model.
 - ii. Attributes: A list of attributes or fields within the data model, detailing their names and characteristics.
 - d. Data Access & Security: Describes the mechanisms and protocols in place to access the data, alongside the security measures implemented to protect the data:
 - i. Access Method: The method by which data is accessed (e.g., API calls, direct database access).
 - ii. Security Measures: Specific security protocols or standards applied to protect the data from unauthorized access or breaches.
 - iii. Content Filtering and Security Policies: Policies in place for filtering data and ensuring it meets security criteria.
 - iv. Authentication Methods: The procedures used to verify the identity of users or systems requesting access to the data.
 - v. Authorization Methods: The controls that determine what data can be accessed by which users or systems.
5. Network Traffic and Load: The "Network Traffic and Load" section outlines the key aspects of how network traffic is managed and optimized within the application, ensuring efficient and reliable service delivery. This part of the application descriptor focuses on describing the types of network traffic, how microservices interact over the network, and the strategies in place for maintaining network performance and handling load effectively. These details design a robust network that can sustain high availability and performance under varying load conditions.
- a. Traffic Type, Time-Based Routing:
 - i. Traffic Type: Defines the kinds of network traffic expected, such as data queries, user requests, or service-to-service communications, which helps in planning network capacity and capabilities.
 - ii. Time-Based Routing: Specifies routing rules that change based on the time of day or other temporal conditions, optimizing network efficiency and resource utilization during peak and off-peak hours.
 - b. Microservice Interconnection: Details the network links and protocols used for interconnecting microservices within the application, ensuring seamless data flow and integration between different services:
 - i. Connection: Describes each connection by source, destination, protocol used, and ports involved.
 - ii. Connection SLA: Service Level Agreements associated with each connection specifying expected performance metrics like latency, availability, error rate, and bandwidth.
 - c. Network SLA: Overall service level agreements that define the network performance expectations across the application, including parameters such as latency, availability, error rate, response time variability, and bandwidth, which are crucial for meeting the performance guarantees promised to users.
 - d. Load Balancing Strategy: Describes the approach and techniques used for distributing incoming network traffic across multiple servers or instances to avoid overloading any single resource, thereby enhancing responsiveness and uptime:
 - i. Strategy: The specific strategy employed (e.g., round-robin, least connections).

- ii. Techniques: Additional techniques like geographic DNS routing that further enhance the load balancing effectiveness.
 - e. Rate Limiting & Throttling: Policies and actions designed to regulate the amount of traffic a user or a service is allowed to send or receive within a given timeframe, which helps in preventing service abuse and managing resource consumption:
 - i. Rate Limiting: The maximum rate at which requests are allowed from a particular user or service.
 - ii. Throttling: Temporarily reducing the bandwidth or rate of requests a user or service can make.
 - iii. Exceptions: Conditions under which these policies may not apply.
 - iv. Rate Limiting Policies: Specific rules defining how rate limiting is implemented.
 - v. Actions: Actions taken when a user or service exceeds their rate limit.
- 6. Deployment Context: The "Deployment Context" section provides a detailed overview of the environments and strategies under which the application will be deployed. This includes information about the infrastructure scalability, containerization practices, cloud platforms, network configurations, and geographical considerations affecting the deployment. By outlining these specifics, this section ensures that the application is deployed efficiently and effectively, tailored to both the technological and geographical nuances of the deployment landscape.
 - a. Scalability, Containerization Details:
 - i. Scalability: Describes the capacity of the application to handle increases in load, including the methods and metrics used to scale resources up or down as required.
 - ii. Containerization Details: Specifies the containerization technology used (e.g., Docker, Kubernetes), which encapsulates the application in a way that is independent of the host system, enhancing portability and scalability.
 - b. Cloud Platforms, Networking & Traffic Management:
 - i. Cloud Platforms: Lists the cloud services and platforms where the application will be hosted, such as AWS, Google Cloud, or Azure, specifying services and configurations used.
 - ii. Networking & Traffic Management: Details the network setup and traffic management strategies that will be implemented to ensure robust and efficient data transmission and accessibility.
 - c. Service Mesh Integration, Policy & Compliance Management:
 - i. Service Mesh Integration: Outlines the integration with service meshes that manage service-to-service communications, providing capabilities like load balancing, service discovery, and security.
 - ii. Policy & Compliance Management: Describes the policies and compliance requirements that the application must adhere to, ensuring that deployment practices meet all regulatory and company standards.
 - d. Location: Details about the specific geographic regions where the application components will be deployed, which can include:
 - i. Microservice Name: Identifies which microservice the location setting pertains to.
 - ii. Microservice Affinity: Specifies any affinity or anti-affinity rules for deploying microservices.
 - iii. Geographical Affinity: Regional considerations that impact deployment, such as data sovereignty laws.
 - iv. Region ID: The identifier for the specific region within the cloud or data center.

- v. Instantiation Order: The sequence in which services or components should be instantiated during deployment.
- e. Maintenance & Update Cycles:
 - i. CI/CD Pipeline: Describes the continuous integration and continuous deployment practices that facilitate frequent and reliable code changes and application updates.
 - ii. Data Management: Specifies how data associated with application deployment and operation is maintained, including backup strategies and data rotation policies.
- 7. Security and Access Control Policies: The "Security and Access Control Policies" section outlines the frameworks and measures implemented to protect the application against unauthorized access and various security threats. It details the strategies and technologies used to verify user identities, manage access permissions, secure data, and respond to potential security incidents. This section ensures the integrity and security of the application, safeguarding user data and system operations against potential vulnerabilities and attacks.
 - a. Authentication, Authorization:
 - i. Authentication: Mechanisms that verify the identity of users or systems attempting to access the application, using methods such as passwords, tokens, or biometric data.
 - ii. Authorization: Processes that determine what resources a user or system can access and the actions they can perform, often implemented following authentication.
 - b. Data Encryption: Protocols and practices used to secure data, ensuring it remains protected both during transmission (in-transit) and while stored (at-rest). This includes using technologies such as SSL/TLS for in-transit data and encryption algorithms like AES for data at rest.
 - c. Firewall Policies, Web Application Firewall, DDoS Protection:
 - i. Firewall Policies: Rules and settings configured in a firewall to control network traffic and block unauthorized access.
 - ii. Web Application Firewall (WAF): A specialized type of firewall that monitors, filters, and blocks harmful HTTP traffic to and from a web application.
 - iii. DDoS Protection: Measures and technologies used to protect the application from Distributed Denial of Service (DDoS) attacks, which attempt to overwhelm resources and disrupt service.
 - d. Role-Based Access Control: Access control policies that assign permissions based on user roles within the organization, ensuring users have access only to the resources necessary for their roles.
 - e. Incident Response Plan: A predefined set of procedures and instructions for the organization to follow in response to a security breach or other incidents, ensuring quick and effective mitigation to minimize damage.
 - f. Monitoring and Auditing:
 - i. Traffic Metrics: Systems used to monitor the volume and type of traffic flowing through the network, essential for detecting anomalies.
 - ii. Audit Trail: Records of events and changes within the application, providing a log that can be reviewed and analyzed to detect unauthorized or suspicious activity.
 - iii. Security Event Logging and Audit: Tools and practices for logging security events, which are crucial for forensic analysis and compliance with regulatory requirements.
 - iv. Alerting: Mechanisms in place to notify system administrators of security threats or incidents in real-time.
 - v. Alerting Methods: The methods used for alerting, which can include emails, SMS messages, or automated phone calls.

An example of an application Descriptor can be found in the in Annex 1

3.2.2 Application descriptor composer

The Application Descriptor Composer in the AC³ architecture, augmented by the Ontology Semantic Reasoner, serves as an interface between end-user requests and the technical requirements for application management. The OSR takes input from the ontology defining different actors in the system and translates human-readable policies into a machine-processable format, enabling the generation of the application descriptor based on end-user GUI requests.

The application descriptor process involves:

- **User Input Interpretation:** The composer interprets the inputs provided by the end-user, such as desired application features, performance requirements, and deployment preferences.
- **Descriptor Generation:** Generating the application descriptor based on interpreted user inputs, detailing components, behaviors, and requirements in a format understandable by the LCM module.
- **LCM Feeding:** The generated application descriptor is then fed into the Lifecycle Management (LCM) module, which uses this information to manage the application's deployment, scaling, and other lifecycle operations.

The Application Descriptor Composer, with the support of the OSR, translates high-level user requirements into a structured format for automated application management, ensuring that the LCM can make informed decisions based on user specifications. This process leads to optimized application deployment and operation, playing a vital role in enabling seamless and efficient interaction between end-users and technical management systems within the AC³ architecture.

3.2.2.1 Graphical User Interface (GUI) for Application Descriptor Composer

The GUI of the Application Descriptor Composer serves as the primary entry point for users, providing a user-friendly interface that seamlessly integrates with other system elements. It is designed to enhance user interaction and efficiency in managing application descriptors. The dashboard offers a range of functionalities including creating, reading, updating, and deleting (CRUD) application descriptor models, visualizing these models, and interacting with them through a multi-step form or file uploads.

CRUD Operations on Application Descriptor Model:

- **Create:** Users can create new application descriptors either by filling out a multi-step form or by uploading a complete JSON file. This flexibility allows users to start from scratch or modify an existing template.
- **Read:** The dashboard allows for the visualization of existing application descriptors, providing detailed views that help users understand and analyze their configurations.
- **Update:** Users can update existing descriptors through the GUI, making changes directly through an interactive form or by re-uploading modified JSON files.
- **Delete:** Descriptors can be removed from the system through a simple user interface option, ensuring that users can manage their storage and keep their dashboard organized.

Form Creation and Upload Options:

- Application Descriptor Creation Using Dynamic Forms

The 'createprofile' component of the Application Descriptor Composer's GUI offers a user-friendly interface to create new application descriptors. This dynamic form is structured into multiple steps organized into distinct sections, each corresponding to a key component of the application descriptor model. Each step is designed to collect specific data in a structured way, enhancing the user's ability to understand and input the necessary information accurately. This structured input process creates a robust and complete application descriptor that can be effectively managed by the system.

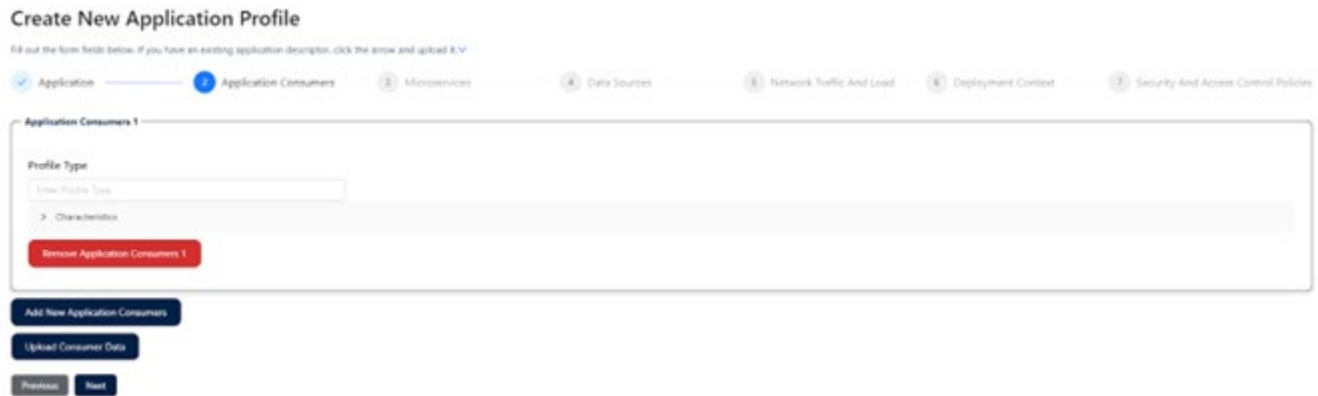


Figure 3 Visual of the multi-step form interface

Figure 3 highlights the different sections included:

- **Application:** Users begin by entering fundamental details such as the application's name, version, and a brief description.
- **Application Consumers:** This step gathers information about the end users or systems that will interact with the application, detailing their roles and requirements.
- **Microservices:** Users specify the microservices that comprise the application, including their functionalities, dependencies, and resource needs.
- **Data Sources:** Details regarding the data sources the application will utilize are defined here.
- **Network Traffic and Load:** Configurations related to network traffic and expected load are set in this step.
- **Deployment Context:** Users provide specifics about the deployment environment, including cloud platforms and deployment regions.
- **Security and Access Control Policies:** Security settings and access control measures are established to protect the application.

This form is powered by a JSON template. It dynamically generates its fields and structure from the predefined JSON template, which ensures that any updates to the descriptor model are automatically reflected in the form without manual adjustments. This template-driven approach allows the form to remain flexible and scalable, accommodating changes in the underlying data model with minimal code modifications.

- **Uploading a Complete Application Descriptor**

For users preferring a quicker setup or those with pre-defined configurations, the dashboard also supports uploading a complete JSON file. This method is ideal for advanced users who have a ready application descriptor

model or wish to use a standard template provided by the system. Users can download the template, fill it with the necessary data, and upload it back to the system. (Figure 3)

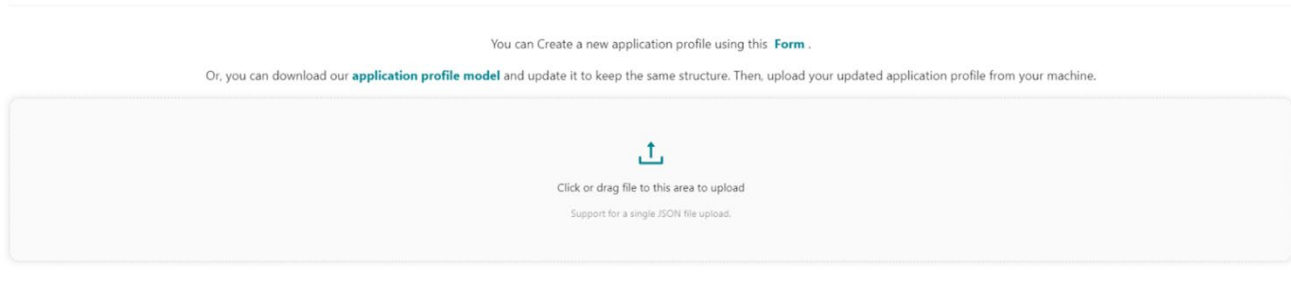


Figure 4 Upload interface

- **Hybrid Approach: Combining Form and Upload Methods**

The dashboard accommodates a hybrid approach where users can fill out certain sections of the application descriptor using the form and upload JSON files for other parts. This method is particularly useful when users want to manually configure key components like microservices or deployment settings while quickly uploading standardized configurations for other sections like security policies or network settings. (Figure 4)

This hybrid method caters to the need for detailed customization in critical areas while offering efficiency where possible, making the process both flexible and user-friendly.

3.2.3 Visualisation and Interaction

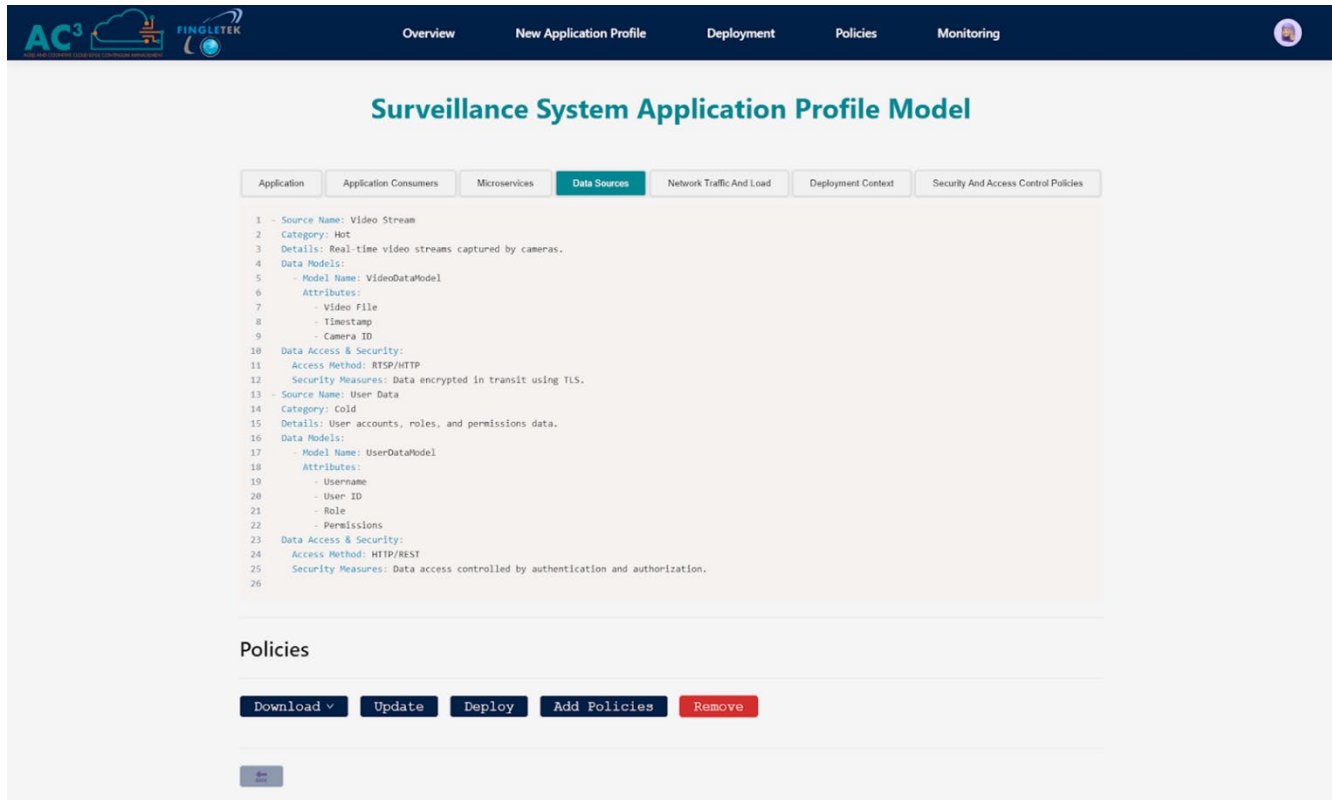


Figure 5 User Interface of Model Profile

Each application descriptor is visualized in a user-friendly manner, with options to expand and collapse sections for better navigation. The system provides an advanced method to present application descriptors in a clear and structured YAML format, which is particularly useful for subsequent processing by the Lifecycle Management (LCM) system.

Application descriptors are dynamically rendered in a user-friendly YAML format. YAML was chosen for its readability and ease of use, particularly in configurations where clarity and human readability are paramount. The script processes the JSON structure of the application descriptors and converts it into YAML, ensuring that all configurations are displayed in an organized and easily navigable format. Presenting the descriptor in YAML format not only aids in readability but also prepares the configuration for seamless integration with the LCM. This format ensures that the transition from configuration to deployment is smooth and error-free.

The interface (Figure 5) allows users to expand and collapse sections of the application descriptor. This feature is implemented using interactive components that toggle the visibility of detailed content, making the interface cleaner and more manageable. Users can focus on specific sections without being overwhelmed by the full complexity of the descriptor.

A significant feature implemented in the script is the ability to download the YAML representation of the application descriptor.

- **Offline Editing:** Users can download and edit the YAML file offline in their preferred environment or tools, providing flexibility in how they manage their configurations.

- **Backup and Version Control:** Downloading the descriptor allows users to maintain local copies for backup purposes or for maintaining version history, which is essential for rollback scenarios and historical comparisons.

These enhancements in the visualization and interaction of the application descriptor models not only improve the user experience but also enhance the operational efficiency of managing application deployments. The ability to visualize, interact with, and directly utilize the application descriptors in a structured format like YAML ensures that users can manage complex configurations with greater

3.2.4 Service catalogue and data catalogues

3.2.4.1 Service Catalogue

The Service Catalogue as it is presented in D2.1 serves as a centralized repository for all applications available in the AC³ environment. It provides developers with a comprehensive listing of application blueprints, detailing their capabilities, requirements, and dependencies. Acting as a central hub, the Service Catalogue assists AC³ developers in managing and maintaining the ecosystem's blueprints, offering a unified view of all software, services/microservices, libraries, machine learning models, and datasets.

The primary role of the Service Catalogue is to showcase the available blueprints, ensuring they are easily discoverable and accessible for all users. By tracking ownership and metadata associated with each blueprint, the catalogue enables developers to understand the full spectrum of available services within the AC³ environment. This comprehensive understanding is facilitated by the catalogue's ability to offer operations on application blueprints. These operations allow developers to register new applications, retrieve information about existing ones, update metadata and computing details, and off-board outdated blueprints.

Additionally, the Service Catalogue supports advanced search functionalities, allowing users to filter applications based on specific criteria and retrieve relevant blueprints. This search capability is underpinned by metadata files stored alongside the corresponding code, containing essential information about the blueprints, such as purpose, usage, and dependencies. By harvesting and visualizing this metadata, the catalogue presents it in a user-friendly manner, making it straightforward for developers to find and utilize the resources they need.

The Service Catalogue also facilitates the dynamic addition of new blueprints by integrating with additional code repositories. This ensures that the catalogue remains up to date with the latest developments and available resources, enhancing its utility and relevance. By aligning with the Data Catalogue, which provides a detailed inventory of data assets, the Service Catalogue ensures seamless integration of services with the necessary data inputs, streamlining the configuration and deployment processes. The AC³ Data Catalogue is implemented using the Piveau Hub as presented in D3.3. It was selected amongst other available options based on its characteristics, support from the community, and integration and customization capabilities. The Piveau Hub data catalogue is a data management solution designed to assist organizations in effectively collecting, managing, and sharing data, particularly in the realm of open data. It serves public administrations and entities by providing a central repository where data from multiple sources can be integrated and managed. This platform supports open data principles, enabling organizations to publish their data in accessible formats under open licenses, which promotes transparency and reuse. It also emphasizes data quality and adherence to standards, ensuring the reliability of the data it hosts. It is scalable, which allows it to handle increasing volumes of data and adapt to growing user needs, and it is customizable to meet specific organizational requirements. Additionally, the platform is built to ensure interoperability, facilitating data exchange and integration with other systems, which is crucial for collaborations. In detail management and search interfaces are supported which facilitate most of

the needs of the component. Firstly, the Piveau Hub Repo Service API¹ simplifies the management of datasets, adhering to the DCAT-AP standard² for structured data descriptions. It integrates seamlessly with Virtuoso, an efficient database server and Triplestore³, making dataset management more accessible. The DCAT-AP standard ensures consistent, comprehensive data descriptions, enhancing data sharing and discovery. The API simplifies the process of making datasets accessible to a wider audience. This promotes transparency, knowledge sharing, and collaboration. This introduction lays the groundwork for exploring the hub-repo Open API. Detailed information on authentication, API endpoints, error handling, and practical examples can be found in the subsequent sections to gain a comprehensive understanding of the API's functionality. The second interface is the Hub-Search which is a powerful search solution designed to function as a Metadata Search Service for the Open Data Portal. This API facilitates the discovery and retrieval of metadata, making it easier for users to find datasets and resources. Hub-Search seamlessly integrates with Elastic Search to provide robust and precise search capabilities. Hub-Search API not only simplifies searching but also fosters transparency and knowledge sharing. It empowers users to easily discover valuable data resources, improving data accessibility and utilization. This introduction sets the stage for exploring the Hub-Search Open API. Detailed information on authentication, API endpoints, error handling, and practical examples can be found in the subsequent sections.

3.2.5 User Interfaces

User Interfaces integrate (through the Application Gateway, D2.1-Section 4.3) all functionalities that AC³ aims to expose (leveraging the Northbound API Engine), utilizing the generic schemas defined in application descriptors and profilers to provide a user-friendly platform for managing, serving, and consuming AC³ services. Through these meticulously crafted blueprints, information is thoroughly documented, ensuring that underlying services comprehend the parameters they use and why they use them. This documentation also offers end users a clearer understanding of the decisions made.

Explainability is particularly significant within the AC³ ecosystem, as many components leverage AI techniques. The HMI plays a dual role: it not only presents services in a user-friendly manner but also simplifies and clarifies each parameter and selection through well-structured documentation. This transparency is essential for logging, tracking KPIs, and resolving any disagreements that may arise. By providing detailed explanations for each decision and parameter, the HMI ensures that both developers and end users can easily navigate and utilize the full range of AC³ services effectively.

3.3 Planned implementations and extensions for final phase

The Application Descriptor Mechanism, in combination with all the other services that AC³ exposes, serves as an essential foundation for effective application lifecycle management within the Cloud-Edge Continuum Configuration Management (CECCM) system. By providing detailed descriptions of an application's features, requirements, and specifications, application profiles act as comprehensive blueprints guiding software development. These profiles interpret both functional and technical requirements, ensuring that applications are efficiently developed, deployed, and monitored throughout their lifecycle. Given the distributed nature of CECC applications, robust application profiles are used for managing applications spanning multiple nodes, potentially across different geographic locations and hardware platforms. By addressing critical elements such as application names, descriptions, consumers, microservices, data sources, load and traffic types, and deployment context,

¹ <https://doc.piveau.de/hub/>

² <https://semiceu.github.io/DCAT-AP/releases/3.0.0/>

³ <https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/>

the Application Descriptor Mechanism ensures meticulous documentation and management, facilitating seamless deployment and operation within the CECC environment.

On top of that, the Lifecycle Management (LCM) process within the CECCM system exemplifies a sophisticated and seamless flow of services designed to optimize application deployment and management. Starting from the Northbound API Engine, which serves as the primary gateway for developers, the process moves through the Service Catalog, providing a centralized repository of blueprints and data sources. The Ontology & Semantic-aware Reasoner translates human-readable policies into machine-processable formats, identifying dependencies and generating the Application Descriptor. Finally, the AI-based Lifecycle Management system dynamically manages the application, ensuring optimal resource allocation and adaptability throughout its lifecycle. This integrated approach guarantees robust, efficient, and intelligent application lifecycle management, catering to the evolving needs of modern digital environments.

Looking ahead, several expansions can further enhance the capabilities and functionalities of the CECCM system. Integrating advanced AI algorithms can improve predictive analytics and decision-making processes, enabling even more efficient resource allocation and application performance optimization (will be presented in the next chapter). Expanding the scope of application profiles to include more granular details and additional parameters can provide deeper insights into application behavior and requirements, enhancing the precision of deployment and management strategies. Developing more sophisticated monitoring and logging tools can provide real-time insights into application performance, facilitating quicker detection and resolution of issues and more effective tracking of KPIs (in cooperation with WP4). Enhancing interoperability with other systems and platforms can enable seamless integration and communication across different environments, fostering a more cohesive and versatile application management ecosystem. Continual improvements to the User Interfaces can further simplify and enhance user interactions, making it easier for developers to manage, serve, and consume AC³ services effectively (in cooperation with WP5). Integrating these advancements with the monitoring mechanisms presented in WP4 and testing the environment with use cases can identify ways to highlight the characteristics of each blueprint to the use case developers, thereby ensuring that the CECCM system remains at the forefront of modern digital environments.

4 AI-based application profiling mechanism

This section introduces a general vision of the need for and importance of an AI-based application profiling mechanism in the development of modern-day systems. Today's rapidly evolving technological landscape has necessitated the fundamental adoption and deployment of AI-based application profiling mechanisms in different industrial systems. As a result, different industrial sectors are increasingly leveraging the power of artificial intelligence to enhance their operations and services, which can be easily traced to the continuous need for efficient application profiling mechanisms that have become essential. Generally, an application profiling mechanism serves as a critical component in understanding the behaviour, performance and resource requirement and utilisation of AI-based applications. Therefore, in subsequent sections, we provide a practical overview of our implementation approach and plan and a preliminary description of the initial development of identified constituent components of the profiling mechanism, offering a glimpse into our methodology and strategies. Additionally, we outline the planned implementation processes, features, and extensions for the final phase, laying the groundwork for a comprehensive and robust solution.

4.1 Overview of the implementation plan

The implementation plan for the AI-based application profiling mechanism encompasses a structured approach to effectively capture, analyse and optimise the performance of AI-based applications. This plan is designed to provide a comprehensive framework for assessing most importantly changes in application behaviour and requirements that may be necessitated due to various factors such as resource utilisation, network traffic patterns, user activities, application dependencies, etc.

The process begins with Application Descriptor Analysis, where the application's initial descriptors are scrutinised to extract crucial data such as resource demands and SLA details. This foundational analysis informs subsequent stages by providing a baseline of expected application behaviour. Interconnection with Monitoring and Information Collection follows, linking the system with real-time monitoring tools. These tools continually collect data on application performance metrics like CPU usage and response times, ensuring the application profiles are updated dynamically. The collected data is then channelled into the Data Processing stage. Here, data is cleaned, transformed, and made ready for deeper analysis, preparing it for the predictive modelling phase. After data processing, the Machine Learning Model Development phase kicks in. In this critical phase, machine learning models are built and trained to predict future application behaviours based on the synthesised insights from historical and real-time data.

These models help the Profiling and Prediction Engine, uses the trained models to generate dynamic application profiles that inform resource allocation and application scaling in real time. Finally, these profiles are fed to the Lifecycle Management (LCM) to ensure optimal application performance and resource utilisation. The Profiling and Prediction Engine acts as a bridge between the machine learning models and the LCM component, translating the model outputs into actionable insights and recommendations. It continuously updates the application profiles based on the latest predictions, ensuring that the LCM component has access to the most up-to-date and accurate information about the application's behaviour and resource needs.

The key components necessary to enable this AI-based application profiling mechanism are:

- Application Descriptor Analysis
- Interconnection with Monitoring and Information Collection
- Data Processing
- Machine Learning Model Development
- Profiling and Prediction Engine

Throughout this process, advanced AI techniques and solutions are employed to enhance the accuracy and efficiency of the profiles. The use of AI not only streamlines the flow of information across components but also ensures that the application profiles are continuously refined and adapted based on evolving application dynamics and operational demands.

Each component is intricately linked, ensuring seamless data flow and integration from the initial analysis to the application of predictive insights, ultimately guiding resource allocation and lifecycle management strategies. This structured approach ensures that the AI-based application profiling mechanism remains robust, adaptive, and aligned with the operational needs of cloud edge applications.

4.2 Initial developments

In this section, we delve into the foundational stages of the AI-based application profiling mechanism within the AC³ project. This section outlines the steps taken to establish a dynamic profiling system that can adapt to the evolving landscape of cloud-edge computing. The focus is on analysing application descriptors, integrating monitoring tools, processing data, and developing machine learning models for building a robust framework that can predict application behaviour and optimise resource allocation in real-time. (Figure 6)

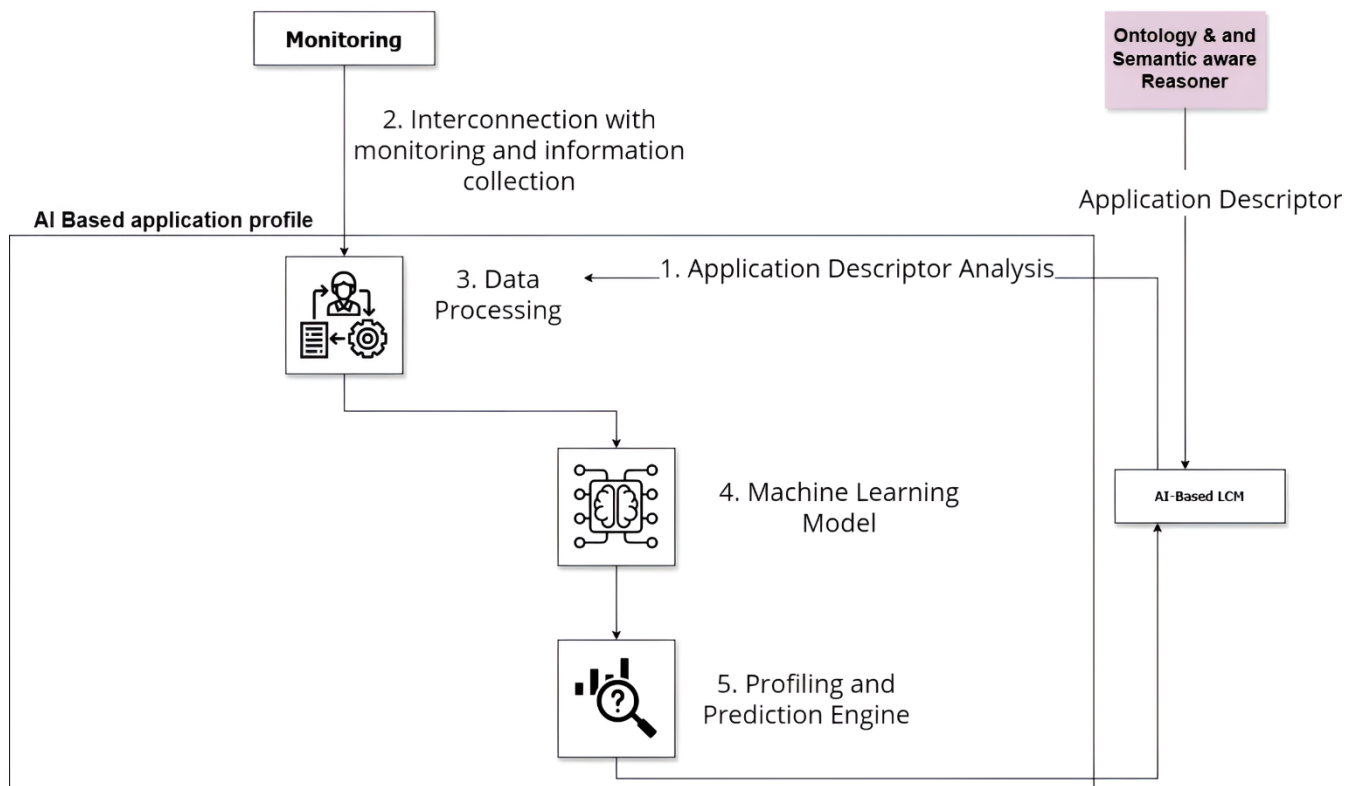


Figure 6 AI Based Application Profile Solution Architecture

4.2.1 The Application Descriptor Analysis

The Application Descriptor Analysis involves a detailed examination of the application descriptor provided by the developer. This descriptor is a blueprint that outlines key characteristics of the application, such as resource requirements, and **Service Level Agreement (SLA)** constraints. These elements are critical for understanding how the application is expected to behave and what resources it will need to function optimally.

In this step, the system parses the descriptor to extract and interpret this information. Parsing involves breaking down the descriptor's content into manageable pieces that can be analysed and understood by the system. This process is essential for translating the developer's specifications into a format that can be used for model training and profile creation.

The extracted information forms the foundation for the initial training of machine learning models. These models are designed to predict the application's behaviour based on the specified traffic patterns, resource needs, and SLA requirements. The analysis of the application descriptor ensures that the models are tailored to the specific characteristics of the application, enabling more accurate predictions and effective resource allocation.

The Application Descriptor Analysis step sets the stage for the AI-based application profiling mechanism. By thoroughly understanding the application's requirements and behaviour patterns, the system can create dynamic profiles that adapt to changing conditions, ensuring optimal performance and resource utilisation.

4.2.2 Interconnection with Monitoring Module and Information Collection

The AI-based application profiling mechanism heavily relies on an efficient interconnection with the monitoring module developed in WP4 to dynamically update and refine application profiles. This step involves setting up a sophisticated system to continuously collect, process, and analyse real-time performance metrics from the WP4 monitoring component.

- Data Collection and Transmission

Data collection can be facilitated using specialised monitoring tools like Prometheus⁴ or Nagios⁵, which may be integrated directly with system hardware and software as part of the monitoring module. These tools can be configured to track and record metrics like CPU usage, memory consumption, and network bandwidth at specified intervals, ensuring a steady flow of real-time data from the monitoring infrastructure.

To maintain data integrity and security during transmission from the monitoring module to the data processing module of the profiling mechanism, encrypted protocols such as TLS/SSL may be employed. Additionally, message queuing technologies like MQTT⁶ or AMQP⁷ can be utilised to ensure reliable and asynchronous data delivery, mitigating the risks of data loss or duplication even in unstable network conditions.

- Data Handling and Stream Processing

Handling the continuous data streams from the monitoring module efficiently can be achieved through technologies like Apache Kafka⁸, which supports high-throughput and fault-tolerant stream processing. This framework may allow for the buffering, processing, and real-time analysis of large volumes of data collected by

⁴ Prometheus: <https://prometheus.io/>

⁵ Nagios: <https://www.nagios.org/>

⁶ MQTT: <https://mqtt.org/>

⁷ AMQP: <https://www.amqp.org/>

⁸ Apache Kafka: <https://kafka.apache.org/>

the monitoring infrastructure, thereby enabling the profiling system to respond dynamically to new data inputs without overwhelming system resources.

4.2.3 Data Processing

The data processing step within the AI-based application profiling mechanism transforms raw data into actionable insights that can dynamically influence application behaviour. This process involves a series of systematic and methodical actions to ensure data is not only collected but also meticulously prepared for machine learning analysis.

- Data Collection and Preparation

Data is initially gathered from the monitoring component, which capture various application metrics in real time, such as CPU usage, memory consumption, and network bandwidth. The first stage of processing involves cleaning this data by removing or correcting any anomalies and outliers, such as illogical CPU usage figures or negative response times.

- Feature Engineering and Data Transformation

Following cleaning, the raw data undergoes feature engineering, where it is transformed into more meaningful features. This includes converting timestamps into cyclical features to capture patterns based on time, like peak usage hours, which reflect the periodic nature of network traffic or system usage. Another step is normalisation or standardisation where numerical values are scaled to a uniform range to prevent any single metric from disproportionately influencing the model's predictions.

- Data Management and Storage

For efficient management and retrieval, data is stored in scalable solutions like time-series databases or big data platforms capable of handling high volumes and velocities. Strategic data indexing is implemented to enhance access speeds during model training. The data is also segmented logically, perhaps by time or operational metrics, facilitating efficient access and analysis.

- Regular Maintenance and Data Quality Assurance

Continuous data quality assurance is maintained through regular checks and balances, including data backups and robust disaster recovery protocols to safeguard data integrity. This ensures that the machine learning models receive high-quality data to generate reliable predictions.

- Operationalizing Data for Machine Learning

The prepared data is then partitioned into datasets for training, validating, and testing the machine learning models. This setup ensures that the models are well-trained to predict future application behaviour accurately and efficiently, based on both historical patterns and real-time events, effectively feeding into the Lifecycle Management (LCM) for optimal decision-making. This comprehensive data processing framework enhances the predictive capabilities of the AI-based application profiling mechanism, ensuring applications perform optimally in a cloud-edge computing environment.

4.2.4 Machine Learning (ML) Model Development

The Machine Learning (ML) component within the AI-based application profile plays a role in dynamically predicting application behaviour and optimising resource allocation. The choice of ML algorithms, such as LSTM, GRU, ARIMA, Prophet, Bi-LSTM, CNN-LSTM, and ARIMA-BP Neural Network, is dictated by the specific nature of

the application data, often comprising time-series elements. These models can be selected for their distinct capabilities in handling various types of data patterns and computational complexities:

- LSTM (Long Short-Term Memory) [1]: Utilised for its proficiency in capturing long-term dependencies within time-series data, making it ideal for metrics like CPU usage, memory consumption, and network bandwidth. It employs backpropagation through time (BPTT) for training on historical data, optimising for future predictions on resource utilisation.
- GRU (Gated Recurrent Unit) [2]: Chosen for its computational efficiency and effectiveness in scenarios requiring rapid updates, such as request rates and active sessions. GRUs simplify the recurrent neural network architecture by using fewer parameters, facilitating quicker training and deployment.
- ARIMA (Autoregressive Integrated Moving Average) [3]: Applies to metrics exhibiting trends and seasonal patterns like response times and error rates. ARIMA excels in linear trend modelling and is calibrated through methods like maximum likelihood estimation and cross-validation.
- Prophet: Deployed for its robustness in handling daily seasonality and missing data, making it suitable for forecasting data throughput and the number of active sessions. Prophet models adapt well to irregular time series data, which is common in real-world application metrics.
- Bi-LSTM (Bidirectional Long Short-Term Memory) [4]: Enhances the LSTM framework by processing data in both forward and backward directions to improve the contextuality of predictions. This model is particularly effective for predicting latency and database read/write operations.
- CNN-LSTM (Convolutional Neural Network-Long Short-Term Memory) [5]: Combines CNN's spatial pattern recognition with LSTM's temporal data handling capabilities. This model is tailored for complex metrics involving spatial and temporal dynamics, such as network bandwidth usage.
- ARIMA-BP Neural Network [6]: Integrates ARIMA's linear predictive qualities with the non-linear pattern recognition of backpropagation neural networks. This hybrid model is particularly adept at addressing error rates and metrics that exhibit both linear and non-linear characteristics.

In the development of machine learning models for AI-based application profiling, the implementation phase involves several key steps that ensure the models are trained effectively on historical application data. This data spans various metrics like CPU usage, network bandwidth, etc., each reflecting an aspect of application performance.

1. Data Preparation: Initially, raw data is pre-processed to clean, normalise, and structure it into a usable format. This may include handling missing values, standardising range scales, and encoding categorical variables.
2. Model Building: Models are constructed using frameworks like TensorFlow or PyTorch. Depending on the metric, different architectures are chosen, such as LSTM for sequential data or CNN for spatial data integration.
3. Compilation: The model is compiled with an appropriate optimizer like Adam, which helps in efficiently minimising the loss function during training. This step also defines the loss metric, such as mean squared error (MSE) for regression tasks.
4. Training: The model is trained on the prepared datasets using techniques like backpropagation. For time-series data, methods like backpropagation through time (BPTT) are used to adjust weights and biases across sequences effectively.
5. Validation and Testing: Using separate validation and testing datasets, the model's performance is evaluated to ensure it generalises well on unseen data. Techniques like cross-validation might be employed to robustly assess model performance.
6. Deployment: Once trained and validated, the model is deployed within the AC³ system to start predicting real-time data. This continuous integration allows for dynamic updating of application profiles based on incoming data streams.

7. **Monitoring and Updating:** after deployment, the model's performance is continually monitored. Retraining cycles are scheduled to update the model with new data, ensuring the predictions remain accurate and relevant to changing application conditions.

This structured approach ensures that the ML models are not only tailored to specific application needs but are also capable of evolving with the application's operational dynamics. The ML component enables the system to make informed decisions about resource allocation, scaling, and optimization. By accurately predicting application behaviour, the system can proactively address potential issues, optimise performance, and ensure efficient use of resources. This results in improved application reliability, user experience, and cost-effectiveness.

4.2.5 Profiling and Prediction Engine: Dynamic Application Profiling Mechanism

The Profiling and Prediction Engine utilizes machine learning models to generate and update dynamic application profiles. This engine forecasts application behavior, allowing for more effective resource management and optimization strategies.

The core of the Profiling and Prediction Engine involves the integration and operationalization of machine learning models that have been trained on historical and real-time data to build the Dynamic Application Profiles. These models analyze patterns and predict future states of application utilization and performance.

- **Model Integration:** Initially, machine learning models are integrated into the engine. These models have been trained to interpret various metrics composing the application profile model.
- **Profile Generation:** Using the outputs from these models, the engine constructs dynamic profiles that describe likely future scenarios of application behavior. These profiles include predictions about resource needs, potential bottlenecks, and optimal scaling opportunities.
- **Feeding Lifecycle Management (LCM):** The dynamic profiles are not static; they are continuously updated with new data and predictions and are systematically fed into Lifecycle Management (LCM). This process ensures that the LCM can make informed decisions based on the latest predictions, adjusting resources in real-time to meet the anticipated demands.

4.2.6 Profiling Metrics and Parameters

The AI-based Application Profile component in the AC³ Functional Architecture relies on a comprehensive set of metrics and parameters to dynamically predict application behaviour and optimise resource allocation. The selection of these metrics is guided by their relevance to understanding the application's performance, resource utilisation, and user interaction. These metrics include:

- ⊘ **Traffic Patterns:** understanding data flow and volume between services for managing network resources and planning scaling strategies.
- ⊘ **Resource Utilisation:** monitoring CPU, memory, storage, and network resources helps predict future needs and detect potential bottlenecks.
- ⊘ **Response Times and Latency:** these metrics ensure the application meets performance requirements and help identify areas for optimization.
- ⊘ **Error Rates:** tracking the frequency of errors aids in identifying stability issues and underlying problems with microservices or infrastructure.
- ⊘ **User Activity:** analysing user interactions provides insights into load patterns and guides feature development and user experience improvements.
- ⊘ **Data Access Patterns:** understanding how data is accessed and used drives optimizations in data storage and processing, impacting overall performance.

- ∄ Application Dependencies: mapping dependencies between services is essential for predicting potential bottlenecks and maintaining stability during updates or scaling.

Table 2 Parameters

| Category | Description | Metric | Description |
|---------------------------------|---|---|---|
| Traffic Patterns | Understanding data flow and volume between services to manage network resources. | <ol style="list-style-type: none"> 1. Request Rate (requests/sec) 2. Data Throughput (Mbps) | <ol style="list-style-type: none"> 1. Number of requests per second 2. Megabits per second transmitted over the network |
| Resource Utilization | Monitoring how much computing resources an application consumes to predict scaling needs. | <ol style="list-style-type: none"> 1. CPU Usage (% or GHz) 2. Memory Usage (MB or GB) | <ol style="list-style-type: none"> 1. Percentage or GHz of CPU consumed 2. Megabytes or Gigabytes of memory consumed |
| Response Times | Ensuring application meets performance requirements by measuring responsiveness. | <ol style="list-style-type: none"> 1. Average Response Time (ms) 2. Latency (ms) | <ol style="list-style-type: none"> 1. Average milliseconds taken to respond 2. Milliseconds for a packet round-trip |
| Error Rates | Tracking frequency of errors to identify stability issues. | <ol style="list-style-type: none"> 1. Error Rate (% or errors/sec) 2. Failed Transactions (count) | <ol style="list-style-type: none"> 1. Percentage or errors per second 2. Number of transactions that failed |
| User Activity | Analysing user interactions to optimize user experience and predict load patterns. | <ol style="list-style-type: none"> 1. Active Sessions (count) 2. User Interaction Patterns (patterns) | <ol style="list-style-type: none"> 1. Number of active user sessions 2. Patterns in user interactions with the application |
| Data Access Patterns | Understanding data storage and retrieval to optimize database performance. | <ol style="list-style-type: none"> 1. Read/Write Operations (operations/sec) 2. Data Retrieval Times (ms) | <ol style="list-style-type: none"> 1. Number of read/write operations per second 2. Milliseconds taken to retrieve data |
| Application Dependencies | Mapping dependencies between services to predict potential bottlenecks or failure points. | <ol style="list-style-type: none"> 1. Dependency Graph (graph) 2. Service Call Frequency (calls/sec) | <ol style="list-style-type: none"> 1. Graph showing inter-service dependencies 2. Number of calls between services per second |

4.2.7 Application Profile Model

The first version of the application profiling includes these key metrics:

- ∄ **Time window (time - time):** Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns) window for the period that the application is evaluated.
- ∄ **CPU Usage (%):** Predicts the percentage of CPU usage over time for scaling decisions.
- ∄ **Memory Usage (GB):** Forecasts memory requirements in gigabytes over time to prevent bottlenecks.
- ∄ **Network Bandwidth Usage (Mbps):** Anticipates network load in megabits per second over time for optimization.
- ∄ **Response Time (ms):** Predicts the milliseconds taken for the application to respond over time for user experience.
- ∄ **Error Rate (%):** Forecasts the percentage of errors over time for proactive maintenance.
- ∄ **Request Rate (requests/sec):** Anticipates the number of requests per second over time for resource allocation.
- ∄ **Data Throughput (Mbps):** Predicts data transfer rates in megabits per second over time for network planning.
- ∄ **Active Sessions (count):** Forecasts the number of active user sessions over time for capacity planning.
- ∄ **Read/Write Operations (operations/sec):** Anticipates database read/write operations per second over time for performance tuning.
- ∄ **Latency (ms):** Predicts network latency in milliseconds over time for optimising communication.

The metrics were chosen for the first version of application profiling in the AC³ System to provide a well-rounded view of application behaviour. They cover key aspects such as resource utilisation, performance, user interaction, and network activity, enabling comprehensive analysis and informed decision-making for dynamic profiling. The selection aims to ensure that the profiling can effectively support scaling, maintenance, and optimization efforts. Continuous evaluation and updates of these metrics are essential to keep pace with evolving application requirements and behaviour.

4.3 Planned implementations and extensions for final phase

1. **Algorithm Selection and Optimization:**
 - a. Conduct a comprehensive study to identify the most suitable ML algorithms for predicting application behaviour based on the established application profile model.
 - b. Explore a range of algorithms, including but not limited to decision trees, support vector machines, neural networks, and ensemble methods, in addition to the already mentioned LSTM and ARIMA models.
 - c. Evaluate the trade-offs between model accuracy, computational complexity, and explainability to ensure the chosen algorithms align with the project's objectives.
2. **Model Training and Validation:**
 - a. Develop a robust training pipeline to train the selected ML models on historical data collected from the monitoring module.
 - b. Implement cross-validation techniques to assess the models' performance and generalise their ability to predict application behaviour accurately.
 - c. Fine-tune model parameters and hyperparameters to optimise performance.

3. **Real-time Data Integration and Model Updating:**
 - a. Establish a mechanism for integrating real-time monitoring data into the application profiling process, enabling continuous model updating and refinement.
 - b. Implement incremental learning techniques to update the models without the need for complete retraining, ensuring efficient use of computational resources.
4. **Profile Adaptation and Decision-Making:**
 - a. Develop algorithms to dynamically adapt application profiles based on the predictions from the ML models, considering factors such as traffic patterns, resource utilisation, and user activity.
 - b. Integrate the updated profiles with the AI-based Lifecycle Management (LCM) module to inform decision-making processes related to scaling, migration, and resource allocation.
5. **Explainability and Interpretability:**
 - a. Focus on enhancing the explainability of the ML models to provide insights into the reasoning behind predictions and decisions, facilitating transparency and trust.
 - b. Implement techniques such as feature importance analysis, model visualisation, and surrogate models to interpret the models' behaviour.
6. **Scalability and Performance Optimization:**
 - a. Ensure the scalability of the implementation to handle large-scale applications and infrastructure.
 - b. Optimise the performance of the ML algorithms and the overall profiling process to minimise latency and resource consumption.
7. **Testing and Validation:**
 - a. Conduct extensive testing to validate the effectiveness of the planned implementations in real-world scenarios.
 - b. Iterate on the design and implementation based on feedback and results from testing to ensure the final phase meets the project's goals and requirements.

5 AI-based LCM mechanism and functions

As introduced in D2.1, the AI-based Lifecycle Management (LCM) component handles the lifecycle management of application-based microservices, which was initially defined by the application developers and translated to a descriptor by the Ontology and Semantic aware Reasoner (OSR). To recall, the LCM procedures of an application consist of preparation (implemented through the OSR), deployment, run-time, and termination, as shown in Figure 7. Each procedure is composed of different tasks:

- **Deployment:** covers the tasks related to configuration and instantiation of the micro-services on the infrastructure. This step involves (1) the initial placement of the micro-services on top of the CEC infrastructure, (2) software image onboarding, and (3) application instantiation. Step 1 is very critical. It relies on the application descriptor that integrates the SLA requested by the micro-services and requires input from both the AI-based application and CEC resource profiles.
- **Run-time:** This is the core function of the LCM as it is in charge of real-time management of the application instance by monitoring and supervising the application performances and making any corrections if a deviation of SLA is detected. An example of correction (i.e., LCM decisions) can be the scale-up or migration of an application instance. In AC³, the AI-based LCM relies heavily on the AI-based application and CEC resource profiles to derive LCM decisions.
- **Termination:** This step concerns everything related to the termination of the application, that is, termination and off-boarding of the images from the infrastructure.

In this section, we provide initially a high-level overview of the AI-based LCM, focusing on the functionalities that are required to be carried out. Then, we highlight the specific implementations and adopted methodologies. The last section provides the planned extensions that are envisioned in the final implementation round.

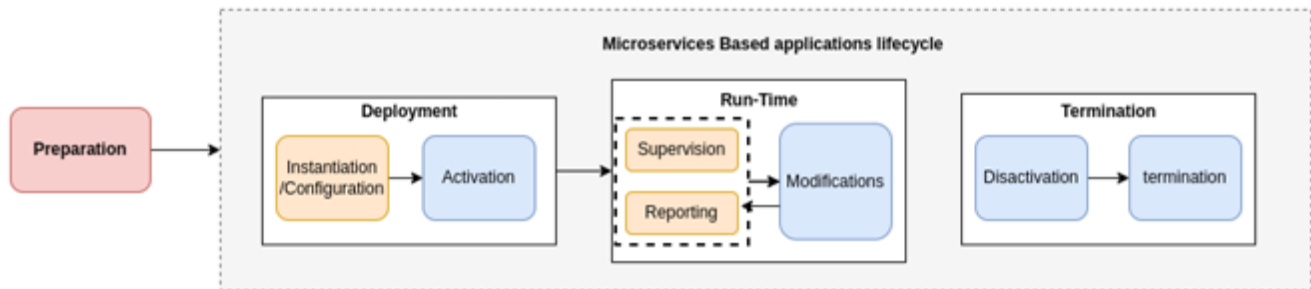


Figure 7 Microservices Based Applications Lifecycle

5.1 Overview of implementation plan

The AI-based LCM module is the core element of the AC³ architecture, responsible for collecting the end user requests, directing the deployment requests towards the infrastructure management entities, and managing the reconfiguration of the running applications through the generation of updated requests following the application profile, the targeted infrastructure resource profile and the application SLAs. The above-mentioned functionalities imply a set of functional features and interfaces that must be implemented to enable the proper communication with the supporting elements and the hosting of the smart decision-making mechanisms and algorithms for enabling the targeted functionalities.

An abstracted version of the LCM module is presented in the highlighted part of Figure 8 together with all key functionalities that should be supported. The same figure shows also the interfacing with the surrounding modules. The LCM functionalities are split into two parts related to the deployment and runtime management.

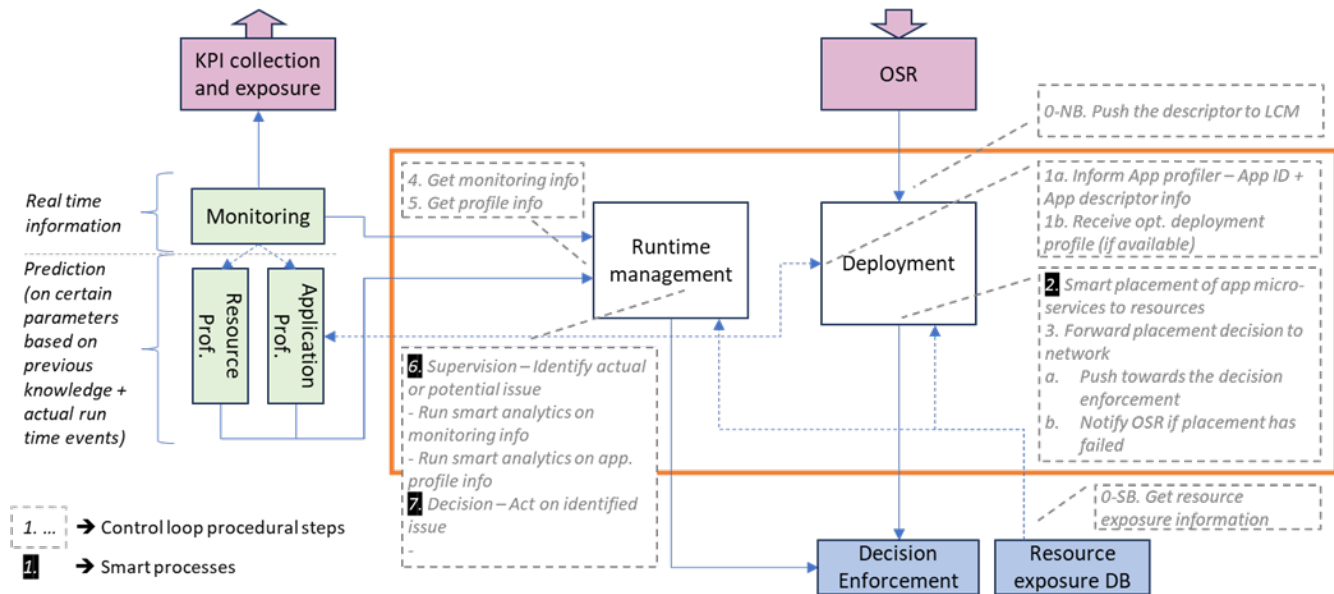


Figure 8 The LCM module and its interaction with the surrounding modules in the AC³ architecture.

According to the overall flow of information a set of specific implementation steps are derived and embedded in Figure 8, pointing to the related modules or interfaces.

Deployment steps:

- **Step 0-NB: Receive the application deployment request:** The application request (application compose API) is pushed to the deployment module. A response for correct or not reception is fed back.
- **Step 0-SB: Informed about infrastructure resources:** Following the resource discovery process (implemented in WP4) the collected and continuously updated information is exposed to the deployment and runtime management modules and made available for the calculation of the application deployment and the selection of potential application reconfiguration during runtime.
- **Step 1a, b: Inform the Application profiler and (if available) receive a deployment profile:** For each application deployment request received, a new request to the application profiler module is sent. This triggers the creation of a new profile, or it is matched to an existing profile. In the latter case the matched profile is returned to the deployment module for proceeding with a suggested deployment
- **Step 2: Extract smart placement decision:** In this process the application deployment function is implemented taking into consideration the resource availability of the targeted infrastructures and the application related SLAs.
- **Step 3: Push placement for deployment:** The extracted placement decision is structured and pushed to the underlay infrastructure for deployment. The same step should implement the expected deployment verification feedback.
- Runtime management steps

- **Step 4: Get monitoring information:** Information is collected from the monitoring module on specific monitoring parameters associated to each of the deployed applications and used to implement immediate decisions, based on SLAs and policies.
- **Step 5: Get application profile information:** The application profile information is collected for each application and used to implement predicted decisions.
- **Step 6: Smart supervision of deployed applications:** Utilizes the collected monitoring information and application profile information to generate potential issues with respect to the application SLA and policies.
- **Step 7: Smart runtime decisions on the deployed applications:** Identifies the optimized reconfiguration decisions to mitigate the identified issues.
- **Step 8: Push decision for application and/or infrastructure reconfiguration:** The decision next pushed towards the related infrastructure interfaces (same as in step 3).

These steps, in turn, define the overall implementation plan of LCM and the integration process that will be adapted within WP5 work.

It is noted that the underlined processes in steps 3, 6 and 7, are those that require the implementation and deployment of the targeted AI-based solutions. The adopted implementation technologies for these steps are further explained in detail in the following sub-section. The rest of the steps are procedural in nature and implemented through standardized solutions and interfaces adopted in AC³ project from previous work. Further details on the overall integration and the interfaces will be provided in WP5 work.

5.2 LCM Functions and initial developments

5.2.1 Smart placement of app micro-services to resources (Placement)

One essential feature that the AI-based LCM should ensure is the initial placement of microservices. The key balance should be found between finding a placement that ensures application SLA while optimizing the energy consumption. Indeed, Microservices placement can be optimized to minimize latency between application microservices. In such cases, selecting resources within the same site or geographical zone can significantly reduce latency. Information about latency and other infrastructure resources can be communicated via the *resource discovery module* or inferred from *resource profiles*. Additionally, minimizing latency between certain microservices and end users can be another initial placement use case. In this scenario, the AI-based LCM can utilize data from the application profile to ensure an initial placement that respects the application's Service Level Agreement (SLA). Various techniques and algorithms, such as combining reinforcement learning techniques with Markov decision processes (MDPs), can be employed for this purpose [7]. Placement can also be optimized to minimize energy consumption or the utilization of computing and networking resources. In this scenario, a range of algorithms can be employed, from brute force methods suitable for small environments to more complex meta-heuristic implementations. These algorithms can be selected based on the specific metric or combination of metrics that need to be minimized.

5.2.2 Runtime supervision

During the runtime phase, the AI-based LCM has two main tasks: supervising application performance and triggering LCM decisions to correct application behavior or optimize CEC resources.

In AC³, supervision is done through the definition of AI-based applications and AI-based CEC resource profiles, which collect monitoring information on the applications and CEC resources. The profiles also use AI/ML to predict the application behaviors and the resource usage to anticipate any potential application SLA violations by leveraging monitoring data and exposed resource metrics as inputs for its prediction algorithms. Time series algorithms such as LSTM, ARIMA, and Meta Prophet⁹ are commonly employed in these scenarios to forecast potential SLA violations and take proactive measures to mitigate them. In this phase, the AI-based LCM also needs to update the information in both the resource and application profiles. This ensures that the LCM has the most accurate and up-to-date information about the resources available and the requirements and behavior of the application. This updating process enables the LCM to make informed decisions and adjustments in real-time based on the current state of the system.

5.2.3 Runtime decision

In this phase, the AI-based LCM triggers LCM actions enforce the underlying systems to prevent, for instance, SLA degradation or optimize CEC resources for energy optimization. One common technique for preventing application SLA degradation is service migration. Services can be categorized as stateless or stateful. With stateless services, the migration process is simpler due to a wider range of destination infrastructure options available to handle the service. However, with stateful services, additional considerations come into play. Firstly, preserving the state of microservices at the optimal moment is crucial to avoid data loss during migration. Secondly, stateful applications may require more resources to manage client requests alongside application data efficiently.

In this deliverable, we will introduce two solutions for micro-service migration, which are run at the AI-based LCM during the run-time procedures. The first algorithm orchestrates service migration for fault tolerance by assessing computing resource utilization within the existing infrastructure. The second algorithm addresses user mobility by strategically re-placing microservices geographically closer to end-users, thus minimizing latency. Presently, these algorithms operate independently within AI-based LCM. However, in the future D3.2, we may entail integrating these approaches to yield a more comprehensive and versatile solution.

5.2.3.1 *Stateful Microservices migration in Multi-Cluster Environments:*

Currently, container orchestration software lacks a robust solution for migrating stateful containers. Instead, these systems typically stop the container running the microservice and restart it elsewhere. While this approach can be useful in certain scenarios, it can lead to data loss in cases where the application's internal state is crucial. To ensure the smooth operation of deployed applications throughout their entire lifecycle, we introduce a proactive solution for stateful container migration within multi-Kubernetes cluster environments. The migration of the microservice in this solution is contingent upon the utilization of resources within the infrastructure where it operates. This helps to trigger microservices migration within the infrastructure, preventing potential performance degradation and ensuring a better user experience, as well as comprehensive application lifecycle management.

The predictor model

As outlined in Section 5 of this deliverable, the application profile model will encompass the resource utilization patterns of the application throughout various stages of its lifecycle. This approach enables us to project future execution requirements based on current and anticipated resource usage within the federated infrastructure.

⁹ <https://facebook.github.io/prophet/>

The central element of this solution revolves around a machine learning time series forecasting model. This model relies on time series monitoring data supplied by other CECCM components, such as the resources discovery module. The monitoring data is fed into the model as vectors, which are then utilized to predict future resource utilization values. Based on these predictions, the system determines whether migrating microservices on this infrastructure is necessary or not.

The model is trained on one of the world's largest public datasets, the GWA-T-13 MATERNA ¹⁰ dataset. This dataset contains performance metrics gathered from 520, 527, and 547 virtual machines (VMs) within a distributed datacenter. Currently, our model utilizes only memory consumption values as training metrics. However, our goal is to enhance this model's capability to predict additional metrics in future versions, such as CPU consumption. The input and output of the model are illustrated in Figure 9.

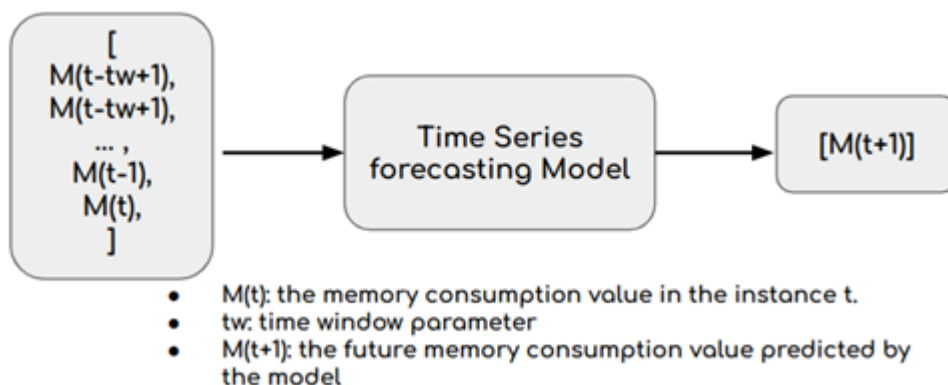


Figure 9 Model's Inputs and outputs

For implementing the model, we opted for LSTM (Long Short-Term Memory networks) due to their well-established reputation for performance and their ability to effectively learn patterns, especially in scenarios with large volumes of data. To facilitate this, we constructed another database consisting of two columns. The first column contains vectors comprising selected values from the original dataset, while the second column corresponds to the subsequent value in the original dataset sequence. The length of these vectors is a crucial parameter in time series modeling known as the time window, which can significantly influence the model's outcomes. The following feature illustrates the transformation of the original database, considering k as the value of the time window parameter and n is the number of the dataset entries (Figure 10).

¹⁰ <http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna>

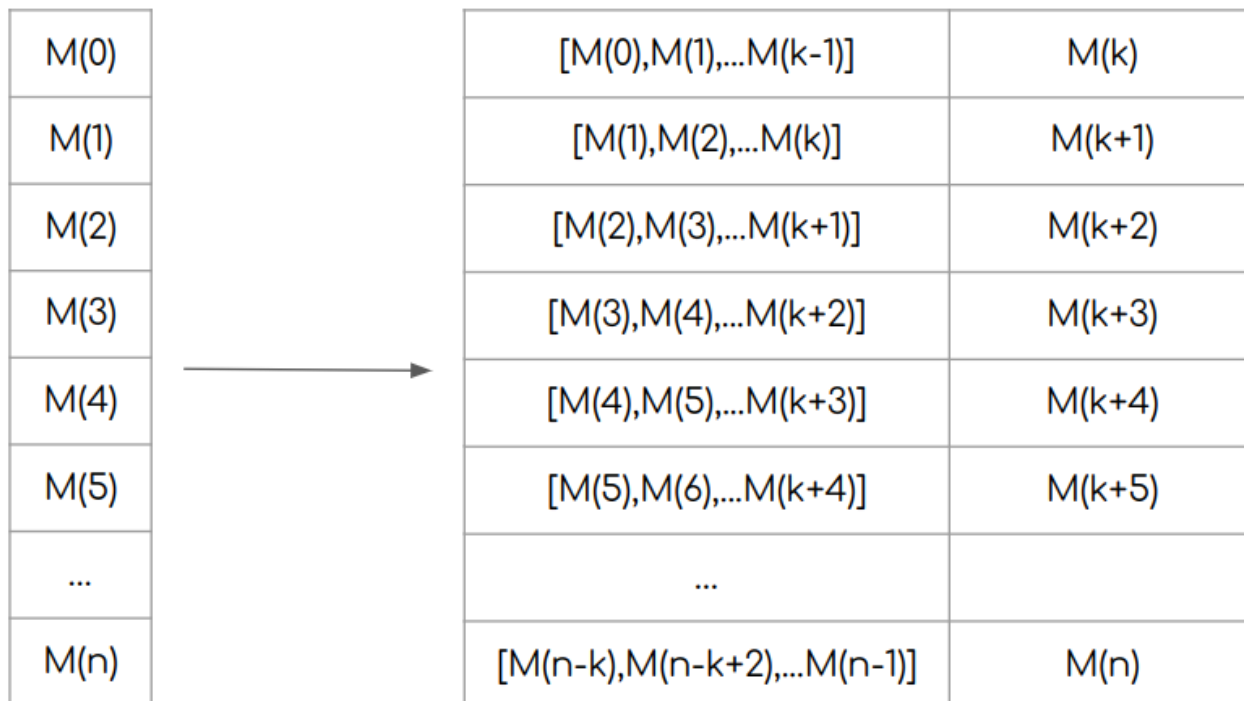


Figure 10 Transforming the initial dataset to training dataset

To train the model, we employed a sequence of layers. Initially, two LSTM layers were incorporated to enable the model to discern various patterns effectively. Subsequently, a set of dense hidden layers was introduced to conduct nonlinear transformations of the data. Finally, a dense layer with a single unit was added as the output layer, aimed at predicting a single value, which is typically the case in RNN models. The model was then compiled using the mean square error loss function and the "Adam" optimizer.

The key distinction among these models lies in the size of the Time Window (TW), representing the length of the input sequence fed to the LSTM model during each training or prediction iteration. We investigated various values (5, 10, 20) for this time window. For instance, when dealing with time-series data and employing a time window of 10-time steps, the LSTM model will analyze the preceding 10-time values to forecast the value at the subsequent time step. The metrics values for the constructed models are outlined in Table 3.

Table 3 Results from Different Model Instances

| TW Value | 5 | 10 | 20 |
|----------|------------------------|------------------------|------------------------|
| MSE | 3.821×10^{-4} | 3.703×10^{-4} | 3.642×10^{-4} |
| Accuracy | 83.19% | 85.87% | 86.32% |

As mentioned earlier, the output of this module will be a prediction of the future value. Based on this prediction, the algorithm will decide whether to migrate the service to another infrastructure by comparing it to a configurable threshold. When a migration decision is triggered, the LCM requests the infrastructure's LMS (Local

Management System) to save the application state and terminate it. Subsequently, the AI-based LCM assesses the available infrastructure and deploys the application with its saved data onto this new infrastructure. The overall functioning of this solution is depicted in Figure 11 .

Currently, the selection of the new infrastructure relies on choosing the infrastructure with the highest available memory. However, this approach is subject to improvement in future versions by considering additional factors and exploring alternative methods for selection.

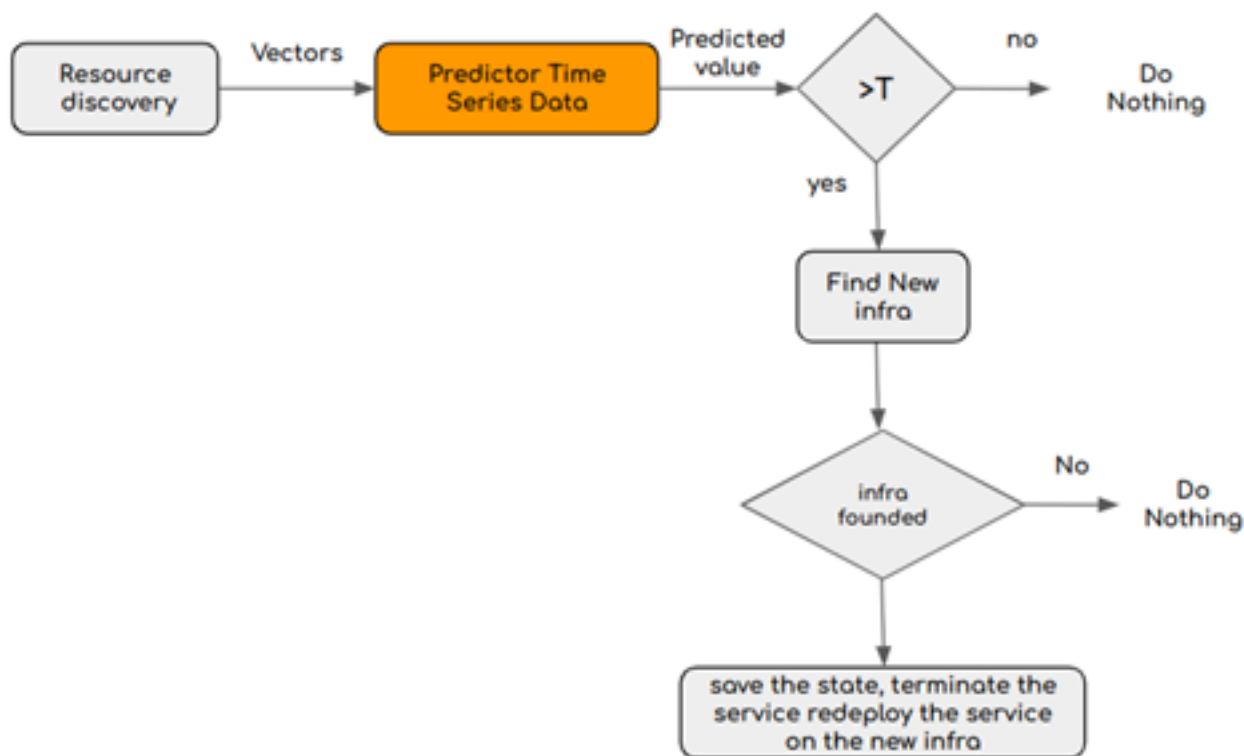


Figure 11 Fault tolerance Migration Algorithm

5.3 Architecture solution

To optimize the execution of this solution, we offer the foundational components for deployment within local systems, such as resource exposures facilitating resource discovery across the infrastructure's available resources. As mentioned previously, when a migration decision is made, the local infrastructure management system must preserve the states of stateful applications on hosts where potential SLA violations could occur. Considering that the majority of today's application deployments utilize containers, the migration solution should address the preservation of container states across various container environments. Currently, Kubernetes and its distributions stand out as the de facto container orchestration solution. In January 2023, Kubernetes introduced the checkpoint container feature as an alpha feature in version 1.25.0. The output of this Kubernetes checkpoint feature is an archive file containing the container state. This file serves as the foundation for constructing another container image from scratch, containing solely the checkpoint file. By running this image,

the operation to restart the container from the checkpoint file, known as the restore process, can be initiated. Upon running this newly constructed image, the underlying container runtime component will restart the original container from the instance when it was checkpointed. It's important to note that currently, the only container runtime implementing the checkpoint and restore feature in Kubernetes is CRI-O.

At present, the checkpoint feature is accessible at the Kubelet agent level via a REST API endpoint. The Kubelet agent¹¹ is the primary node agent that manages container operations on a Kubernetes node. However, to restart a container, users must manually build the image from the checkpoint. To streamline the integration of decision mechanisms with the checkpoint and restore features, we propose a component called the checkpointer. This component follows an operator pattern, a method that extends Kubernetes functionality by using custom resources and controllers to automate the management of complex stateful applications. Consisting of a controller and a custom resource definition (CRD), the controller reacts to the creation, editing, or deletion of a custom resource instance. Adopting the operator pattern offers several advantages, such as eliminating the need for additional logic code to handle API requests. Furthermore, it extends the Kubernetes API, facilitating direct interaction with the Kubernetes plugin component. Cluster administrators can benefit from simple YAML manifests to perform checkpoint, build, and push operations. The primary role of the checkpointer operator is to run on each node, checkpoint stateful containers upon request, create an image from this checkpoint, and push it to a shared image registry accessible across all nodes in all clusters. To achieve this, the custom resource should include pod namespace, pod name, container name for checkpointing, along with the image registry endpoint and credentials. Leveraging the Buildah code source, a tool designed for building OCI-compatible container images ensures the checkpointer's lightweight nature and easy deployability on each node in the cluster. Figure 12 depicts a sample of a custom resource instance utilized based on Redis¹² operating within the default namespace on the workernode01 node. The checkpointer will temporarily store the image in the Quay image registry, and its credentials must be provided. This addresses a crucial question regarding the security of the service migration process. The solution lies in securing the image registry where the checkpointed images will be temporarily housed.

¹¹ <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>

¹² <https://redis.io/>


```
apiVersion: cache.eurecom.com/v1alpha1
kind: Checkpoint
metadata:
  name: checkpointer-sample
spec:
  source_pod_name: redis
  source_pod_name_space: default
  source_pod_container: redis
  destination: quay.io/AC3/restore-counter:checkpointed-version
  registryUsername: ac3
  registryPassword: password
  sourceNodeId: workrrnode01
```

Figure 12 Checkpoint Object Sample

To enhance comprehension of this solution, we will delineate the communication workflows for migrating stateful services within nodes of the same cluster and across different clusters. The primary concept entails that when a migration decision is prompted by insufficient resources on the host node, the system initially seeks another available node within the same cluster. If no such node is found, it proceeds to schedule the service to another cluster with available resources. Figure 13 illustrates the workflow for migrating stateful containers within the same cluster, while Figure 14 outlines the workflow for migrating containers to another cluster.

Upon receiving a deployment request from the resource orchestrator, the cluster manager initiates the process by gathering metrics data from the cluster monitoring system. This data aids in selecting a suitable node for service deployment. The cluster manager then sends a request to the underlying plugin, specifying the desired node for service deployment. The Kubernetes plugin enforces this decision by forwarding the request to the cluster API server. Subsequently, the container runtime on the specified node pulls the necessary images from the shared image registry to initiate the service.

Simultaneously, the predictor model collects data from the cluster monitoring system to predict future memory consumption values. If high values are anticipated for a specific node, the model prompts the underlying plugin to generate custom resource instances to preserve the states of stateful containers on that node. The checkpointer controller on the respective node then checkpoints the container state when creating a custom resource instance. Concurrently, the Kubernetes plugin is instructed to delete the pod and forward the request to the cluster API server and the cluster manager. The new checkpointed image path is provided to redeploy the pod on a new node, initiating a new deployment process for the cluster manager. Here, the checkpointer controller builds an image from the checkpoint and pushes it to the image registry. Once completed, the cluster manager assigns the new node name to the plugin, which schedules the pod on the new node. During image retrieval, CRI-O recognizes the checkpointed image, facilitating the restoration of the container to its previous state.

In scenarios where all cluster nodes are fully utilized, as in small Kubernetes clusters, our solution handles this by initiating a migration request to another cluster. In such cases, the first cluster manager identifies, through metrics retrieved from the monitoring system, that the existing nodes cannot meet the service requirements. It then acts as a final user, requesting the resource orchestrator to deploy the service while providing the new checkpointed image path. The resource orchestrator consults the resource discovery module to obtain available resources and subsequently forwards the deployment request to another cluster. The service deployment in the second cluster is restored to its normal state, as if it were within the same cluster.

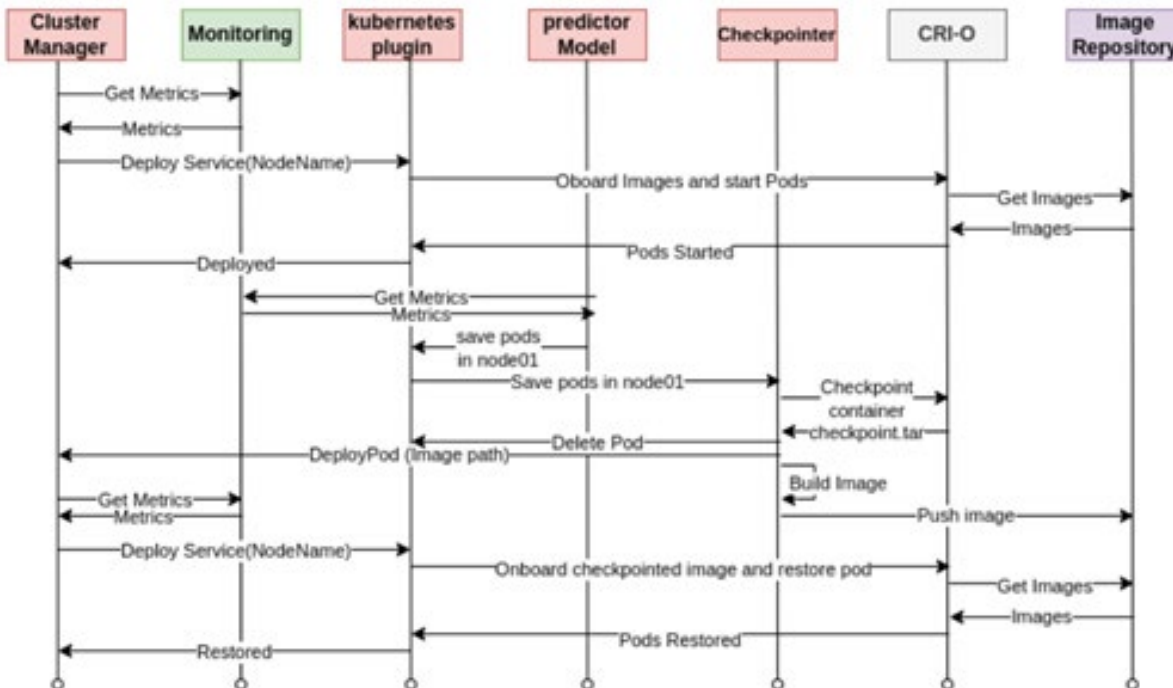


Figure 13 Intra-Cluster Migration Workflow.

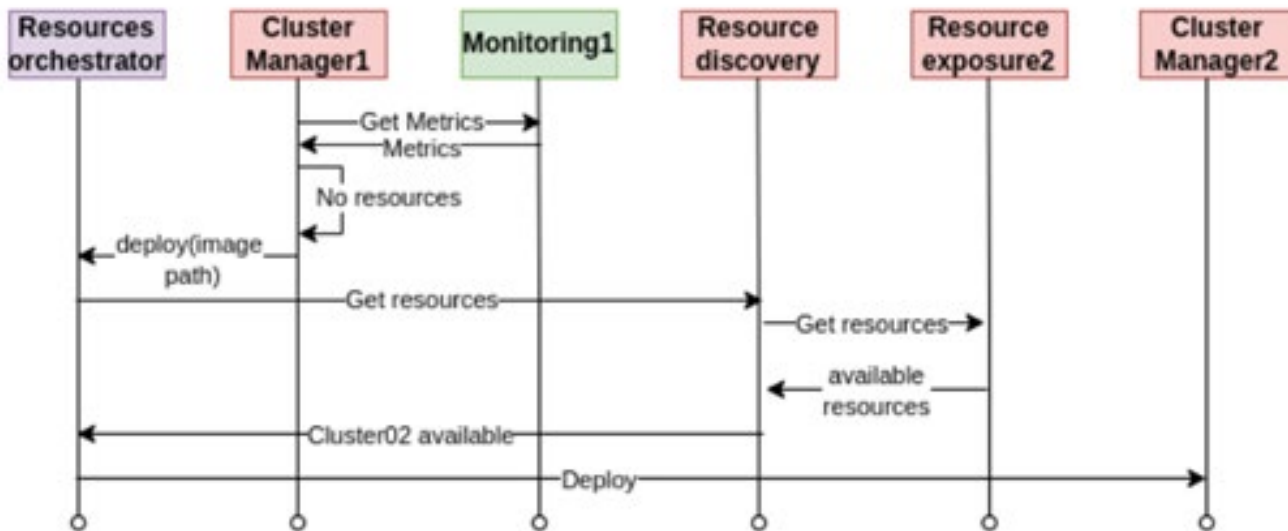


Figure 14 Intra-Cluster Migration Workflow.

5.4 Results

To evaluate the efficacy of our solution, a Kubernetes cluster configuration was implemented, consisting of one virtual machine serving as the master node and two additional virtual machines designated as worker nodes. The cluster was provisioned using Vagrant in conjunction with VirtualBox as the hypervisor. All machines within the cluster were equipped with Fedora 37 operating system, utilizing version 6.0.7 of the Linux kernel. Each machine was configured with 2 CPU cores and 4096 MB of RAM. The physical machine hosting these virtual machines features a 13th Gen Intel® Core™ i7-1365U processor with a maximum frequency of up to 3.70 GHz for E cores and up to 5.00 GHz for P cores, accompanied by 32 GB of LPDDR5-6400 MHz memory. Additionally, all machines in the cluster utilized version 1.26.0 of kubectl, kubelet, and kubeadm, along with version 1.26 of cri-o.

Two distinct types of applications were utilized to evaluate the effectiveness of our solution. The first application, drawn from a Kubernetes blog post ¹³, is a straightforward counter application ¹⁴. Its primary function is to increment a counter with each request, serving as a basic yet pertinent tool for evaluating Kubernetes’ checkpoint feature.

The second application employed in our testing regimen is a Redis in-memory database. We varied the size of data injected into Redis (10MB, 50MB, 100MB, 200MB, and 500MB) to create containers of different sizes. These containers represented various states that our operator needed to capture, build, and subsequently push to our local image registry. For each data size, we conducted multiple tests with varying numbers of containers, ranging from 1 to 5. In practice, we deployed a specified number of containers on the initial worker node (worker-

¹³ <https://kubernetes.io/blog/2022/12/05/forensic-container-checkpointing-alpha/>

¹⁴ <https://quay.io/repository/adrianreber/counter?tab=tags>

node01) and executed a memory-intensive program. Subsequently, our system autonomously decided to migrate all stateful containers from worker-node01 to worker-node02 based on predefined criteria.

As previously mentioned, our solution is housed within a lightweight migration controller tasked with checkpointing, building, and pushing the image. Given that this solution primarily addresses fault tolerance, we define the saving time as the cumulative duration required to rescue a container from a compromised node. This duration encompasses the time needed for checkpointing the container state, building an image from this checkpoint, and pushing this image to the image registry for utilization on the destination node. Furthermore, the image pulling time and restoration time in the destination node were taken into account.

The total saving time plot for single instances across different Redis database sizes (from empty to 500MB) illustrated in Figure 15a, indicates a predictable increase in saving time as the size of the container's state increases. This is expected since larger states take longer to checkpoint, build, and push due to the increased data volume, increasing the container size directly impacts various operational durations, subsequently influencing the overall container's saving time. Having conducted multiple iterations of the test set, we observed a level of stability across various operational durations. The box plots depicted in Figure 15b,c and d respectively illustrate the variation of the minimum, maximum, and median values of checkpointing, building, and pushing times for different images.

The limitation of migrating large container states manifests primarily during the image build operation and in the push phase. This process involves various steps, including packaging the application and its dependencies into a scratch container image, which is then pushed to a registry for deployment across different environments. The challenge becomes more apparent with larger container states due to the increased complexity and size of the image being constructed. The box plots representing checkpoint, build, and push times offer valuable insights into the variability and distribution of these operations across different container sizes. Remarkably, the median times for checkpointing remain relatively stable regardless of container size. This suggests that the checkpoint operation is not significantly affected by the state size, indicating a consistent performance in capturing the container's current state by the CRI-O container runtime.

However, when it comes to the build and push phases, a more notable increase in median times is observed as container sizes grow. This increase can be attributed to several factors. Firstly, the larger size of the container state necessitates more computational resources and time for packaging and assembling the image components. Additionally, the network transfer times also contribute to the overall duration, especially when pushing the image to a remote registry. In essence, while checkpointing operations demonstrate resilience to variations in container size, the build and push phases encounter notable hurdles when dealing with larger container states.

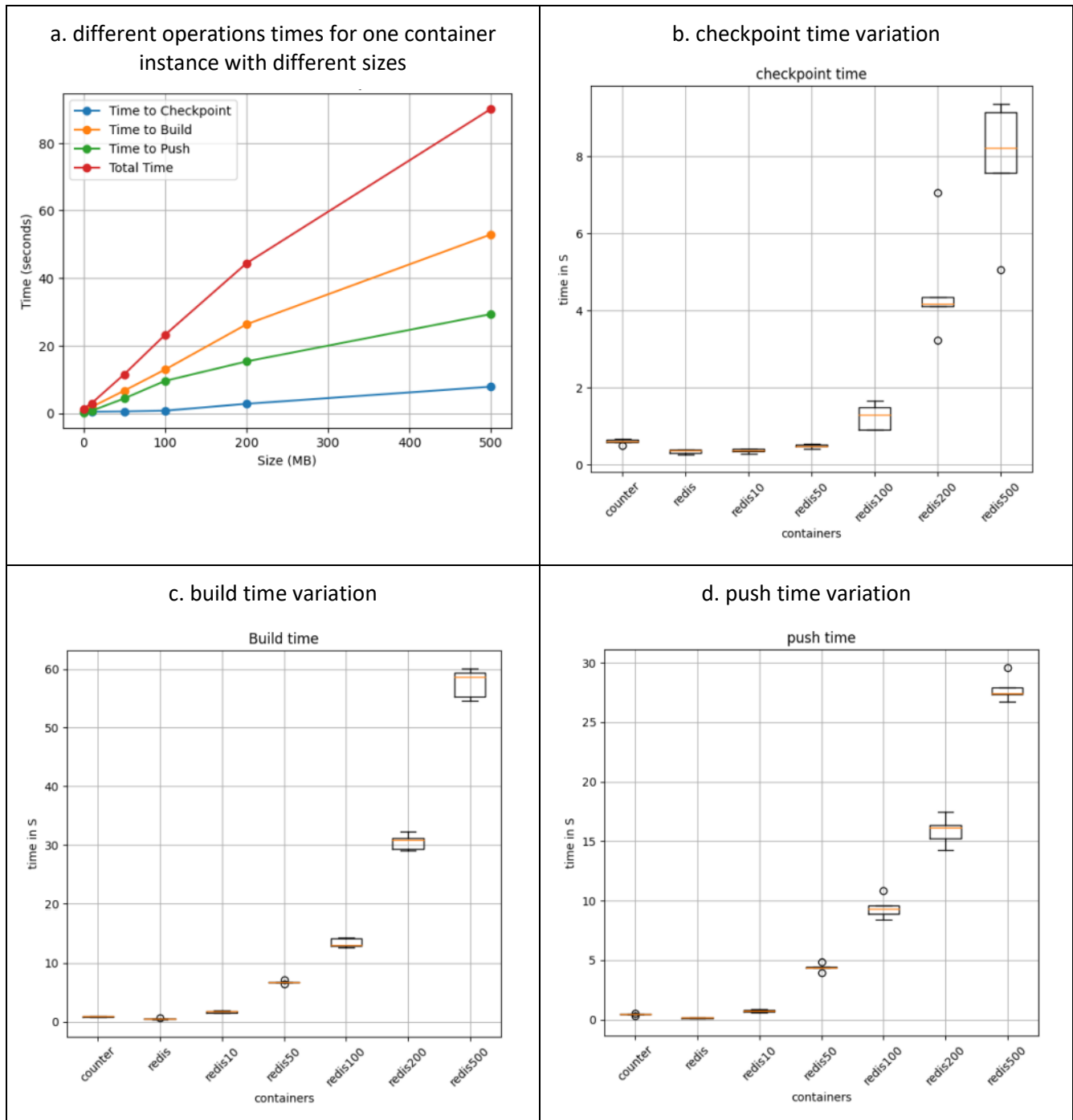


Figure 15 Benchmarking Results

Addressing these challenges necessitates optimizing resource allocation, enhancing network efficiency, and implementing strategies to streamline the packaging and deployment processes, thereby facilitating smoother migration of large container states across diverse environments.

The Figure 16 plot, which delineates the total saving time across multiple container instances, unveils a non-linear progression in time as additional containers are introduced. This non-linear trend implies concurrent execution of certain operations, such as the build process of one instance occurring simultaneously with the checkpointing of another. It is probable that resource contention, encompassing CPU, memory, and I/O bandwidth, emerges as a substantial contributor, hindering a linear scalability in performance.

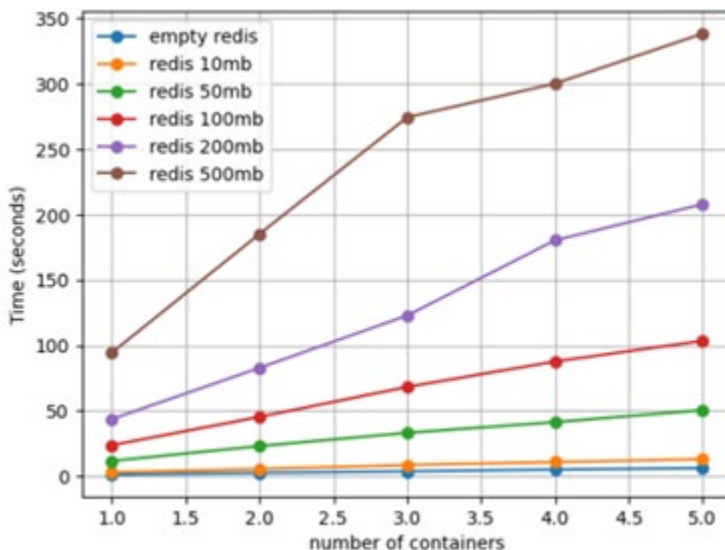


Figure 16 the total saving time of different container sizes with different numbers of instances.

Note that in our analysis, the pod restores time includes both the duration required for pulling the checkpointed image and initiating a running container from it. For the sake of simplicity and focus, we opt to disregard the scheduling and pod creation times in this context, as their durations are minimal and occur concurrently with the saving process. Consequently, the pod is ready to execute the container before the push operation concludes. The plot in Figure 17 illustrates the median values of container restoration, categorized by container size. It is evident that akin to other operation times, the restoration duration experiences a modest linear growth concerning the container size. Notably, external factors such as the location of the image registry and network bandwidth can significantly influence these values, primarily due to the predominant time allocated to pulling the image from the registry.

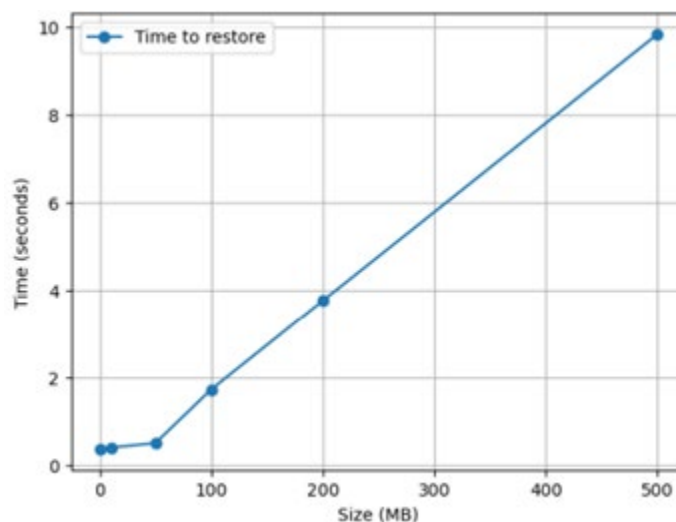


Figure 17 Average time to restore container with respect to its size.

5.4.1.1 Proactive mobility-induced stateful microservice migration

Modern applications like smart cities, smart transportation, autonomous driving, extended reality (XR), and immersive holographic communications have stringent requirements for storage, computing, latency, and bandwidth. This necessitates the network infrastructure to provide high-quality access to edge servers capable of supporting these applications. To meet these demands, various heterogeneous services will be offloaded from the cloud to edge servers, bringing them closer to end users. This shift in computing resources closer to the users is known as Edge Computing.

Since the edge computing paradigm is geographically distributed, its services are sensitive to users' mobility. As users move and change geographical areas, the consuming services in an edge server may be breached, or be located away from the user, downgrading the QoS of applications. Service migration (or, else, relocation) is one of the most used approaches for keeping critical services close to mobile users, thus enhancing QoS levels for the consuming applications. As a user moves in the underlying network but is still in the coverage of the serving edge host, i.e., intra-edge host mobility, the edge system does not need to relocate the service (i.e., application instance and/or state) to keep service continuity. However, if a user moves out of the coverage area of the source edge host to the coverage area of another edge host (destinate), i.e., inter-edge host mobility, this may result in service interruption to the user. In order to provide service continuity to the user the edge system needs to relocate the service from the source to the target edge host. In line with the ETSI MEC specifications [8], microservice migration may be addressed as either stateless or stateful. The latter requires not only relocation of the application instance (as stateless does) but also transfers the application state. In AC³, we target stateful applications to ensure seamless service continuity and minimal latency during the session transition. Based on the predicted user location, data caching can be triggered to start migrating the required state data to the destination edge node before handoff happens. In this direction and capitalizing on the relevant literature, we adopt a chunked data migration approach as in [9] that migrates state data following a proactive mobility-induced approach that is based on data access frequency. This approach will be part of the proposed solution that differentiates with [9] as it considers the innovative concept of Reinforcement Learning (RL) to design a more powerful State Manager promoting intelligence, automation and optimal decision making. Further details about the components of State Manager will be presented in the following paragraphs.

Description of proactive scheme

Service migrations are typically triggered reactively after handoff events, but this can cause disconnections and interruptions to user services. To guarantee service continuity for end-users, we assume a proactive scheme that encloses efficient mobility and QoS level prediction models to decide on where, when and if (based on QoS levels) service/data migrations can be performed taking into account the resources of edge servers on the predicted future location of the user.

Mobility-induced service migration in edge network involves several steps to ensure seamless transitions and optimal performance. Here's an outline of the process:

- **Identify Mobility Patterns:** Understand the mobility patterns of users within the edge network analyzing data on user movements, handoffs, and typical usage scenarios.
- **Application/Service Profiling:** Through profiling, the LCM can make informed decisions based on the latest predictions of profile metrics associated with the application/service running at the edge network to determine their resource requirements, latency sensitivity, and mobility constraints, as not all services may be suitable candidates for migration.
- **Location Awareness:** Consider mechanisms to gather information about the current location of the user or device accessing the service. This could involve integration with location-based services or leveraging mobility management protocols.
- **Migration Triggers:** Assume triggering events based on user mobility for initiating service and state migration. This means that a service may be migrated to a closer edge node when a user moves away from the serving node to a new location.
- **Migration Decision:** Decision-making is responsible for finding the optimal edge node to initiate service and state data parts migration after triggering by the output of (when and where) machine learning algorithms that predict user mobility.
- **Migration Execution:** Implement mechanisms for executing service migration in a controlled and coordinated manner. This usually involves transferring stateful components of the service, updating routing configurations, and ensuring data consistency. Stateful migration execution entails the state data transfer and the running application instance relocation.
- **Handoff Management:** Manage the handoff process from one edge node to another to minimize service disruption and thus ensure continuity for users. This includes maintaining session state, updating DNS records, and rerouting traffic seamlessly. Also, service handoff efficiency is affected by many factors in the edge nodes environment: the network conditions between edge nodes, the network conditions between end user and edge node and the available resources on the edge nodes, such as available CPU, Memory, Storage.
- **Monitoring and Optimization:** Continuously monitor the performance of migrating services and the effectiveness of mobility-induced migration strategies. Make adjustments as needed to optimize resource utilization, minimize latency, and enhance user experience.

By following these steps, we can effectively implement a mobility-induced stateful service migration in an Edge Computing environment, improving agility, performance, and user satisfaction. The central entity of the proposed solution is the so called “State manager” that implements internally the above mechanisms for migrating (or replicating) an application state from source to destination edge node (EN) assuming that state data is stored in Redis, an independent, in-memory data storage. The mechanism decides “when and where” migration should happen according to the motion information of the mobile user (MU), taking into account ENs environment conditions to maintain the users’ experience by optimizing resource allocation at ENs. Based on the predicted future location of the user, it finds a list L^{EN} with the k nearest candidate destination ENs (d-ENs) around MU and learns among them the optimal d-EN to keep on serving the application assuming an RL approach that guides/governs the migration process of microservices on the edge nodes to optimize via the reward function

the trade-off between application SLA (mainly latency) and resources usage at edge. Selecting the d-EN from the k nearest ENs, we anticipate minimization of migration time, power consumption and computational cost taking into account EN resources. The State Manager is composed by the following key components

A module observes/monitors the user's mobility information and based on an AI model predicts the future location of the MU. Users' mobility trace data are deemed as time-series data gathered at different timestamps. LSTM is a neural network model that can keep a memory of previous inputs and is particularly efficient for time series prediction. Therefore, an LSTM is used to address the design of user's trajectory predictor for extracting the future location information from historical records. Exploiting the output of LSTM, it identifies the specific list L^{EN} with the k -NN d-ENs in the MU's forecasted location.

Moreover, during the execution time of service, the edge nodes' environment may be highly dynamic by appearing fluctuations in key performance metrics, such as available CPU and Memory. By monitoring and collecting such data for the ENs in the list L^{EN} , not the whole edge nodes environment, an efficient time series prediction model, such as LSTM, can forecast CPU and/or Memory resource usage [10]. This information will help to assess the k proximal edge node's status and exclude those with potential failure to properly execute the assigned service according to the application resource requirements (Application profile) due to non-adequate resources (e.g., overload). So, a refined list of ENs is derived which is noted as R^{EN} . Then, its output will feed/trigger Decision Module informing it about when the mobile user node is likely to go towards its new location and the list with the potential new serving d-ENs.

The decision module finds the optimal d-EN from the list R^{EN} by employing an RL agent that runs at s-EN and explores interactions with the edge nodes in R^{EN} to learn the best strategy for the rewards. Within AC³, the RL agent is opted to manage optimally edge network resources under the problem of state data relocation in parallel with application migration. RL algorithms help to automate and optimize the decision-making process of migrating services between different edge nodes (i.e., in distributed computing environments), adapt to changing conditions, and optimize resource utilization. However, given the nature of RL, it's essential to carefully design the RL framework and validate its performance considering factors such as learning convergence speed, scalability and adaptability-robustness in dynamic real-world deployments.

Here's an outline of how RL is applied to service migration:

- **Environment:** Define/Model the edge nodes environment in which service migration occurs, including factors such as network conditions, resource availability, workload characteristics, and cost metrics.
- **RL State Representation:** Develop a representation of the edge nodes environment state that captures relevant information for decision-making. This may include information about the current state of the system, such as resource utilization, latency, and workload demand.
- **Action Space:** Define the actions that the RL agent can take to migrate services, such as selecting target edge node adjusting resource allocations. The action space should encompass a range of possible migration strategies and configurations.
- **Reward Function:** Design a reward function that quantifies the QoS migration decisions based on predefined objectives and metrics. This may include factors such as resource utilization, minimizing latency, maintaining service-level agreements (SLAs) or maximizing reducing cost.
- **RL Algorithm Selection:** Choose an appropriate RL algorithm for learning the optimal policy for service migration. Common RL algorithms for this task include Q-learning, Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), or Actor-Critic methods.
- **Training Process:** Train the RL agent to learn a migration policy that maximizes the cumulative reward function. The RL agent uses the generated training data to learn an optimal policy for service migration.

This involves iteratively updating the agent's parameters based on the observed rewards and the chosen RL algorithm.

- **Deployment and Evaluation:** Deploy the trained RL agent and evaluate its performance in orchestrating service migrations, monitor key performance indicators and validate that the RL-based migration strategy achieves the desired objectives
- **Learning and Adaptation:** Update and refine the RL-based migration policy based on feedback from the environment and changes in workload patterns or infrastructure conditions. This may involve online learning techniques or periodic retraining of the RL agent.
- **Integration with Orchestration System:** Integrate the RL-based migration approach with an existing orchestration system, and specifically Kubernetes (K8s), to automate the deployment and management of migrated services in dynamic environment [9][11][12]. This way, the applications can be packaged as containers and relocated, along with their state, using container migration technologies aided by the elaborated proactive mobility-induced mechanism that pipelines with the application and edge resources profile and an LSTM-based model for location prediction before running the RL agent at s-EN edge node.

The decision module under a soft migration model [10] selects proper data to be transferred (not the whole state) proactively to the d-EN assuming a chunked data migration strategy that is based on data access frequencies. Following the approach in [9], data are characterized as cold (least accessed) and hot (most accessed) according to their access frequencies. This discrimination helps determine the probability of data migration indicating that cold data are more inclined to be part of the proactive migration process.

To estimate the access frequency per chunk of the state data, we calculate the total number of operations O_k at chunk k (insert I_k and update U_k) until certain time t : $O_k = I_k(t) + U_k(t)$. By applying the previous formula to all data chunks, it is possible to obtain the value of access frequencies f_k per data chunk k as: $f_k = \frac{O_k}{\sum_{i=1}^n O_i}$. Finally, the migration probability assigned to each data chunk k is defined as: $P(s) = \frac{1}{f}$. State data migration takes place before the handoff happens if $P(s) \geq \tau$, where τ is a probability threshold that depends on data characteristics. Thus, the s-EN migrates proactively the service data part to the d-EN. In this time interval, the mobile user node continues to be provisioned through the s-EN. Also, cached/migrated data will be periodically checked in terms of accuracy and integrity.

Within AC³, the RL agent's design for managing edge network resources under the problem of application and optimal state data migration will be based on advanced Deep Q-Learning Network (DQN) architectures. Therefore, the interaction between the agent and the ENs environment is conceptualized as a Markov Decision Process (S, A, R, γ). In this process (Figure 18):

- **State $s \in S$** (with S being the state space) keeps the current allocation/usage of resources across the d-ENs in list R^{EN} , including the s-EN (CPU, memory, storage), amount of microservices deployed, resources required for the microservices and monitored KPIs for microservices.

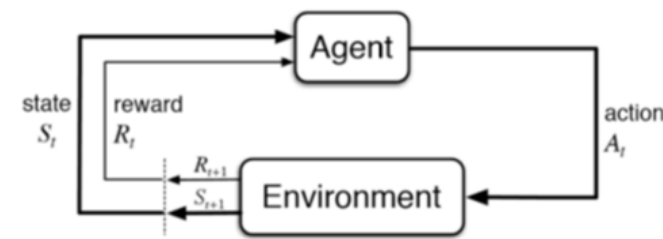


Figure 18 RL agent

- **Actions** $a \in A$ (with A being the actions space) represent the *ID* of all possible d-ENs in list R^{EN} , each with an associated confidence metric that quantifies the d-EN's readiness to host the service.
- **Reward function R** captures the agent's goal during action selection and, in AC³, is designed to balance the SLA-violations and latency across the network, directing the agent towards the optimal orchestration of ENs resources.
- The **policy π** stands for the distribution that maps state to action while $Q(s, a)$ is the action-value function under **policy π** .
- γ is a discount factor in the range $[0,1]$ involved in the estimation of the value function that helps to find the appropriate action a .

To ensure a stateful system, state data is stored in an in-memory data storage and specifically Redis. Using Redis for live microservice migration can enhance the process by providing fast and efficient data storage, caching, and messaging capabilities. Here's how Redis can be leveraged in various aspects of microservice migration:

Service Discovery: Redis can serve as a centralized service registry where microservices can register themselves upon startup. This enables dynamic discovery of services by other microservices, facilitating communication and coordination during migration.

Configuration Management: Redis can store configuration parameters for microservices in key-value pairs or JSON structures. During migration, configuration changes can be applied in real-time by updating values in Redis, allowing for dynamic adjustment of microservice behavior without requiring downtime.

Caching: Redis's in-memory caching capabilities can improve the performance of microservices by storing frequently accessed data closer to the application. During migration, Redis can continue serving cached data even if the microservice instances are temporarily unavailable or being migrated to new nodes, reducing the impact on end-users.

Messaging and Coordination: Redis can be utilized for messaging and coordination between microservices during migration. For example, when a microservice instance is ready to be migrated, it can publish a message to Redis indicating its readiness, and other microservices can subscribe to these messages to take appropriate actions, such as updating their routing tables or adjusting their behavior.

Session Management: Redis can be used to manage session data in distributed microservice architectures. During migration, session state can be seamlessly transferred between instances using Redis as a shared session store, ensuring continuity for users across different microservice versions or nodes.

Health Checking and Monitoring: Redis can be integrated with monitoring tools to perform health checks on microservice instances and track their availability and performance metrics during migration. This enables proactive detection of issues and automatic failover or rollback procedures if necessary.

Overall, leveraging Redis in live microservice migration can contribute to improved scalability, reliability, and agility by providing a versatile data storage and messaging platform that supports real-time coordination and seamless transitions between microservice versions or instances.

5.5 Planned implementations and extensions for final phase

5.5.1 Current Implementation status

Most components of the application layer within our framework have been successfully implemented. This includes local management systems and several underlying plugins such as OpenShift and Kubernetes plugins. Additionally, the Checkpointer Operator, which is designed to save the state of stateful containers as needed, is also in place. However, there is potential for further optimization of the Checkpointer values, particularly concerning the management of simultaneous checkpointing processes.

5.5.2 Resource Exposure Component

The implementation phase of the Resource Exposure component is currently underway. Our objective is to ensure compatibility with multiple cache technologies and various monitoring tools, including Prometheus. This compatibility is essential for enhancing the flexibility and robustness of our framework.

5.5.3 AI-based LCM (Lifecycle Management)

For the AI-based LCM part, we have developed a model capable of predicting future memory consumption values. Nevertheless, it is crucial to extend the model's predictive capabilities to encompass other metrics, such as CPU utilization and network resources, including latency between different sites. Future versions of the model should either consolidate these predictions into a single model with multiple inputs or employ different models for each metric.

To achieve this, we are currently utilizing the GWA-T-13_Materna-Workload-Traces dataset. However, the data in this dataset is collected at five-minute intervals, which poses a challenge given the large time window in the context of computing and network resource utilization. Therefore, exploring alternative data sources for training our models may be necessary to achieve more accurate and reliable results.

Moreover, to efficiently tackle the case where stateful migration is triggered due to user mobility, another model that exploits trajectory historical time series data has been developed to predict the future location of the user. Then, its output will feed a Deep RL-based procedure that makes the final decision about the new edge node for the proactive microservice migration (taking into account the application profile and edge nodes predicted status in terms of resources at the new location). In the next period, we aim to finalize the optimal architecture and parameter selection for the selected deep models experimenting with appropriate datasets (e.g., from CRAWDDAD) for their training and develop an evaluation procedure in order to demonstrate their models' performance as a whole.

6 Conclusions

In the deliverable titled "Report on the Application LCM in the CEC – Initial," the primary goal is to clarify the various components related to the user plane within the Cloud-Edge Continuum Configuration Management (CECCM) system. This deliverable comprehensively examines the components that interact directly with application developers, providing critical insights and functionalities necessary for effective application lifecycle management. Key components include user interfaces, which enable developers to interact seamlessly with the CECCM system, facilitating the development, deployment, and management of applications through structured YAML and JSON files. These user interfaces serve as the primary gateway for developers, offering an intuitive and efficient means to oversee application management processes. Additionally, the application profiles, ontology modeling tools, and application descriptor models are pivotal in defining the structure, requirements, and intents of applications. They provide a robust framework for creating detailed application descriptors, encapsulating all necessary information for deployment and management. The service and data catalog are a centralized repository of services and data sources within the CECCM system, allowing developers to explore and integrate various resources and ensuring that applications are constructed using the most suitable components and data sources available.

The introduction of the AI-based Lifecycle Management System (LCM) is a significant milestone in this deliverable since it is cornerstone of AC³ platform, encompassing both deployment and runtime management. This system leverages advanced AI algorithms to optimize the initial placement of microservices based on detailed application and resource profiles, enhancing overall efficiency and performance. Also, solutions for micro-service migration, both executed by the AI-based Lifecycle Management System (LCM) during runtime procedures were presented. Finally, the next steps on finalizing and optimizing several key components of AC³ are presented.

7 References

- [1] "S. Li, J. Bi, H. Yuan, M. Zhou, and J. Zhang, "Improved LSTM-based prediction method for highly variable workload and resources in clouds," in 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2020, pp. 1206–1211."
- [2] "R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks," in 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS), IEEE, 2017, pp. 1597–1600."
- [3] "C. Sudhakar, A. R. Kumar, N. Siddartha, and S. V. Reddy, "Workload prediction using arima statistical model and long short-term memory recurrent neural networks," in 2018 International Conference on Computing, Power and Communication Technologies (GUCON),, " , IEEE, 2018, pp. 600–604..
- [4] "M. E. Karim, M. M. S. Maswood, S. Das, and A. G. Alharbi, "BHyPreC: a novel Bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine," IEEE Access, vol. 9, pp. 131476–131495, 2021."
- [5] "A. S. Hati, "Convolutional neural network-long short term memory optimization for accurate prediction of airflow in a ventilation system," Expert Syst Appl, vol. 195, p. 116618, 2022."
- [6] "J. Wang, Y. Yan, and J. Guo, "Research on the prediction model of CPU utilization based on ARIMA-BP neural network," in MATEC Web of Conferences, EDP Sciences, 2016, p. 03009."
- [7] "K. Ray, A. Banerjee, and N. C. Narendra, "Proactive microservice placement and migration for mobile edge computing," in 2020 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2020, pp. 28–41."
- [8] "G. ETSI, "Mobile edge computing (MEC); End to end mobility aspects," ETSI Standards Search, 2017."
- [9] "P. Bellavista, A. Corradi, L. Foschini, and D. Scotece, "Differentiated service/data migration for edge services leveraging container characteristics," IEEE Access, vol. 7, pp. 139746–139758, 2019."
- [10] "V. Tsakanikas, T. Dagiuklas, M. Iqbal, X. Wang, and S. Mumtaz, "An intelligent model for supporting edge migration for virtual function chains in next generation internet of things," Sci Rep, vol. 13, no. 1, p. 1063, 2023."
- [11] "F. Barbarulo, C. Puliafito, A. Viridis, and E. Mingozzi, "Extending ETSI MEC towards stateful application relocation based on container migration," in 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM),, " IEEE, 2022, pp. 367–376..
- [12] "M.-N. Tran, X. T. Vu, and Y. Kim, "Proactive stateful fault-tolerant system for kubernetes containerized services," IEEE Access, vol. 10, pp. 102181–102194, 2022."

Annex I: Example of an Application Descriptor

An Example of an Application Descriptor that may be used for deployment.

```
application:
name: "Surveillance System"
id: "12345678-1234-5678-1234-567812345678"
version: "1.0"
description: "Real-time surveillance system for monitoring video feeds and
analytics."
#####
####
#####
####
application_consumers:
- profile_type:
characteristics:
interfaces_experiences:
interface_type: ""
endpoint: ""
technologies: ""
consumer_specific_functionality:
authentication_and_authorization:
method: ""
permissions: ""
security_measures: ""
```

```

usage_patterns:
#           Define           usage           patterns           here.
#####
#####
#####           Surveillance           System           application           consumers
#####
#####
#####
application_consumers:
-           profile_type:           "Customer"
characteristics:
interfaces_experiences:
interface_type:           "Web           Application"
endpoint:           "http://surveillance.fingletek.com"
technologies:           "HTTP/REST"

consumer_specific_functionality:
authentication_and_authorization:
method:           "Username           and           Password"
permissions:           "Defined           by           the           admin"
security_measures:           "TLS           encryption"

-           profile_type:           "Admin"
characteristics:
interfaces_experiences:

```



```

interface_type: "Web" "Application"
endpoint: "http://surveillance.fingletek.com"
technologies: "HTTP/REST"

consumer_specific_functionality:
authentication_and_authorization:
method: "Username" "and" "Password"
permissions: "CRUD"
security_measures: "TLS" "encryption"

#####
####

#####
####

microservices:
- name: ""
id: ""
service_type: ""
purpose: ""
image: ""
version: ""

api_endpoints:
- endpoint: ""
purpose: ""
method: ""

```

```
dependencies:
-
resource_requirements:
cpu:
memory:
storage:

operational_support_systems:
monitoring_and_auditing:
monitoring:
logging:
audit_and_monitoring:

scalability_and_load_balancing:
deployment_scaling_instructions:
load_balancing_strategy:
scaling_policies:
scalability_thresholds:
replica_count:

resource_management:
resource_requirements:
```

```

resource_updates: ""
resource_allocation: ""

performance_metrics:
response_time_variability: ""
concurrent_user_load: ""
data_throughput: ""

incident_response_and_compliance:
failure_handling_time: ""
sla_integration: ""

configuration_and_environment:
environment_variables:
image_pull_policies: ""

#####
####

#####      Surveillance      System      microservices      decomposition
#####

#####
####

microservices:
-           name: "backend"
id:         "154fe329-5edf-4833-bbbb-668c0df2bcf0"
purpose:    "Manage surveillance operations and real-time video processing."
image:      "https://fingletek.com/surveillance-backend:latest"

```

```
version: 1.0

api_endpoints:
  API_ENDPOINT_BASE_URL: "http://backend.fingletek.com"
  API_ENDPOINT_PORT: "7086"
  purpose: "Backend" API
  method: "HTTP/REST"

dependencies:
  - name: database
  id: 82bbcd33-b389-4c16-b312-0144bf4d00bc

resource_requirements:
  cpu: "4" vCPU
  memory: "8GB"

max_requirements:
  cpu: "4" vCPUs
  memory: "8Gi"

operational_support_systems:
  monitoring: "Prometheus"
  logging: "ELK" Stack
  audit_and_monitoring: "Auditbeat"
```

```
scalability_and_load_balancing:
deployment_scaling_instructions: "Horizontal scaling using Kubernetes"
load_balancing_strategy: "Round Robin"
scaling_policies: "Auto-scaling based on CPU usage"
scalability_thresholds: "Triggered at 70% CPU usage"
replica_count: 3

resource_management:
resource_requirements: "Defined in Kubernetes manifest"
resource_updates: "Automatic resource updates"
resource_allocation: "Managed by Kubernetes"

performance_metrics:
response_time_variability: "Low"
concurrent_user_load: "Up to 1000 users"
data_throughput: "High"

incident_response_and_compliance:
failure_handling_time: "Within 5 minutes"
sla_integration: "Integrated with SLA management system"

configuration_and_environment:
environment_variables:
```

```
-           DB_HOST:           "database"
-           DB_PORT:           "5432"
-           DB_NAME:           "uc2"
-           DB_USER:           "postgres"
-           DB_PASS:           "root"
image_pull_policies:  "Pull      latest      image      on      deployment"

-           name:              "frontend"
id:           "4c7e08c6-6df4-11ee-b962-0242ac120002"
purpose:      "User      interface      for      real-time      surveillance      monitoring."
image:        "https://fingletek.com/frontend-surveillance:latest"
version:      1.0

api_endpoints:
-           endpoint:          "http://frontend.fingletek.com"
port:         "3000"
purpose:      "Frontend      UI      API"
method:       "HTTP/REST"

dependencies:
-           name:              backend
id:           154fe329-5edf-4833-bbbb-668c0df2bcf0
-           name:              videoanalytics
id:           579afc46-6df4-11ee-b962-0242ac120002
```

```
resource_requirements:
min_requirements:
cpu: "1" vCPUs"
memory: "2Gi"

max_requirements:
cpu: "4" vCPUs"
memory: "8Gi"

operational_support_systems:
monitoring: "Prometheus"
logging: "ELK" Stack"
audit_and_monitoring: "Auditbeat"

scalability_and_load_balancing:
deployment_scaling_instructions: "Horizontal scaling using Kubernetes"
load_balancing_strategy: "Round Robin"
scaling_policies: "Auto-scaling based on CPU usage"
scalability_thresholds: "Triggered at 70% CPU usage"
replica_count: 3

resource_management:
resource_requirements: "Defined in Kubernetes manifest"
```

```
resource_updates:      "Automatic      resource      updates"
resource_allocation:   "Managed      by      Kubernetes"

performance_metrics:

response_time_variability:      "Low"

concurrent_user_load:      "Up      to      1000      users"

data_throughput:      "High"

incident_response_and_compliance:

failure_handling_time:      "Within      5      minutes"

sla_integration:      "Integrated      with      SLA      management      system"

configuration_and_environment:

environment_variables:

-      BACKEND_API_URL:      "http://backend.fingletek.com"
-      BACKEND_API_PORT:      "7086"
-      WEBSOCKET_BASE_URL:      "ws://fingletek.socket.connection"
-      WEBSOCKET_PORT:      "8585"

image_pull_policies:      "Pull      latest      image      on      deployment"

-      name:      "videoanalytics"

id:      "579afc46-6df4-11ee-b962-0242ac120002"

service_type:      "Video      Analytics"

purpose: "High-performance GPU-accelerated video analytics using NVIDIA DeepStream."
```



```
image: "https://fingletek.com/videoanalytics:latest"
version: 1.1.0

api_endpoints:
- endpoint: ""
  purpose: "Video Analytics Engine"
  method: "TCP/RTSP"

dependencies:
- name: backend
  id: 154fe329-5edf-4833-bbbb-668c0df2bcf0
- name: database
  id: 82bbcd33-b389-4c16-b312-0144bf4d00bc
- name: IOTDevice-Camera
  id: b654f0a2-6d0e-11ee-b962-0242ac120002

resource_requirements:
cpu: "4 vCPUs"
memory: "16GB"
gpu: "NVIDIA GPU (specific model based on throughput requirements)"

max_requirements:
cpu: "8 vCPUs"
memory: "16GB"
```

```
gpu: "NVIDIA GPU (specific model based on throughput requirements)"

operational_support_systems:
monitoring: "Prometheus"
logging: "ELK Stack"
audit_and_monitoring: "Auditbeat"

scalability_and_load_balancing:
deployment_scaling_instructions: "Horizontal scaling using Kubernetes"
load_balancing_strategy: "Round Robin"
scaling_policies: "Auto-scaling based on CPU usage"
scalability_thresholds: "Triggered at 70% GPU usage"
replica_count: 3

resource_management:
resource_requirements: "Defined in Kubernetes manifest"
resource_updates: "Automatic resource updates"
resource_allocation: "Managed by Kubernetes"

performance_metrics:
response_time_variability: "Low"
concurrent_user_load: "Up to 1000 users"
data_throughput: "High"
```

```
incident_response_and_compliance:
failure_handling_time:          "Within          5          minutes"
sla_integration:                "Integrated    with    SLA    management    system"

configuration_and_environment:
environment_variables:
-          CAMERA_URL:          "rtsp://camera-device.fingletek.com"
-          CAMERA_PORT:        "554"
-          DB_HOST:            "database"
-          DB_PORT:            "5432"
-          DB_NAME:            "uc2"
-          DB_USER:            "postgres"
-          DB_PASS:            "root"
-          DATA_TYPE:         "Hot"
image_pull_policies:           "Pull    latest    image    on    deployment"
-          name:                "database"
id:                             "82bbcd33-b389-4c16-b312-0144bf4d00bc"
purpose:                        "PostgreSQL          Database          Service"
image:                          "https://hub.docker.com/\_/postgres"
version:                         8.1.0

api_endpoints:
ENDPOINT_URL:                   "psql://database.fingletek.com"
```

```
ENDPOINT_PORT : "5432"
purpose: "PostgreSQL Database Access"
method: "TCP"

dependencies: []

resource_requirements:
cpu: "0.2 vCPU"
memory: "256 MB"
storage: "10 GB"

max_requirements:
cpu: "1 vCPUs"
memory: "2Gi"
storage: "30 GB"

operational_support_systems:
monitoring: "Prometheus"
logging: "ELK Stack"
audit_and_monitoring: "Auditbeat"

scalability_and_load_balancing:
deployment_scaling_instructions: "Horizontal scaling using Kubernetes"
load_balancing_strategy: "Round Robin"
```

```
scaling_policies:      "Auto-scaling      based      on      CPU      usage"
scalability_thresholds:  "Triggered      at      70%      CPU      usage"
replica_count:                3

resource_management:

resource_requirements:      "Defined      in      Kubernetes      manifest"
resource_updates:          "Automatic      resource      updates"
resource_allocation:        "Managed      by      Kubernetes"

performance_metrics:

response_time_variability:                "Low"
concurrent_user_load:      "Up      to      1000      users"
data_throughput:                "High"

incident_response_and_compliance:

failure_handling_time:      "Within      5      minutes"
sla_integration:      "Integrated      with      SLA      management      system"

configuration_and_environment:

environment_variables:
-      MYSQL_ROOT_PASSWORD:      "password"

-      name:      "IOTDeviceCamera"
```

```
service_id: "b654f0a2-6d0e-11ee-b962-0242ac120002"
purpose: "Captures video data and metadata for streaming and storage."
image: "http://fingletek.com///camera-service-image:latest"
version: 1.0.3

api_endpoints:
RTSP_BASE_URL: "rtsp://camera-device.fingletek.com"
RTSP_PORT: "554"
purpose: "Video Stream"
method: "RTSP/TCP"

dependencies: []

resource_requirements:
cpu: "0.3 vCPU"
memory: "384 MB"
storage: "30 GB"

max_requirements:
cpu: "2 vCPUs"
memory: "2Gi"
storage: "50 GB"

operational_support_systems:
```

```
monitoring_and_auditing:
monitoring: "Prometheus"
logging: "ELK Stack"
audit_and_monitoring: "Auditbeat"

scalability_and_load_balancing:
deployment_scaling_instructions: "Horizontal scaling using Kubernetes"
load_balancing_strategy: "Round Robin"
scaling_policies: "Auto-scaling based on video stream"
scalability_thresholds: "Triggered at 75% video stream capacity"
replica_count: 3

resource_management:
resource_requirements: "Defined in Kubernetes manifest"
resource_updates: "Automatic resource updates"
resource_allocation: "Managed by Kubernetes"

performance_metrics:
response_time_variability: "Low"
concurrent_user_load: "Up to 1000 users"
data_throughput: "Very High"

incident_response_and_compliance:
```

```
failure_handling_time:          "Within          2          minutes"
sla_integration:                "Integrated      with      SLA      management      system"

configuration_and_environment:

environment_variables:

-          CAMERA_DB_URL:          "psql://database.fingletek.com"
-          CAMERA_DB_PORT:          "5432"
-          CAMERA_SECRET:          "ultrasecretkey"
-          CAMERA_PORT:            "554"
image_pull_policies:            "Pull          latest      image      on      deployment"

#####
####

#####
####

data_sources:

-          source_name:            ""

category:

-          "Hot/Cold"

details:

#          We          can          define          data          sources          details          here.

geographical_location:

-          country:                "Country          name"

coordinates:

longitude:                      ""

latitude:                       ""
```



```

broker_address: "GPSTBroker.fingletek.com"

data_models:
-
    model_name: ""
attributes:
# Define attributes for the data model.

data_access_and_security:
access_method: ""
security_measures: ""

#####
####

##### Surveillance System data sources
#####

#####
####

data_sources:
-
    source_name: "Video Stream"
category: "Hot"
details: "Real-time video streams captured by cameras."

data_models:
-
    model_name: "VideoDataModel"
attributes:
-
    "Video File"
-
    "Timestamp"

```

```
-           "Camera"                                     ID"

data_access_and_security:
access_method:                                     "RTSP/HTTP"
security_measures:  "Data encrypted in transit using TLS."

-           source_name:                               "User"           Data"
category:                                             "Cold"
details:  "User accounts, roles, and permissions data."

data_models:
-           model_name:                               "UserDataModel"
attributes:
-           "Username"
-           "User"                                     ID"
-           "Role"
-           "Permissions"

data_access_and_security:
access_method:                                     "HTTP/REST"
security_measures: "Data access controlled by authentication and authorization."

#####
####

#####
####
```

```
deployment_context:
scalability:
#           Define           scalability           context.
containerization_details:           ""
#   Define   container   technology   -   e.g.,   Docker,   Kubernetes
availability_and_resilience:
cloud_platforms:           ""
# Defines preferred cloud platforms - e.g., Amazon AWS, Microsoft Azure, Google
cloud           platform
networking_and_traffic_management:
#   Define   networking   and   traffic   management   details.

location:
-           microservice_id:           ""
#   Specifies   a   microservice   to   be   deployed.
microservice_affinity:           ""
# Values here could be Same Clusters, Same Node and if left empty, the system can
decide           where           to           place           it
geographical_affinity:           ""
# Defines the cloud type e.g., Central Cloud, edge, far edge, etc.
region_id:           ""
# Identifies the region/location for deploying the microservice by the ID.
instantiation_order:           ""
#   Defines   the   order   of   deployment   of   the   microservice.
```

```
service_mesh_integration: ""

policy_and_compliance_management:
#   Define   policy   and   compliance   management   details.

sla_and_policy_engine:
#   Define   SLA   and   policy   engine   details.

rate_limiting_and_throttling_policies:
#   Define   rate   limiting   and   throttling   policies.

maintenance_and_update_cycles:

ci_cd_pipeline: ""

data_management: ""

#####
#####

#####   Surveillance   System   deployment   Context
#####

#####

deployment_context:

scalability:   "Elastic   scaling   based   on   data   volume."

containerization_details:   "Docker,   Kubernetes"

availability_and_resilience:

cloud_platforms:   "Amazon   AWS,   Microsoft   Azure,   Google   Cloud   Platform"

networking_and_traffic_management:
```

```
location:
- microservice_id: "82bbcd33-b389-4c16-b312-0144bf4d00bc" # ID of the "database"
  service
microservice_affinity: ""
geographical_affinity: "Central" "Cloud"
region_id: "3"
instantiation_order: "1"

- microservice_id: "b654f0a2-6d0e-11ee-b962-0242ac120002" # ID of the "IOTDevice-
  Camera" service
microservice_affinity: "Node"
geographical_affinity: "Edge"
region_id: "1"
instantiation_order: "2"

- microservice_id: "154fe329-5edf-4833-bbbb-668c0df2bcf0" # ID of the "backend"
  service
microservice_affinity: "Cluster" id: 82bbcd33-b389-4c16-b312-0144bf4d00bc
geographical_affinity: "Central" "Cloud"
region_id: "3"
instantiation_order: "3"

- microservice_id: "579afc46-6df4-11ee-b962-0242ac120002" # ID of the
  "videoanalytics" service
microservice_affinity: "Node" id: b654f0a2-6d0e-11ee-b962-0242ac120002
```

```
geographical_affinity: "Edge"
region_id: "1"
instantiation_order: "4"

- microservice_id: "4c7e08c6-6df4-11ee-b962-0242ac120002" # ID of the "frontend"
  service
microservice_affinity: ""
geographical_affinity: "Central Cloud"
region_id: "2"
instantiation_order: "5"

service_mesh_integration: "Not applicable."

policy_and_compliance_management: "Complies with GDPR and HIPAA."

sla_and_policy_engine: "Integrates with internal policy engine."

rate_limiting_and_throttling_policies: "API rate limiting in place."

maintenance_and_update_cycles:
ci_cd_pipeline: ""
data_management: "Regular data cleaning and updates."
#####
#####
#####
```

```
#####  
  
network_traffic_and_load:  
-                                     microservice_interconnection:  
microservice_id:  
-                                     []  
-                                     connection:  
source:  
-  
destination:  
-  
protocol:  
-  
port:  
-  
connection_sla:  
latency:  
availability:  
error_rate:  
response_time_variability:  
bandwidth:  
  
-                                     network_sla: []  
#                                     Define SLA parameters  
#                                     Define links and their properties
```

```
- traffic_type:
# Define traffic type details.
load_balancing_strategy:
strategy: ""
techniques:
- # Define load balancing techniques.
details:
geographic_dns_routing: []
- time-based_routing:
# Define time-based routing details.
- rate_limiting_and_throttling:
rate_limiting: ""
throttling: ""
exceptions:
- # Define exceptions.
#####
#####
##### Surveillance System network traffic and load #####
#####
#####
network_traffic_and_load:
- microservice_interconnection:
microservice_id:
```



```

- "82bbcd33-b389-4c16-b312-0144bf4d00bc" # ID of the "database" service
- "b654f0a2-6d0e-11ee-b962-0242ac120002" # ID of the "IOTDevice-Camera" service
- "154fe329-5edf-4833-bbbb-668c0df2bcf0" # ID of the "backend" service
- "579afc46-6df4-11ee-b962-0242ac120002" # ID of the "videoanalytics" service
- "4c7e08c6-6df4-11ee-b962-0242ac120002" # ID of the "frontend" service

-                                     connection:
source:
-                                     154fe329-5edf-4833-bbbb-668c0df2bcf0
destination:
-                                     vip.82bbcd33-b389-4c16-b312-0144bf4d00bc
protocol:
-                                     TCP
port:
-                                     5432
connection_sla:
latency:           "Less than 500 ms"
availability:      "99.9% or higher"
error_rate:        "Less than 1% of requests (Very Low)"
response_time_variability: "Low"
bandwidth:         "High" # "1 Gbps"

-                                     connection:

```

```
source:
-                                     4c7e08c6-6df4-11ee-b962-0242ac120002
destination:
-                                     vip.579afc46-6df4-11ee-b962-0242ac120002
protocol:
-                                     TCP
-                                     connection:
source:
-                                     4c7e08c6-6df4-11ee-b962-0242ac120002
destination:
-                                     vip.154fe329-5edf-4833-bbbb-668c0df2bcf0
protocol:
-                                     TCP
port:
-                                     3000
connection_sla:
latency:          "Less than 500 ms"
availability:     "99.9% or higher"
error_rate:       "Less than 1% of requests (Very Low)"
response_time_variability: "Low"
bandwidth:        "High" # "1 Gbps"
-                                     connection:
```

```
source:
-
579afc46-6df4-11ee-b962-0242ac120002
destination:
-
vip.b654f0a2-6d0e-11ee-b962-0242ac120002
protocol:
-
RTSP
port:
-
554
connection_sla:
latency: "Less than 500 ms"
availability: "99.9% or higher"
error_rate: "Less than 1% of requests (Very Low)"
response_time_variability: "Low"
bandwidth: "High" # "1 Gbps"
-
connection:
source:
-
579afc46-6df4-11ee-b962-0242ac120002
destination:
-
vip.82bbcd33-b389-4c16-b312-0144bf4d00bc
protocol:
-
TCP
port:
-
5432
```

```

connection_sla:
latency:          "Less          than          500          ms"
availability:     "99.9%          or          higher"
error_rate:       "Less          than          1%          of          requests          (Very          Low)"
response_time_variability: "Low"
bandwidth:        "High"          #          "1          Gbps"

-                                     connection:
source:
-                                     579afc46-6df4-11ee-b962-0242ac120002
destination:
-                                     vip.154fe329-5edf-4833-bbbb-668c0df2bcf0
protocol:
-                                     TCP
port:
-                                     3000
connection_sla:
latency:          "Less          than          500          ms"
availability:     "99.9%          or          higher"
error_rate:       "Less          than          1%          of          requests          (Very          Low)"
response_time_variability: "Low"
bandwidth:        "High"          #          "1          Gbps"

- network_sla: # End to end network link SLA parameters

```

```
latency:          "Less than 500 ms"
availability:     "99.9% or higher"
error_rate:      "Less than 1% of requests (Very Low)"
response_time_variability: "Low"
bandwidth:       "High" # "1 Gbps"
# Define links and their properties
- traffic_type: "HTTP Requests, RTSP, websocket"
load_balancing_strategy:
strategy:       "Round Robin"
techniques:
- "Session-based routing"
details:
geographic_dns_routing:
- backend.154fe329-5edf-4833-bbbb-668c0df2bcf0.example.com
- frontend.4c7e08c6-6df4-11ee-b962-0242ac120002.example.com
- database.vip.82bbcd33-b389-4c16-b312-0144bf4d00bc.example.com
- time-based_routing:
"Off-peak hours traffic rerouted to backup servers."
- rate_limiting_and_throttling:
rate_limiting:  "100 requests per minute"
throttling:     "Up to 10,000 requests per hour"
exceptions:
```

```
- "Emergency access bypass for authorized users."
#####
#####

#####
#####

- security_and_access_control_policies:
authentication:
# Define authentication details.

method: ""
session_lifetime: ""
multi-factor_authentication: ""

authorization:
# Define authorization details.

role-based_access_control:
roles:
- # Define roles.

resource_access_levels:
- # Define resource access levels.

data_encryption:
in-transit: ""
at-rest: ""
```

```
firewall_policies:
#           Define           firewall           policies.

web_application_firewall:
ingress_egress_rules:           ""

ddos_protection:           ""

incident_response_plan:
standard:           ""
response_time:           ""
incident_classification:
-           #           Define           incident           classification.

compliance:
-           #           Define           compliance           details.
#####
#####
#####  Surveillance  System  security  and  access  control  policies
#####
#####
#####
security_and_access_control_policies:
authentication:
```

```
method: "Username and Password"
session_lifetime: "2 hours"
multi-factor_authentication: "Enabled for administrative users."

authorization:
role-based_access_control:
role_1:
- "Admin"
resource_access_levels_Admin:
- "User Management"
- "All Cameras"
- "Regional Service"
- "Query"

role_2:
- "Costumer"
resource_access_levels_Costumer:
- "Region Camera"
- "Query"

data_encryption:
in-transit: "TLS encryption"
at-rest: "AES-256 encryption for database storage."
```



```
firewall_policies: "Firewall rules implemented for IP whitelisting and network
segmentation."

web_application_firewall: "Not applicable."

ddos_protection: "DDoS protection service implemented."

monitoring_and_auditing:
metrics: "Response time and Error rate"
audit_trail: "Keeps a log of all API access."

incident_response_plan:
standard: "ISO 27001"
response_time: "Within 1 hour for critical incidents"
incident_classification:
- "Critical Incidents"
- "Major Incidents"

compliance:
- "GDPR"
- "HIPAA"

#####
#####

#####
#####
```