D2.3 Report on technological tools for CECC

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	D2.3. Report on technological tools for CECC		
Work Package	WP2		
Contractual due date	M12: 31st December 2023	**Actual submission date**	
Type	R- Document, report	**Dissemination Level**	PU – Public
Lead Beneficiary	EUR		
Responsible Author	Mohamed Mekki (EUR)		
Contributors	Mohamed Mekki (EUR), Sofiane Messaoudi (EUR), Ayoub Mokhtari (EUR), Adlen Ksentini (EUR), Souvik Sengupta (ION), Dimitrios Amaxilatis (SPA), Nikolaos Tsironis (SPA), Ioannis Zenginis (IQU), Josh Salomon (RHT), Ben Capper (RHT), Abdelhak KADOUMA (FIN), Wassim KRIBAA (FIN), Ibrahim AFOLABI (FIN), John Beredimas (CSG), Athanasios Kordelas (CSG), Elias Dritsas (ISI/ATH), I.Lakoumentas (ISI/ATH), D.Selis (ISI/ATH) and P.Marantis (ISI/ATH)		
Peer reviewer(s)	Gleibis Camejo Castillo (ARS) and Amadou Ba (IBM)		

Funded by the European Union

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

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
V1.0	13/09/2023	5%	Initial Deliverable Structure (ToC)	Mohamed Mekki (EUR)
V1.1	17/11/2023	70%	All tools description filled	Sofiane Messaoudi (EUR), Ayoub Mokhtari (EUR), Adlen Ksentini (EUR), Souvik Sengupta (ION), Dimitrios Amaxilatis (SPA), Nikolaos Tsironis (SPA), Ioannis Zenginis (IQU), Josh Salomon (RHT), Ben Capper (RHT), Abdelhak KADOUMA (FIN), Wassim KRIBAA (FIN), Ibrahim AFOLABI (FIN), John Beredimas (CSG), Athanasios Kordelas (CSG), Mohamed Mekki (EUR)
V1.2	24/11/2023	85%	Gap analysis	Mohamed Mekki (EUR)
V1.3	29/11/2023	90%	Prepared the initial full draft	Mohamed Mekki (EUR)
V1.4	05/12/2023	95%	Received internal reviewers' feedback	Gleibis Camejo Castillo (ARS), Amadou Ba (IBM)
V1.5	11/12/2023	99%	Addressing the review comments	Souvik Sengupta (ION), Dimitrios Amaxilatis (SPA), Nikolaos Tsironis (SPA), Ioannis Zenginis (IQU), Josh Salomon (RHT), Ben Capper (RHT), Abdelhak KADOUMA (FIN), Wassim KRIBAA (FIN), Ibrahim AFOLABI (FIN), John Beredimas (CSG), Athanasios Kordelas (CSG), Elias Dritsas (ISI/ATH), I.Lakoumentas (ISI/ATH), D.Selis (ISI/ATH) and P.Marantis (ISI/ATH), Mohamed Mekki (EUR)
V1.6	14/12/2023	100%	Final review	Mohamed Mekki (EUR), Adlen Ksentini (EUR), Christos Verikoukis (ATH/ISI)

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
AAS	Authentication & Authorization Service
AC³	Agile and Cognitive Cloud edge Continuum management
AppD	Application Descriptor
CA	Certificate Authority
CECC	Cloud Edge Computing Continuum
CECCM	Cloud Edge Computing Continuum Manager
CIK	Connector Instance Key
CIR	Container Image Registry
CLiSO	Cloud-native Lightweight Network Slice Orchestration
CNF	Container Network Function
CRUD	Create, Read, Update and Delete
DAPS	Dynamic Attribute Provisioning Service
DATs	Dynamic Attribute Tokens
DID	Distributed Identifier
DMZ	Demilitarized Zone
EDC	Eclipse Dataspace Components
GXFS	Gaia-X Federation Services
IaaS	Infrastructure as a Service
IDS	International Data Spaces
IDSA	International Data Spaces Association
IoT	Internet of Things
K8s	Kubernetes
LCM	Life Cycle Management
LMS	Local Management Systems
MEC	Multi-access Edge Computing
MOS	Management and Orchestration System
NFV	Network Function Virtualization
NOT	Notarization Service
NST	Network Slice Template

OCM	Organization Credential Manager
PaaS	Platform as a Service
ParIS	Participant Information Service
PCM	Personal Credential Manager
PKI	Public Key Infrastructure
PPL	Piveau Pipeline
SD-WAN	Software-Defined Wide Area Network
SLA	Service Level Agreement
SSI	Self-Sovereign Identity
SWRL	Semantic Web Rule Language
TRU	Trust Services
UID	Unique Identifier
WRL	Web Rule Language
XFSC	Eclipse Cross-Federation Services
OWL	Web Ontology Language

1. Executive Summary

The document is deliverable "D2.3: Report on technological tools for CECC" of the AC^3 (Agile and Cognitive Cloud edge Continuum Management) project funded under the Horizon Europe Research and Innovation Action programme. This deliverable is a report on the technological enablers envisioned to implement the Cloud Edge Computing Continuum Manager (CECCM) components as well as the different Local Management Systems (LMS) handling the Cloud Egde Computing Continuum infrastructure. Specifically, the document surveys the existing tools and technological enablers covering all the components of the AC^3 architecture (i.e., CECCM components and LMS), such as human machine interface, service orchestrators, data catalog and management platforms, Kubernetes, Openshift, LFEdge, Kubedge, SD-WAN enablers, Kubernetes Cluster management and interconnection tools. The deliverable also provides a classification of whether these tools are adapted to micro-service, cloud, edge, and far edge. It is worth mentioning that this deliverable corresponds with the preliminary outcome of task 2.3 titled "Technological tools for CECC".

The first high-level functional architecture of the AC^3 framework, as well as the CECCM components, were introduced and described in the deliverable D2.1. The envisioned architecture has been divided into 3 planes: the user plane, the management plane, and the infrastructure or CECC resource plane. The user plane includes the components that interact directly with the application developers. The management plane is the core function of the CECCM. It includes three key components: (i) Application and resource management that handles LCM of applications as well as orchestrates and manages the CECC infrastructure using AI/ML based solutions; (ii) Data management that manages access to cold and hot data following the Gaia-X procedures to access data spaces and IoT data sources federated with AC^3; (iii) the abstraction and federation layer needed to manage the federated infrastructure composed of computing resources (Cloud, Edge and Far edge). In AC^3, we envision a federated CECC infrastructure, which is composed of Public/Private cloud resources, edge resources, far-edge resources, and networking resources provided by different infrastructure providers.

Each Plane requires the implementation of different components and functional blocks that together allow the AC^3 framework to provide its services to the application developer and or the CECCM user while interacting with different LMS handling the CECC infrastructure. However, not all the building blocks of the framework need to be implemented from zero. Indeed, by reviewing the state of the art, there are several existing tools and technologies from partners' portfolios or well-known and used technologies that can be adapted and reused fully or partially to achieve the vision of CECCM.

To avoid any future integration issues, it is important to carefully select the technological tools to be used in the project and align the project's activities with these choices. Considering these facts, in this deliverable, the consortium surveys and presents the existing tools that can be used to implement the AC^3 framework. Those tools include open-source technologies and tools developed by members of the consortium.

Several technological tools, orchestration systems, and open-source solutions are listed in this deliverable. The listed tools are grouped according to the plane to which they belong. All tools related to the user plane are grouped in one section, including the components that interact directly with the application developers: (i) the Human Machine Interface that allows an application developer to interact with the CECCM to develop, deploy, and manage applications life cycle; (ii) The applications profiles, ontology modeling tools, and application descriptor models; (iii) the Service and Data Catalogue existing tools. Another section is dedicated to the tools that are needed at the management plane and the interaction with the infrastructure plane. Indeed, a first part is dedicated the management plane components, cloud-native application's Life Cycle Management systems (including service orchestration and monitoring), Data Management solutions, and related technologies. A second part is devoted to the tools used to interact with the infrastructure plane, wherein we focus mainly on the Local Management Systems (LMS), which are employed to manage and orchestrate the resources and networking. To recall, in AC^3, we do not contribute to the infrastructure plane; rather, we use the interfaces exposed by the LMS to interact with CECC infrastructure. This will allow the CECCM to use a federated infrastructure involving different stakeholders.

The deliverable concludes on the gaps that exist in the current tools such as the lack of AI-based solutions for application and resources management, and the challenges to implement seamless federation of resources, which will be considered and tackled in WPs 3 and 4.

The outcome of this document is the output of task 2.3 and will be considered as an input for work package 2, 3, 4 and 5.

2. Introduction

In today's rapidly evolving technological landscape, where the convergence of Cloud, Edge, and Far Edge technologies (known as Cloud Edge Computing Continuum - CCEC) is driving innovation, the Agile and Cognitive Cloud-edge Continuum management (AC^3) project stands at the forefront of this transformative wave. AC^3 is a pioneering European endeavor aiming to redefine the boundaries of contemporary computing paradigms and meet the demands of the modern digital age. CECC architecture is complex as it involves a high number of technologies and stakeholders which should cooperate to deploy an application over CECC. Indeed, deploying an application over the CECC infrastructure requires an agile and cognitive management system that abstracts this complexity to the application developer.

To alleviate this complexity, the AC^3 consortium has delivered in D2.1 a first high-level architecture that describes the key functional blocks of the CECC Manager (CECCM), which aims to deploy micro-service-based applications along with the requested data (hot or cold) on top of a Cloud Edge Computing Continuum infrastructure. The CECCM components and its internal functional blocks interact with applications' developers to ease the definition and Life Cycle Management (LCM) of micro-service-based applications, such as description, deployment, and monitoring of applications, ensuring optimal access to data hot or cold. Meanwhile, CECCM uses AI/ML algorithms to guarantee cognitive and agile management of application Life Cycles and optimal resource management. Finally, CECCM interacts with LMS to access CECC resources based on a federation of infrastructure providers. Deliverable D2.3, titled "Report on Technological Tools for Cloud Edge Computing Continuum (CECC)," emerges as a cornerstone as it aims to establish clear state-of-the-art technological tools that can be the basis of the CECCM components to avoid implementing the different components from scratch, while shedding light on missing gaps that need to be fulfilled in the concerned WPs (i.e., in WPs 3 and 4).

The AC^3 architecture is organized into three layers: User Plane, Management Plane, and Infrastructure or CECC plane. Each plane has objectives that need to be realized to handle the application LCM and manage the infrastructure efficiently using state-of-the-art AI/ML algorithms. The user plane functional blocks directly interact with the application developers, allowing them to execute the well-known CRUD operations, i.e., Create, Read, Update, and Delete applications. The management plane functional blocks enforce the CRUD operations, translating them to formats and operations understandable by the different LMSs. The last plane is the infrastructure plane, corresponding to the computing and networking resources. The two first planes, which are covered by the CECCM, and the interfaces between the management plane and the different LMSs, handling the CECC infrastructure, need to be devised by the AC3 project.

In this deliverable, the consortium's main objective is to navigate through the various existing technological tools that can be used to implement the functional blocks of the CECCM. We have mainly reviewed tools from the partners' portfolios as well as well-known tools such as Kubernetes. These tools are categorized and mapped to the AC^3 architecture functional blocks, such as Service Catalogue, Application Life Cycle Management, Data Management, and Local Management System. Having listed the existing tools and those that can be used to implement the functional blocks in WPs 3 and 4, we have analyzed the gaps that require adaptation and the development of new tools to best align with the AC^3 framework.

This deliverable will be a reference guideline for the WPs 3 and 4 during the development phase of the CECCM and will ease the integration process envisioned in WP5. Indeed, by establishing earlier a clear vision on the technological tools to use, the integration process will be smoother by reducing the usual integration issues.

2.1. Mapping AC^3 Outputs

The purpose of this section is to map AC^3 Grant Agreement commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to AC³ GA Deliverable & Tasks Descriptions

AC³ GA Component Title	AC³ GA Component Outline	Respective Document Chapter(s)	Justification
DELIVERABLE			
D2.3 Report on technological tools for CECC			
TASKS			
Task T2.3 Technological tools for CECC	**Task T2.3:** The aim of this task is to perform an extensive comparative study of the available technological tools that can be used to implement the different components of the CECCM and LMS in accordance with the predefined architecture model and target metrics.	Section 4, Section 5, Section 6	Presented the available technological tools for each layer of the AC³ Framework architecture.

2.2. 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 3 recalls the AC³ CECCM framework architecture.
- Section 4 presents the tools that are related to the User plane that includes the components that interact directly with the application developers: the Human Machine Interface that allows an application developer to interact with the CECCM to develop, deploy and manage applications life cycle. The applications profiles, ontology modeling tools and application descriptor models. In addition to the Service and Data Catalogue existing tools.
- Section 5 provides the tools that can be used for Infrastructure and Resources management. The tools include cloud-native application's Life Cycle Management systems, Data Management solutions and technologies, and Local Management Systems which correspond to the technology adopted by the infrastructure to manage and orchestrate the resources and networking.
- Finally, Section 6 concludes the gaps that need to be addressed by the project in order to implement the AC³ CECCM Framework.

3. Overall Architecture of the AC3 CECCM Framework

In this section we recall the high-level architecture adopted by AC³. The proposed architecture, illustrated in Figure 1, is composed of three planes: the user plane, the management plane, and the Infrastructure or CECC plane. Each plane has its own components and uses well-defined interfaces to communicate with the other planes. While the user and management plane are components of the CECC Manager (CECCM), the CECC plane corresponds to the infrastructure constituting the CECC, i.e., data source, computing nodes (central cloud, edge, and far edge). This architecture is detailed in deliverable 2.1.

The user plane includes the components that interact directly with the application developers. It includes the Application Gateway that allows an application developer to interact with the CECCM to develop, deploy, and manage applications' life cycle. The service catalogue includes a blueprint of the application's description, which can be extended or adapted by the application developer to create new applications to be deployed by the CECCM. The service catalogue also includes information on data sources, particularly cold data (i.e., data lake) federated using the Gaia-X approach. Finally, the last component of the user plane is the Ontology and Semantic aware Reasoner that translates and interprets all policies used by different CECCM actors (e.g., data source and application developers).

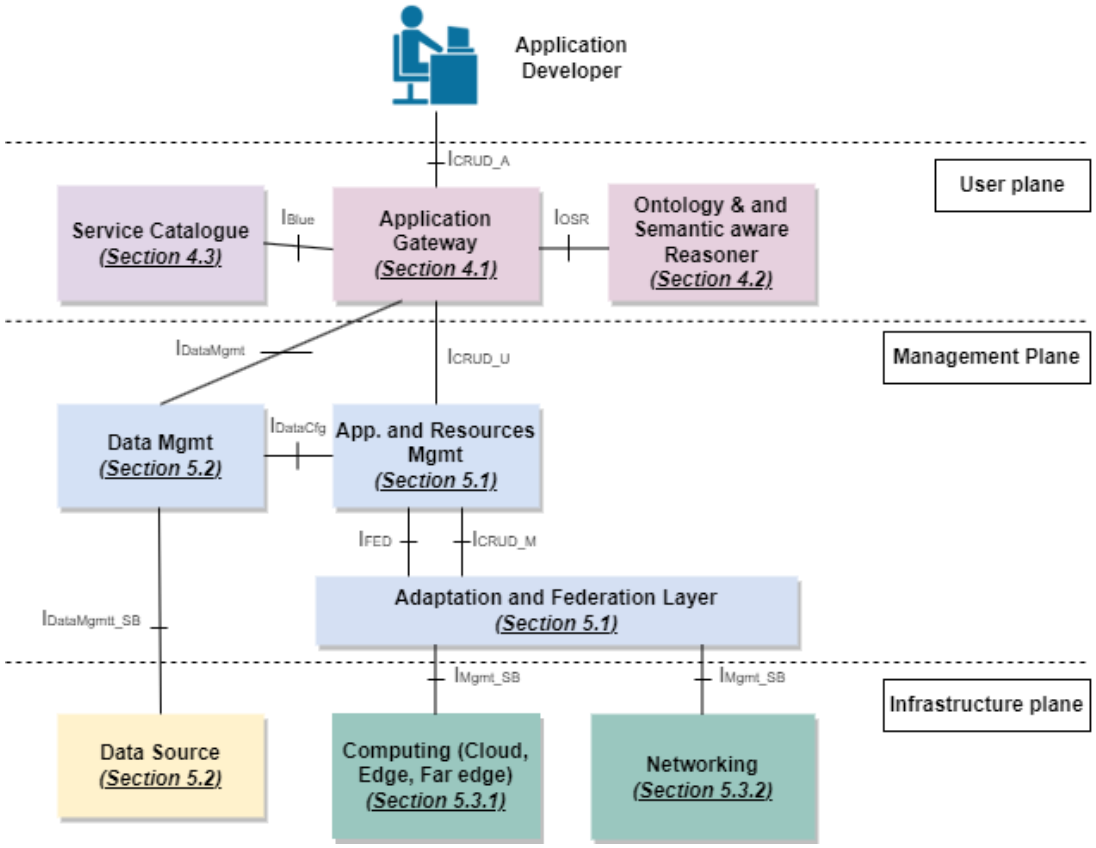

Figure 1. High-level architecture of AC³ and corresponding sections in the deliverable

The management plane is the core function of the CECCM. It includes three key components: application and resource management, in charge of the LCM of the applications and the CECC infrastructure management and orchestration. Data management that manages access to cold and hot data following the Gaia-X procedures to access data spaces and IoT data sources federated with AC³. The abstraction and federation layer, as in AC³, we envision that the owner of the CECCM may use federated resources (computing and networking) from different infrastructure providers, thus the abstraction and federation layer role is needed to allow resource discovery of the federated infrastructure and CRUD over the federated CECC infrastructure.

Finally, the Federated CECC infrastructure plane is composed of Public/Private cloud resources, edge resources, far-edge resources, and networking resources.

In the following sections, we present the candidate technological tools to implement the CECCM functionalities, divided into User Plane and Infrastructure and Resources management. First, the User Plane includes the Human Machine Interface that allows an application developer to interact with the CECCM. The service catalogue includes blueprints of the application's description and information on data sources. And the Ontology and Semantic aware Reasoner that is responsible for interpreting the policies of the different CECCM actors. Second, the Infrastructure and Resources management**,** in which we present the tools that can be used to implement the cloud-native application's life cycle management while supporting application and infrastructure monitoring. Data management describes the tools that allow the management of access to cold and hot data, in addition to procedures allowing to register, connect, and extract data from sensors owned by the application developer or a third-tier IoT provider. Local Management Systems (LMS) which correspond to the technology used to manage and orchestrate infrastructure's resources and networking.

4. User Interaction

This section reviews all tools related to the AC^3 user plane, including the components that interact directly with the application developers: (i) the Human Machine Interface that allows an application developer to interact with the CECCM to develop, deploy, and manage applications life cycle; (ii) The applications profiles, ontology modeling tools, and application descriptor models; (iii) the Service and Data Catalogue existing tools.

4.1. Human Machine Interface

4.1.1. EURECOM's Web portal

EURECOM hosts a facility that allows deploying cloud-native applications on top of a Cloud Edge Continuum infrastructure. It supports different wireless connectivity including 5G, Lora, etc. To ease the automatic deployment of a trial on top of the facility and collect key performance indicators (KPI), EURECOM developed a Graphical User Interface (GUI) in the form of a Web portal. The last one provides the trial owner a high-level view management interface to deploy, remove, and monitor a trial. All the trials in EURECOM facility runs as a slice [1], to guarantee multi-tenancy. To be enforced on top of the infrastructure, all the trial needs to be described using a Network Slice Template (NST) that includes information on the Cloud/Edge components, such as the application images, the needed computing resources for each application, such as CPU, memory, storage, and the location where to deploy the application. The NST also includes information on the wireless connectivity needed if IoT devices or 5G cellular devices need to be deployed. The NST is further divided into three parts: meta-data on the trial, wireless and network connectivity, and applications to run on the Cloud Edge Continuum. The applications and their needed resources are defined using the ETSI NFV standard descriptor, known as Network Service Descriptor (NSD) [2]. The NST format is based on JSON format. The web portal abstracts the creation of the NST to the application developer. The latter indicates using forms the needed information, and automatically the web portal generates the NST file. Then, the NST is sent to the management and orchestration system to deploy on top of the infrastructure the trial.

In, Figure 2, we illustrate the web portal architecture. It comprises a front-end, trial enforcement, life-cycle management, and KPI monitoring and presentation, as well as two databases (DB). All the components collaborate to ensure the trial's life-cycle, consisting of the definition and preparation, configuration and instantiation, run-time management, and deletion.

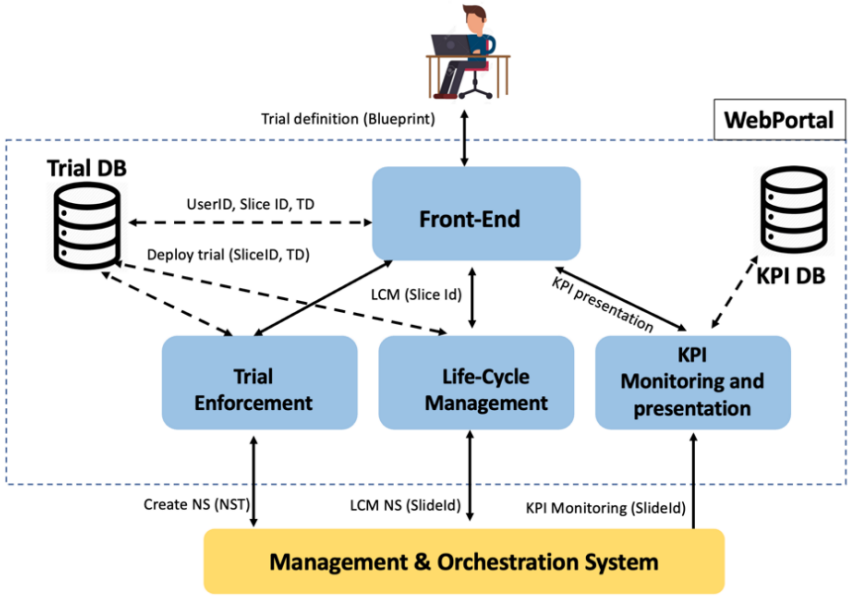

Figure 2. EURECOM's Web portal components

1) **Trial definition and preparation:** The trial definition and preparation are done by the trial owner using the front-end GUI, which corresponds to a web portal. This step consists of filling out a form that describes the trial scenario, the network resources, and the KPI to measure. Besides meta-data information on the trial, such as the start and end time of the trial, filled data concerns all the components of the network slice needed to run the trial. EURECOM facility allows the trial owner to specify the information on the needed cloud/edge resources, wireless connectivity, and application image location. Following ETSI NFV model, the applications are deployed as a network service in form of a monolithic or micro-service-based components. The network service can be composed of one or more applications, but also applications that can run on the client side, i.e., on the user device. The trial owner can deploy not only services at the edge but also on the device side to be able to test the server side of the application from a 5G or IoT device. To define a network service (i.e., a set of applications connected together to provide a service), the trial owner uses the GUI to create an NSD, which will contain one or more applications defined using the Application Descriptor (AppD) model of MEC ETSI [2] or Virtual Network Function Descriptor (VNFD) of NFV ETSI [3]. We extended the AppD with one field that indicates the type of deployment: edge or far edge (i.e., on the 5G or IoT device). Again, the trial owner can use the GUI to fill the AppD field, simplifying the configuration process. Consequently, the trial owner does not need to know about the NSD and AppDs formats. Indeed, the trial owner has to fill the form, and automatically the NSD with all AppDs is generated. At this step, the trial owner needs to provide information, such as the location of the application image(s) to deploy both on the edge and far edge, the amount of CPU as well as memory to assign to the application. Finally, the trial owner can select, from a list, the KPI to monitor. As output, the front-end produces a Trial Descriptor (TD) that contains all the information entered by the trial owner, i.e., meta-data, wireless and networking information, NSD with the list of AppD, and the KPI to monitor. New information is added to the meta-data part, which is the trial Identifier (ID) generated by the front end. The trial ID is used to identify the trial, as a vertical may run several trials of the same scenario but with different resource configurations. The TD is stored, along with other information (like the vertical ID), in the Trial DB.
2) **Configuration and instantiation:** This phase starts after the generation of the TD by the front-end module. Once the TD is stored in the Trial DB, the Trial enforcement is called. The Trial enforcement translates the TD to an NST and uses the Northbound Interface (NBI) of the Management and Orchestration System (MOS), to first request the configuration of the Network Slice and check the resource availability. Once the request is accepted by the MOS (a Slice ID is created and sent back), the Trial enforcement requests the instantiation of the Network Slice using the returned Slice ID. When the MOS confirms the instantiation of the network slice, meaning that the trial can start, the Trial enforcement updates the status of the trial to “running” in the DB and informs the front end that the trial can start. The front end displays this information on the GUI (i.e., a web page), showing the trial’s status, and allowing the vertical to start the monitoring process. Once done, the front end forwards the request, including the Slice ID and KPI list (obtained from the Trial DB), to the KPI monitoring and presentation module. The latter sends the request to the MOS along with the Slice ID and KPI list. The MOS replies with two URLs; the first is to access the dashboard to visualize KPI in real-time, and the second is to subscribe to a broker to access the data stream representing the KPI in raw data form. The latter will allow the vertical to store data on the trials for future usages, such as training Machine Learning (ML) models. Then, the KPI monitoring, and presentation module creates an entry in the KPI DB, where the Slice ID and the corresponding URLs are stored. Then, it forwards the URLs to the web portal, which displays them to the trial owner. The latter can decide to use only the dashboard, subscribe to the broker’s URL to obtain the raw data, or use both.
3) **Run-time management:** The front end allows the vertical to update the assigned computing and network resources to a running trial. Again, this can be done through the web portal, where the vertical selects the resource type. Two possibilities are given to the vertical, update the Radio Access Network resource by requesting more or fewer radio resources; and update the computing resources

of a running application. For each running application at the edge, the vertical can specify new values for CPU and memory. The web portal redirects the request, with the Slice ID, to the Life-cycle management module. The latter uses the NBI of the MOS to update the network slice resource. The MOS confirms or rejects the update if there are not enough resources, and hence the vertical is informed about the status of the request. The vertical can also resume a trial and restart it. The resume step consists in sending a request to the MOS through the life-cycle module to stop the slice without deleting its associated computing and network resources. The MOS also postpones the collection of KPI. The restart procedure consists in instantiating the network slice again. When resumed, the trial status in the Trial DB is updated accordingly.

4) **Deletion:** The vertical, when deemed appropriate, can manually stop and delete the trial before the end via the web portal. Then, the request is sent to the MOS via the lifecycle module. Unlike the resume case, the MOS will stop the slice and remove all the resources dedicated to it. The virtual image of the applications is off boarded from the computing infrastructure (NFVI). The trial DB is updated by removing the Slice ID corresponding to a trial. The front end proposes to the vertical if the TD should be stored for future use, for instance, as a Blueprint. If the vertical accepts, the TD is not removed from the DB. It will be proposed as a Blueprint to create another trial
5) **Monitoring:** Monitoring the performances of the different components is a critical process when testing a network service. Indeed, the vertical needs to extract useful information regarding the behavior of its applications from the infrastructure point of view. While the vertical can easily extract service level KPI, it can be very pertinent to combine them with infrastructure KPI to build root cause analysis and improve the performance of its applications. We grouped the collected KPI into three groups: one on the RAN (such as latency, uplink, and downlink data rate), one on the edge cloud (such as CPU and memory usage as well as data rate), and finally, one on the network slice level (such as the time needed to deploy and decommission a network slice).

It should be noted that EURECOM's Web portal can be used to cover the Application Gateway functionalities. Of course, adaptations are needed as AC^3 will use OSR model to deploy the micro-service-based application.

4.1.2. MAESTRO Service Orchestrator – Front-end user/developer interface

The MAESTRO platform end-user interface serves as the entry point of potential platform stakeholders to onboard and manage their application over targeted infrastructures. This front-end environment provides all the required entry fields for the onboarding of application components as well as the space for the visualization of monitoring parameters, provided both by the network orchestrator and internally from the management of the applications' lifecycle.

The end user (application developer or vertical end-user or manager) is required initially to login successfully. This activates the corresponding profile for each user type.

In the first step, the end-user can identify the necessary components from the related repository or enter new components. Each application consists of a number of components that must be chained together in order to provide the required functionality. Also, for each component, the developer should have the capability to define certain networking and functional parameters based on which the network orchestrator will provide the required placement in terms of location and resources. Therefore, for each new component that is introduced there is a list of parameters that are defined such as mobility, and CPU or GPU, RAM and storage needs. In addition, the interface provides the capability to the components to scale (in terms of required resources) as the workload increases or decreases, thus adding to the dynamicity of the platform tool, while ensuring the resource optimization.

The next step in the interface allows the developers to construct the application graph by logically connecting the application components among them and providing the link characteristics (e.g., delay, bandwidth, communication ports etc.), as well as any locality aspects (e.g., components to run at specific edge/site).

The MAESTRO dashboard can include information from a variety of applications or node parameters such as deployment of application components and the live usage of resources through related resource metrics.

(Beyond the dashboard visualization, such data can be used also for further processing and extraction of prediction models or in general behavioral aspects of the application. In addition, the interface can be used to provide network and node state alarms received by the network orchestrator or any other link related metrics that can be fed back to MAESTRO.

Finally, an important role of the operators' Interface is to act as the entry point and activator of the policies, as well as the monitoring of already activated polices, which in turn manages the dynamic scaling of applications.

4.2. Application / Ontology Descriptor

4.2.1. Ontology and applications profiles Description

Application profiles provide detailed descriptions of an application's features, requirements, and specifications. They serve as blueprints for software development, highlighting an application's functional and technical requirements and providing a clear understanding of its purpose and capabilities. In software development, application profiles are used to guide the development, deployment, and monitoring of an application throughout its lifecycle.

Application profiles are crucial in the context of CECC applications due to the complexity of these applications. Often, these applications are distributed across several nodes, each of which may be in a different location and use a different hardware platform.

The key elements of an application profile descriptor in the CECC context are: **Application name and description, Application Consumers, Microservices, Data sources, Load and traffic types, Deployment context**

Ontology is a formal specification of a conceptualization that aims to represent the knowledge of a specific domain of interest. It establishes a set of concepts and categories, as well as their properties and relations, and provides a shared vocabulary for describing and reasoning about a domain.

Ontology editors provide graphical interfaces to model ontologies by defining classes, properties, relationships, and restrictions. They aim to simplify the ontology development process.

- Two prominent open-source ontology editors are Protégé and WebProtégé from Stanford University.
    - **Protégé** [4] offers a desktop application with an intuitive GUI and integration with reasoners for modeling OWL (Web Ontology Language) ontologies.
    - **WebProtégé** [5] provides a lightweight web-based environment optimized for multi-user collaboration.
- Both editors support standard ontology languages like OWL 2 and can be leveraged to develop the application profile ontology through visual interfaces rather than manual coding.

In addition to dedicated ontology editors, there are also frameworks and libraries to manage ontologies programmatically within applications.

- **Apache Jena** [6] is an open-source Java framework that provides an API for building semantic web and linked data apps. It has capabilities for loading, storing, querying, and inferencing over ontologies using code.
- **OntoStudio** [7] is a commercial ontology modeling suite with features like graphical editing and integrated reasoners. It also offers APIs and libraries for accessing ontologies from external applications.

Tools like Jena and OntoStudio allow tightly coupling the application logic with the ontology by manipulating it directly through code. This facilitates tasks like dynamically querying the ontology to classify applications based on their profiles.

Semantic reasoning plays a crucial role in leveraging ontologies for a variety of applications, including but not limited to CECC and microservices. A semantic reasoner is a software tool or system that uses formal logic

and semantic knowledge to infer new information from existing knowledge. It is designed to process and analyze structured data, such as ontologies, taxonomies, or knowledge graphs, in order to draw logical conclusions and make intelligent inferences.

There are several semantic reasoners available that could be utilized to infer new knowledge from an ontology.

- **Pellet** [8] is an open-source Java library for OWL reasoning. It provides standard reasoning capabilities like classification, consistency checking, and realization. Pellet could be used to validate constraints in an ontology and classify applications based on their profiles.
- **HermiT** [9] is an open-source reasoner powered by hypertableau calculus algorithms. It can efficiently classify ontologies and detect inconsistencies. HermiT could help validate an ontology and ensure logical consistency.
- **Fact++** [10] is an open-source forward chaining reasoner built in C++. It utilizes a RETE engine for optimized incremental inference. Fact++ would be applicable to derive new knowledge from large sets of facts in the ontology.
- **RacerPro** [11] is a commercial reasoner capable of handling complex ontologies with very expressive description logics. It provides both TBox and ABox reasoning. RacerPro could be leveraged if an ontology uses advanced OWL 2 constructs beyond the basic profiles.

Reasoning rules are a central aspect of semantic reasoning systems. They allow expressing logical relationships between different entities or concepts and can be used to infer new information from existing information. Rules are often used to express logical inferences that cannot be directly expressed using the ontology primitives. The rules are processed by the semantic reasoner, which uses them to perform logical inferences and deduce new information from the ontology. This process may involve applying reasoning techniques such as forward chaining and backward chaining, as well as checking logical consistency and resolving contradictions.

Various rule languages have been developed to express the rules used by semantic reasoners in the context of ontology languages.

- **The Semantic Web Rule Language (SWRL)** [12] is an extension of OWL, one of the main languages for representing information on the Semantic Web. SWRL adds the ability to specify logical rules on data defined in OWL. SWRL is fundamentally based on a subset of the first-order logic language, which means it can express fairly complex relationships between entities in an ontology. SWRL makes it possible to specify rules that can be used to make inferences about data.
- **Rule Interchange Format (RIF)** [13] is a standardized rules language developed by the World Wide Web Consortium (W3C). Its main objective is to promote the exchange of rules between different systems, and to enable interoperability between various rule formats.
- **Web Rule Language (WRL)** is another form of rule language used in the context of the Semantic Web. WRL is a subset of the RuleML rule language and is mainly used to express complex rules in the context of semantic web ontologies. WRL makes it possible to combine OWL ontologies with logical rules, offering richer expressiveness.

4.2.2. EURECOM's Network Service Descriptor

EURECOM's NST is used to deploy a trial on top of its Cloud Edge Continuum facility (as described in section 4.1.1.1). The NSD is used to describe the application deployment to automate the LCM of the applications. Each application or a component of the application (i.e., micro service) is described using a Virtual Network Function Descriptor (VNFD), if it needs to be deployed at the centralized cloud or AppD if the component needs to be deployed at the edge. Both descriptors rely on the ETSI NFV and MEC specifications.

Both the VNFD and AppD include information needed to automate the deployment:

- Meta-data: it describes data added by the Management and Orchestration, such as the Id of the NSD, the owner Id, and the version.
- URL: URL indicating the image location on the Internet. It can be a public directory, such as Docker Hub, or a private directory. The URL can also indicate a Github link if the image needs to be built.
- Needed computing resources: CPU, Memory, and storage.
- Algorithms to scale up or down resources.
- Configuration parameters: These parameters are needed when deploying a container-based application or micro-service image.
- Location: Location where to deploy the applications. It can be a region Id or Infrastructure identifier (cloud, edge, or far edge)
- Traffic redirection: This is needed for AppD to indicate the type of traffic that needs to be redirected to the edge application.

4.2.3. Languages to describe applications

JavaScript Object Notation (JSON) is a lightweight data interchange format. For computers, it is easy to generate and parse. For humans, it is easy to read and write thanks to its simple syntax and treelike structure. It enables representing structured data. In the context of the project, specifically for modelling application profiles, we opted for the format for several reasons:

- JSON is a textual data format that is easy to read and write for both humans and machines. This simplifies the development and maintenance of application profile models.
- JSON is a widely accepted standard supported by many technologies and programming languages including JavaScript, Python, Java, and many others. This facilitates integration with different system components.
- Although simple, JSON is flexible enough to represent more complex data structures if needed.

YAML is a human-readable data serialization language that is often used to code configuration files. For some, YAML stands for Yet Another Markup Language, for others it is the recursive acronym YAML Ain't Markup Language, which stresses that YAML is used to represent data rather than documents. We chose to use the YAML format to define the deployment and monitoring policies for our project for several key reasons:

- One of the most significant advantages of using YAML is its native compatibility with deployment tools. YAML is used to define and manage services, networks, and volumes for different services.
- YAML is designed to be easily readable by a human. Its structure is simple and intuitive, making the configuration file transparent and easy to understand.
- YAML allows complex data structures through its arrays, dictionaries, and scalar types. This flexibility allows us to create highly flexible deployment and monitoring policies.

OWL (Web Ontology Language) is a semantic markup language used to represent ontologies and knowledge graphs. In our project, we leverage OWL to formally define the concepts, properties, and relationships of our application profile ontology.

We chose OWL because it provides greater machine interpretability than other ontology languages. OWL has a well-defined semantics grounded in description logic, allowing automated reasoners to infer additional knowledge from the asserted facts. This capability is useful for validating the ontology, ensuring consistency, and deriving implicit relationships.

The application profile ontology represents the key characteristics and requirements of microservice-based applications. By using OWL to model the ontology, we can leverage semantic reasoners to classify applications based on their profiles, detect inconsistencies, query for specific criteria, and more.

4.3. Service and data Catalogue

4.3.1. Service Catalogue

A service catalogue includes information about the services that are deployed. It may be extended and include also generic information on services that are onboarded, but not initiated (i.e. service repository) Such information may vary depending on the implementation.

For AC3 the goal is to maintain the initial service deployment request and the information related to the end-user requirements (policies) introduced per deployment request. These can be extended also to data related requests in which case an interconnection with data catalogue is required. At this level there is no need to include live service updates since these are handled in an automated manner through the AI-based application and resource management block.

Related service catalogue implementations have been developed for the MAESTRO orchestrator solution. In its earlier version the catalogue is a simple repository of the registered services that includes the user-declared onboarding parameters (bandwidth, latency, component dependencies) and the image locations. The developments towards the latest version includes also the integration of resource catalogue information for the interrelation with potential deployment locations as we a dynamic service update part to accommodate live service deployment updates during the service lifecycle.

4.3.2. Data Catalogue

The establishment of data spaces is instrumental in facilitating a secure, interoperable, and sovereign data exchange ecosystem within the AC³ CECC framework. A critical element within such data spaces is the data catalogue, which serves a pivotal function in both the genesis and the maintenance of data spaces. It lays the groundwork for data identification, retrieval, and regulatory compliance across these computational continua. The data catalogue functions as an integral conduit between data providers and consumers, ensuring that the extensive data repositories within the data spaces are not only accessible and comprehensible but also subject to effective oversight. In this document, we aim to furnish a succinct exposition of several of the most distinguished data catalogues presently in existence.

4.3.2.1. IDS Metadata Broker

The International Data Spaces (IDS) framework lays the groundwork for the evolution of a worldwide digital economy by merging a technical infrastructure with governance models that support the secure, standardized, and facile integration of data within data spaces. This standard champions data sovereignty, empowering organizations and individuals with control over the usage parameters of their data within the value chain, such as the conditions, timing, and pricing. This autonomy is pivotal for fostering novel, intelligent services and facilitating ground-breaking inter-organizational business procedures.
The development of the data spaces ecosystem is predicated on the orchestration of several key components as per the IDS framework:

- Identity Provider: Establishes the digital identity of participants.
- IDS Connector: Serves as the gateway for engaging in data exchange.
- App Store and Data Apps: Provide applications for data usage and management.
- Metadata Broker: Facilitates the organization and retrieval of data descriptions.
- Clearing House: Ensures compliance and records transactions.
- Vocabulary Hub: Harmonizes the terminologies used within the ecosystem.

An expansive description of the functionalities of these components is available in the IDS Whitepaper [14]. Participants looking to engage in the ecosystem must first authenticate their identity via the Identity Provider and then connect through the IDS Connector. This initiates the process of data exchange among the various stakeholders in the data space environment. Figure 3 illustrates the interplay between different IDS components, delineating the establishment of data spaces and the mechanics of data sharing among the ecosystem's participants.

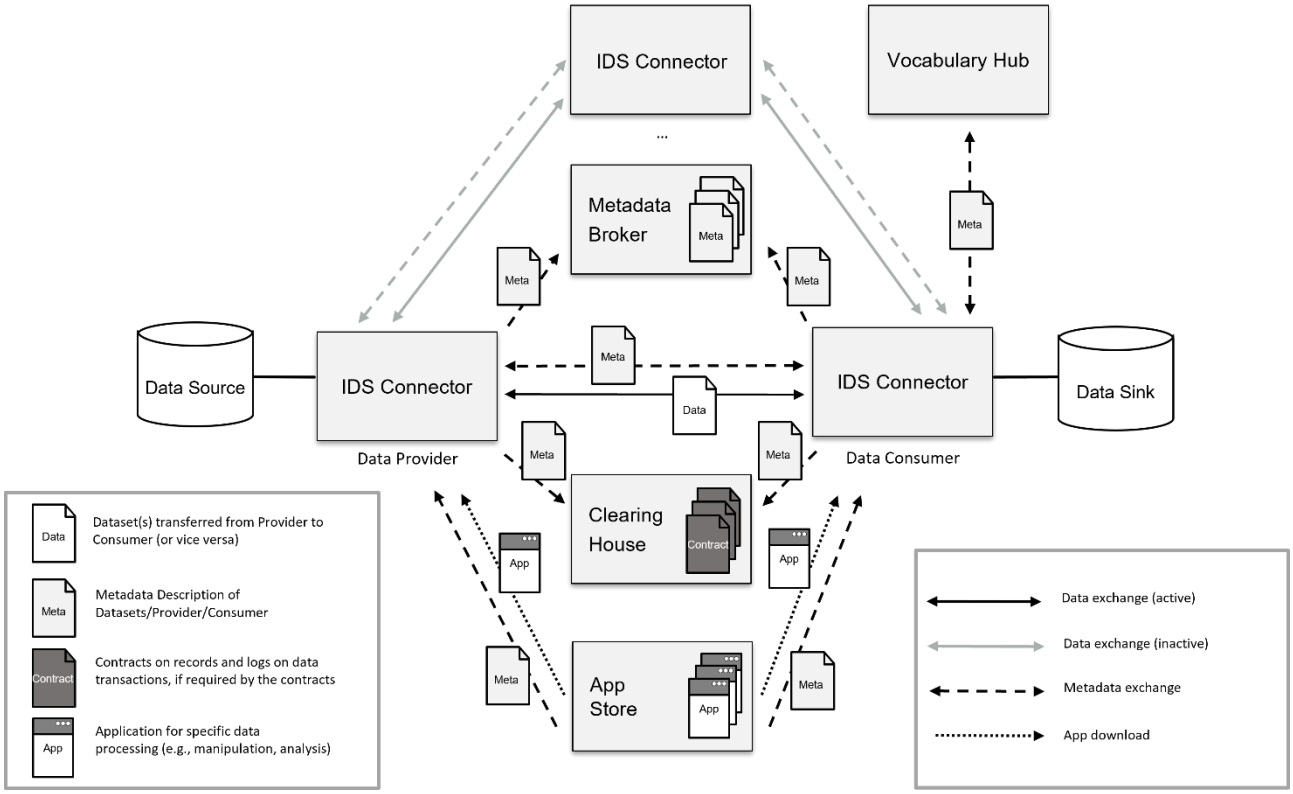

Figure 3. Interaction between different IDS components for creating Data Spaces

The Metadata Broker, as delineated by the International Data Spaces Association (IDSA) reference architecture, is the component that most closely embodies the functionalities of a data catalogue. Its pivotal function is to maintain interoperability among data space entities through the utilization of uniform metadata schemas. Additionally, it underpins data sovereignty by cataloguing critical information regarding data governance, usage protocols, and proprietorship, thereby enabling stakeholders to retain authority over their data repositories. Although individual data providers are responsible for administering their proprietary data catalogues, the Metadata Broker orchestrates a cohesive, federated framework, permitting queries to be conducted across the collective data space network. The Broker parallels the traditional data catalogue by acting as the nexus for stakeholders to locate and apply data within the IDS environment. Furthermore, elements such as the IDS Connector also parallel certain aspects of a data catalogue by allowing data providers to delineate and implement data utilization policies; however, they do not function as catalogues in their own right. Collectively, the suite of IDS components synergizes to manifest a comprehensive virtual data space. This integrated system facilitates the registration, discovery, and procurement of metadata about various data sources, all within the ambit of ensuring data sovereignty and fortified data transactions.

4.3.2.2. XFSC (Former GXFS) catalogue

The Gaia-X initiative in Europe is pioneering a framework through its Gaia-X Federation Services (GXFS) aimed at fostering a competitive digital ecosystem, one that offers a European alternative to the dominant, often non-European cloud service providers. This endeavor is not only focused on reinforcing Europe's digital autonomy but also on stimulating a vibrant market for innovation. Gaia-X envisions data spaces as secure and compliant networks in accordance with European standards, where data and services are exchanged within a federated, open data infrastructure. A central objective of Gaia-X is the assurance of data sharing and sovereignty, which includes establishing standards for data spaces. Within this framework, a data catalogue serves as a vital index, detailing information on the data and services accessible in the Gaia-X

ecosystem. Specifically, this catalogue is a repository of metadata regarding various data sets, guiding users on data availability, access modalities, and the terms of use. The Gaia-X Federation Services (GXFS) integrates into the wider Gaia-X architecture, providing capabilities that support a federated data infrastructure. Among these capabilities are federated catalogues, which list datasets and services, enhancing their discoverability and interoperability. Within the GXFS catalogue are foundational features that enable the identification of Gaia-X resources, assets, and participants by potential users. Entities within GAIA-X are characterized by Self-Descriptions, which, when intended for public use, are inputted into the relevant catalogue. The catalogue's purpose is to facilitate the connection between consumers and the most suitable offerings and to track important updates to those offerings. The catalogue stores numerous Self-Descriptions, both as raw JSON-LD files and as part of a Self-Description Graph, enabling sophisticated queries through interlinked Self-Descriptions. The GXFS Catalogue is designed as a distributed system, composed of multiple components to leverage existing technologies and allow scalability. These components, detailed at official website [15], can be deployed independently and include:

- Catalogue: The primary module, providing essential catalogue functions.
- Authentication: An external module managing authentication and user profiles.
- Graph-DB: A graph database that holds all assertions from active Self-Descriptions and executes semantic search operations.
- File Store: A binary storage system, which archives the Self-Description files and schema files, including their historical iterations.
- Metadata Store: A repository for the metadata pertaining to the Self-Descriptions and Schemas in the File Store.

Currently, the Gaia-X Federation Services (GXFS) project has been moved under the governance of the Eclipse Foundation and the project is going to continue evolving under the name of XFSC with full open-source governance. For more details and to get assistance for deployment, one can visit the Eclipse Foundation's official GitLab repository [16] or follow the Software requirements specification for GXFS [17]. Following the Gaia-X guidelines, one of the consortium members – IONOS has already developed a deployment script to run the Catalogue over the IONOS Cloud infrastructure, which is available to the IONOS's official GitHub repository [18].

4.3.2.3. EDC Catalogue

The Eclipse Foundation is instrumental in fostering the development of technologies pivotal for constructing data spaces and ancillary components, such as data catalogues. These technological solutions furnish the requisite framework to efficiently administer, catalogue, and regulate data accessibility in a manner that is secure, compatible, and scalable. Initiatives like the Eclipse Dataspace Connector align with the broader ambition to establish secure and autonomous data spaces, providing essential elements for the safe and regulated exchange of data across diverse data domains. The Eclipse Dataspace Connector exemplifies a platform offering the components necessary for the secure and governed sharing of data among various data spaces. In the context of the Eclipse Foundation's projects, a data catalogue is designed to act as a repository, offering a searchable inventory of datasets and services for user and application discovery. It retains metadata concerning data assets to aid in data identification and comprehension, enforces data governance standards to ensure compliance with designated access and sharing policies, and promotes interoperability among disparate data providers and consumers through adherence to shared standards and protocols. Figure 4 conceived by the Eclipse Foundation, depicts a functional schematic of data spaces. It illustrates how data space connectors, in collaboration with data catalogues, function as logical stewards to integrate ecosystem participants and facilitate sovereign data exchange. Most of the related development are open-source and can be found in official GitHub repositories [19] of the Eclipse Foundation.

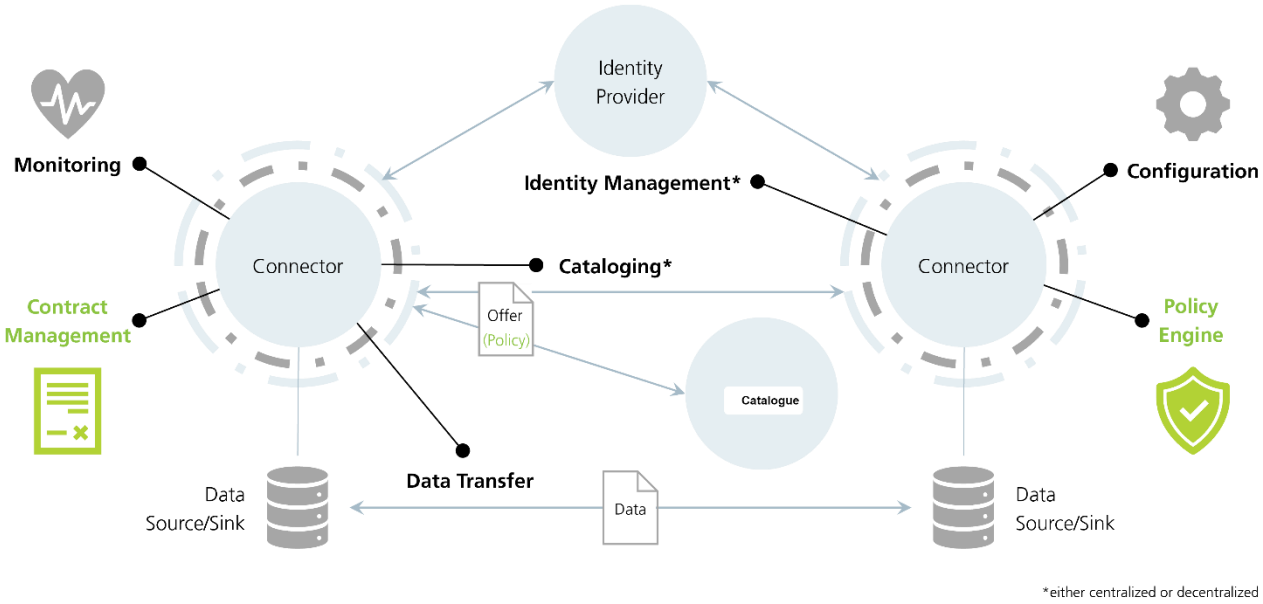

Figure 4. A conceptual diagram of the creation of Data Spaces

4.3.2.4. Piveau Catalogue

Besides IDS, XFSC, and EDC Catalogues Piveau [20] has also offered an open-source metadata catalogue solution for managing their data management platform for the public sector. Piveau harnesses Semantic Web technologies, aligning with the World Wide Web Consortium's (W3C) Data Catalog Vocabulary (DCAT) standard and the European DCAT-AP (DCAT Application profile for data portals in Europe) [21] specification for Open Data. It bridges the divide between the theoretical metadata frameworks and their practical implementation in live environments. With a focus on Open Data, Piveau stands at the forefront of providing solutions for public entities and nonprofit organizations to disseminate metadata catalogues that are both interoperable and adaptable. The data management system proposed by Piveau is constructed upon a microservices architecture, supplemented by a bespoke pipeline system, to enable a modular and extensible assembly of features. Within its functional architecture, the Piveau hub is the pivotal repository for data storage and registration. In contrast, Piveau Consus undertakes the role of data ingestion, engaging in the collection, scheduling, transformation, and standardization of data from a myriad of sources. Additionally, the Piveau metrics module is tasked with the generation and upkeep of detailed quality metrics, which are then integrated back into the Hub. A critical facet of this architecture is the Piveau pipeline (PPL), which represents the sequential data processing workflow, defined by a straightforward JSON document enumerating various processing segments. For more details one can follow the Piveau official documentation [22].

5. Infrastructure and Resources Management

This section is dedicated to the tools that are needed in the management plane and the interaction with the infrastructure plane. The section is divided into two parts. The first one covers the tools dedicated to the management plane components, cloud-native application's Life Cycle Management systems (including service orchestration and monitoring), Data Management solutions, and related technologies. The second part is dedicated to the tools used to interact with the infrastructure plane, wherein we focus mainly on the Local Management Systems (LMS), which are employed to manage and orchestrate the resources and networking. To recall, in AC^3, we don't contribute to the infrastructure plane; rather, we use the interfaces exposed by the LMS to interact with CECC infrastructure. This will allow the CECCM to use a federated infrastructure involving different stakeholders.

5.1. Application Life Cycle Management

5.1.1. Orchestration

5.1.1.1. CLiSO Framework

CLiSO is an end-to-end Cloud-native Lightweight Network Slice Orchestration framework capable of orchestrating Container Network Functions (CNF) that need to be deployed on top of a container-based infrastructure managed by Kubernetes and Openshift. CLiSO is deployed in the EURECOM facility as a Management and Orchestration System (MOS) to deploy container-based applications or micro-services on a Cloud Edge Continuum infrastructure. CLiSO can deploy on public, private, and hybrid cloud, which also allows dynamic management of infrastructure.

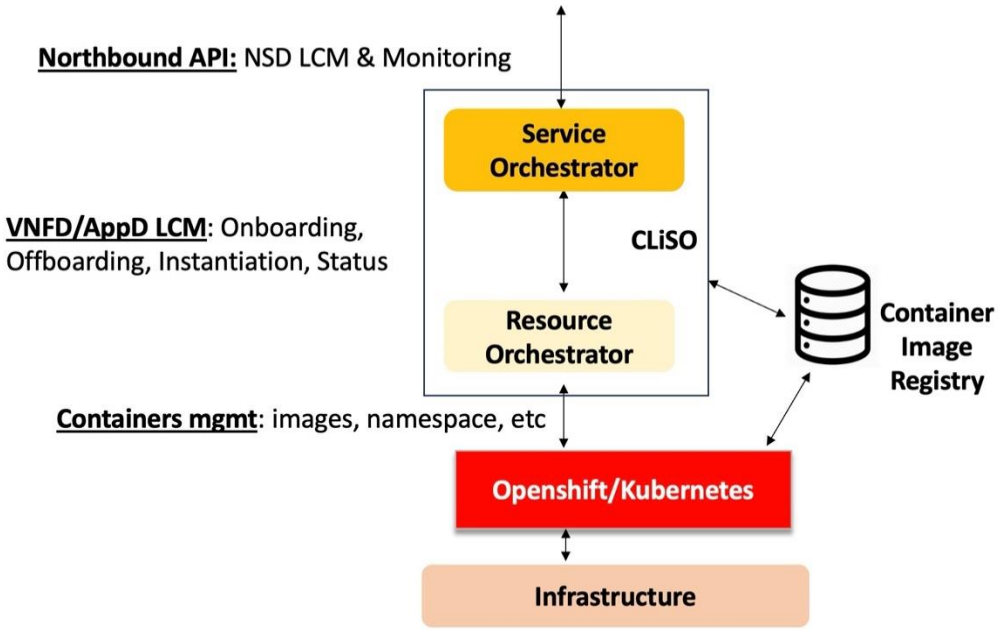

Figure 5. Cloud-native Lightweight Slice Orchestration Framework

As illustrated in Figure 5, CLiSO is composed of two layers the Service Orchestration Layer (SOL), which is responsible for translating Service Level Objects (i.e., AppD/VNFD) to Resource Level Objects (RLOs) (i.e., containers coordinating with resource orchestrator. Whereas Resource Orchestration Layer (ROL) manages the resources by communicating with the underlying resource pool managed by Local Management Systems (LMS) according to AC^3 terminology. CLiSO interacts with different LMS using the notion of plug-in that uses the LMS NBI. In terms of LMS, CLiSO supports Kubernetes, OpenShift, and K3S. CLiSO exposes Northbound API to handle NSD LCM composed of a set of applications or micro-services described using AppD or VNFD. CLiSO uses a Container Image Registry (CIR) to store containers downloaded from the Internet (using URL

link included in AppD and VNFD) to be deployed on the infrastructure. CLiSO supports dynamic resource management of resource pool. Indeed, CLiSO allows the registration of LMS in an asynchronous way. In AC^3, CLiSO can be used at the management level to deploy micro-service-based applications on top of the Cloud Edge Continuum infrastructure.

5.1.1.2. MAESTRO Service Orchestrator – Application and resource management

The core part of the MAESTRO Orchestrator framework provides service orchestration for containerised applications, registered and onboarded in the form of Application Function Chains (AFCs) and includes two main functionalities: a) the service deployment cycle and b) the runtime service and resource update cycle. The platform is integrated with vanilla Kubernetes APIs and supports service-level telemetry using Prometheus & NetData, extendable to other monitoring engines (e.g. from network resources or Kubernetes platform). The current design includes the ability to define SLAs during service order and apply policy related criteria through a runtime decision engine which can be extended to include AI based decisions. A schematic of the MAESTRO framework is presented in Figure 6 and is further explained in the following paragraphs.

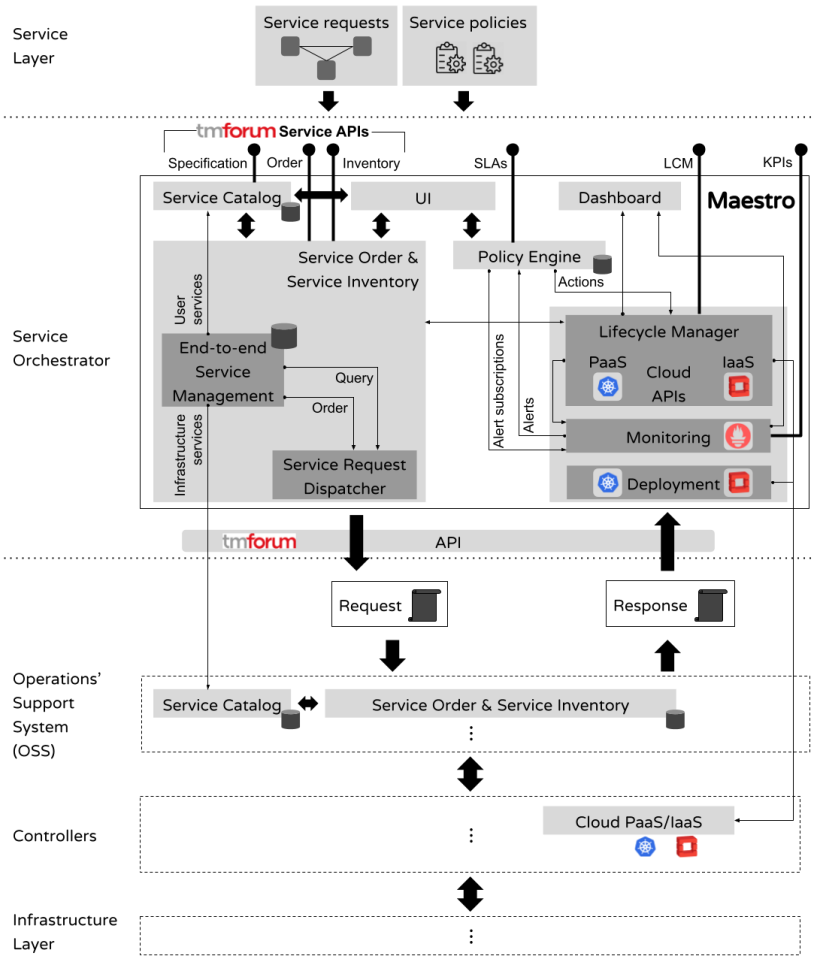

Figure 6. MAESTRO service orchestration framework

The MAESTRO deployment cycle phase: The deployment process of the MAESTRO is handled through the service order and inventory block. The service layer requests and policies formulate a certain service order using the related TMF standards. An e2e service manager promotes the order to a service request dispatcher the interfaces with the underlay infrastructures. The process may include an initiation phase through which certain infrastructure details may be queried and collected, exposing details (e.g. topology information) to the service orchestration for formulating the requests. Once the request is processed the available resources

are allocated and the connection information is returned to the deployment module of the service orchestrator for deploying the images and initiate the service. The underlay concept promotes the logical separation and management of the two layers allowing applications to be defined at high level with specific SLAs and requirements (i.e. in an end-user friendly format) and then translated through the service onboarding API to service deployment intents that are handled at the network orchestration level.

The MAESTRO Runtime management phase: Once the service has been initiated, the lifecycle manager is responsible for the configurations during runtime. The overall mechanism includes a monitoring module and a policy engine that extracts decisions for service reconfiguration based on the interrelation between monitored parameters, policies and service SLAs. The process is triggered from monitoring alerts that are next analysed and injected to the lifecycle manager as a series of configuration events that must be implemented in the appropriate order. The process is linked to the service and resource inventory for retrieving the service and network status respectively. The flow of configuration events is then pushed to the deployment manager that interfaces with the underlay network orchestrator. The decision engine can be upgraded with AI-based mechanisms for automated selection of optimised configuration flows. It is noted that in its simplest form (not shown in the diagram above) the decision outcomes can simply return to the service level (end-user) as a list of recommendations that then can be managed manually through the deployment loop.

5.1.2. Monitoring

5.1.2.1. EURECOM's Monitoring

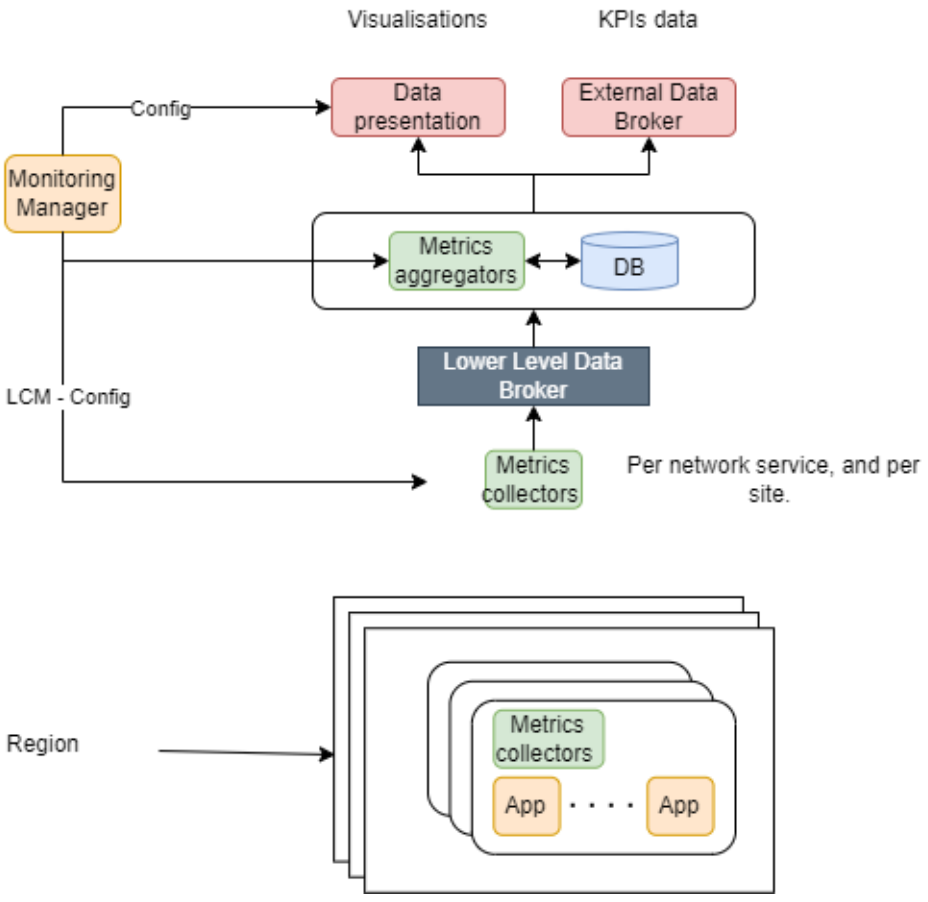

Figure 7. EURECOM's Monitoring System

Figure 7 shows the architecture of the monitoring system. It is designed to monitor services that span multiple domains and infrastructures, such as the deployment over different regions on top of the CECC infrastructure. It is composed mainly of:

- **Monitoring Manager**: It is responsible for creating and configuring the metrics collectors and aggregator for each service. It also creates the dashboards and visualizations related to the service at the Data presentation components.
- **Data presentation**: this component is responsible for providing visualizations about the data to the service owner or applications developer. Each service has it is own dashboards implemented on top of two visualization tools:
    - Grafana [23]: an open-source, multi-platform, interactive Web application for metrics analysis and visualization. It provides customizable dashboards with configurable panels for visualizing metrics stored in the InfluxDB time series database. Dashboards are custom created per service, and metrics are provided on the overall performance of the service and per application.
    - Kibana [24]: It provides visualization capabilities on top of the content indexed on Elasticsearch, allowing us to provide application logs to the application developer. Logs are collected using FluentD [25] and stored on an Elasticsearch [26] cluster. The Kibana user interface capabilities allow logs to be filtered by application, service or cluster. Kibana's user interface features allow users to filter logs by application, service or cluster, and to browse log history.
- **Data brokers:** In order to manage the metrics stream collected all over the CECC infrastructure and to minimize the bottlenecks in the system, we use data broker to transfer the metrics:
    - Lower-level data broker: It is responsible for transferring data from the infrastructure to the CECCM where it can be stored or fed to AI algorithms. At this level we use Kafka, described in Section 5.2.5.2, because it provides routing by topic, which gives a simple and robust model for internal metrics transfer.
    - External data broker: Used to deliver metrics to the application developer. At this level each application owner will have a specific topic for the metrics of its service. The Broker used at this level is based on RabbitMQ, a push-based system described in Section 5.2.5.1.This choice is motivated by the fact that RabbitMQ allows more control over the message routing. It offers more elaborate routing capabilities by providing various exchanges (direct, fan-out, headers, topic).
- **Metrics collectors:** The metrics collectors are deployed for each service part in each region. Their role is to collect metrics from the infrastructure concerning the target applications and services, and transmit the data to the metrics broker. At the lower-level data broker, the metrics are pushed to the topic specific to the subservice.
- **Metrics aggregators:** the role of the metrics aggregators is to collect the metrics from the metrics broker, where the metrics stream is stored in the topic related to the sub service id. The metrics aggregators add the service id to the data in order to allow the mapping between the service and the metrics collected from the infrastructure.

5.2. Data Management

5.2.1. Data Spaces

The term "Data Spaces" typically denotes the establishment of virtual environments that facilitate the secure and effective sharing and management of data among entities or individuals. This notion is at the heart of numerous European strategies designed to forge secure and sovereign data transaction protocols within and between various industries and sectors. Prominent organizations like the International Data Spaces Association (IDSA), Gaia-X, and the Eclipse Foundation are leading the charge in crafting the requisite standards, structural frameworks, and technological solutions to bring the idea of Data Spaces to fruition.

5.2.1.1. International Data Spaces (IDS) Components

Data sovereignty stands as a cornerstone of the International Data Spaces, characterized by the ability of individuals or corporate bodies to exercise autonomous control over their data. The International Data

Spaces initiative advocates a Reference Architecture Model tailored to support this precise capability, encompassing the stipulations for a secure and reliable data exchange within business ecosystems. This initiative strives to fulfill several key objectives: fostering trust among stakeholders, ensuring security and data sovereignty throughout the ecosystem, advocating for the decentralization of data repositories, guaranteeing standardized interoperability for seamless communication across different technological frameworks, and facilitating the emergence of innovative, data-centric marketplaces. Figure 8 delineates the Reference Architecture Model as proposed by the IDSA.

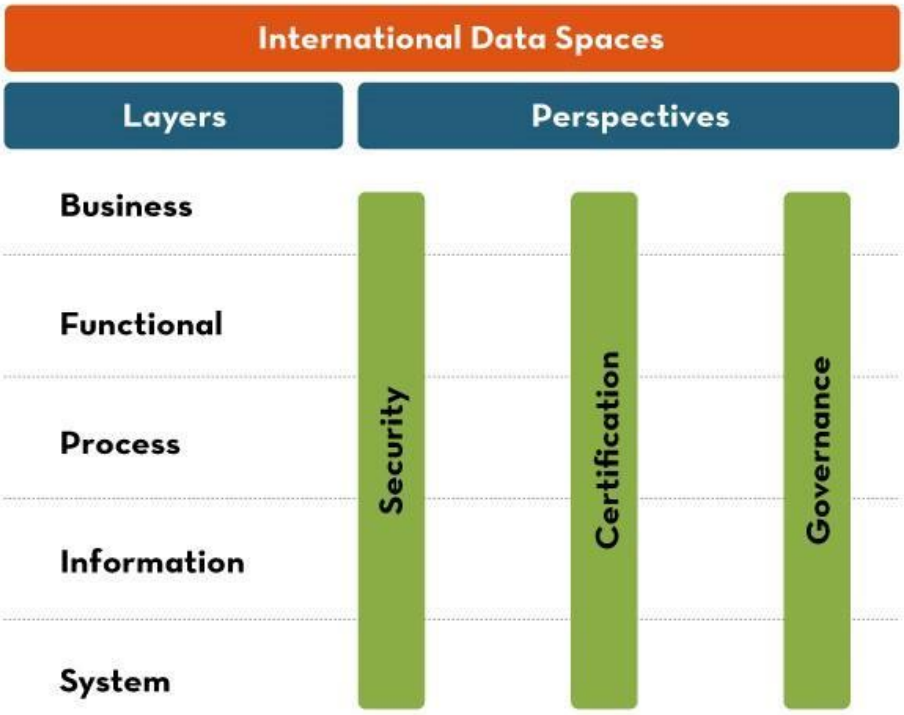

Figure 8. IDSA Proposed Reference Architecture Model

The Reference Architecture Model of the International Data Spaces comprises several stratified components: a) The **Business Layer** articulates the array of potential roles that entities within the International Data Spaces may adopt and delineates the foundational patterns of their interactions. b) The **Functional Layer** outlines the functional prerequisites of the International Data Spaces and delineates the resultant features to be implemented, independent of any pre-existing technologies or applications. c) The **Process Layer** details the interplay among various elements of the International Data Spaces, offering a process-oriented perspective of the Reference Architecture Model. d) The **Information Layer** defines the universal Information Model, which serves as the common vocabulary within the International Data Spaces. This model is a fundamental consensus among participants, ensuring compatibility and interoperability. e) The **System Layer** maps the roles identified in the Business Layer and the processes from the Process Layer onto a specific data and service architecture, forming the technical nucleus of the International Data Spaces. This layer is constituted by an array of core technical components essential for the development of data spaces.

One such pivotal technical component is the Identity Provider, which integrates three ancillary elements: **Certificate Authorities** (CAs), which oversee the issuance and management of technical identity credentials; the **Dynamic Attribute Provisioning Service** (DAPS), which distributes ephemeral tokens infused with current connector information; and the **Participant Information Service** (ParIS), which offers both machine- and human-readable data pertaining to IDS participants.

Conversely, the IDS Connector stands as a critical technical element within the IDS Reference Architecture. The network of International Data Spaces is comprised collectively of these connectors, each facilitating data exchange through its Data Endpoints, thereby negating the necessity for centralized data repositories. These

connectors should be accessible by other organization's IDS Connectors, which might necessitate alterations to firewall configurations or the establishment of a demilitarized zone (DMZ). An IDS Connector should be accessible via standard Internet Protocol (IP) and capable of functioning in any suitable setting. Entities may manage multiple IDS Connectors for purposes such as load balancing or data segmentation, and these can be hosted either on-premises or cloud-based environments. Figure 9 illustrates the architecture of an IDS Connector, which encompasses one or multiple computers or virtual machines, the operating systems they run, the Application Container Management, and the foundational Connector Core Service(s). A detailed description can be found on the IDSA website [27].

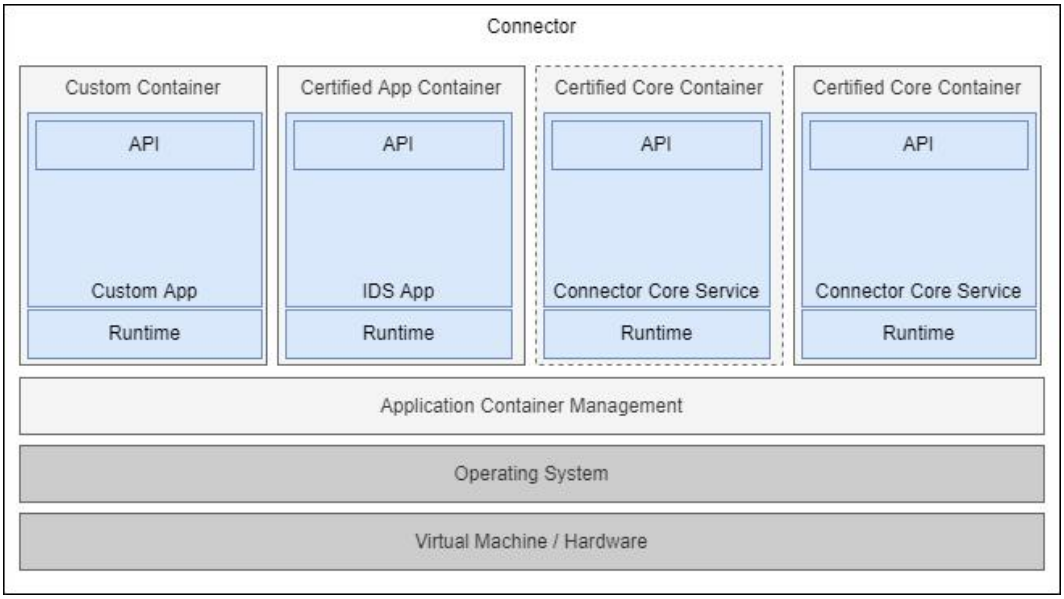

Figure 9. Architectural diagram of IDS Connector

In addition to the IDS Connector, the **App Store and App Ecosystem** constitute another crucial technical component that facilitates the development of data spaces. IDS Connectors can deploy IDS Apps for various functions. There are three distinct categories of IDS Apps: The Data App, which handles data-related tasks; the Adapter App, which connects disparate systems; and the Control App, which manages and oversees operations within the IDS environment. All these app types can be integrated with and managed by the IDS Connector. The **Metadata Broker**, another variant of the IDS Connector, is dedicated to the management of Self-Descriptions, including their registration, publication, maintenance, and retrieval. This component shares similarities with a data catalog, primarily functioning through interactions with the metadata within the catalog. Despite its name, an IDS Metadata Broker does not operate as a message broker, nor does it actively distribute data assets. Furthermore, the **IDS Clearing House**, integral to the architecture, combines an IDS Connector with a logging service that meticulously records transactional data crucial for clearing, billing, and usage control, as outlined in the Process Layer. Utilizing this data, the Clearing House offers Clearing and Settlement Services based on usage agreements, facilitating the automation of financial transactions between Data Providers and Data Consumers. Additionally, it supports a Billing Service that enables Data Space Operators to invoice participants.

Lastly, the **Vocabulary Hub** addresses the need for interoperability within the IDS ecosystem. It equips developers of domain-specific vocabularies with the necessary tools and features to craft, refine, and disseminate their terminologies. While adherence to the RDF (Resource Description Framework) pattern is anticipated, the imposition of other standards, such as Linked Data principles or formal ontologies, remains flexible and not mandatory.

5.2.1.2. GXFS Data Spaces Components

Gaia-X expands the concept of dataspace by incorporating universal data services, such as storage and web servers, to foster interoperability among diverse cloud services and IT infrastructures. Given the presence of

numerous organizations dedicated to defining standards, offering reference architectures, and providing basic implementations for data spaces, Gaia-X commits to a strategy of shared standards rather than a single proprietary solution. This common framework is crucial for ensuring seamless interoperability across different systems. According to Gaia-X, there are several fundamental technical elements that are consistent across data space models, which include:

- Dataspace: This is seen as a comprehensive suite of components that collectively facilitate the autonomous exchange of data and services. A visual representation shows that multiple dataspaces can operate simultaneously within the same compliance framework.
- Asset Provider: This role is assumed by an entity or individual that has ownership or control over a particular asset, such as a dataset or service, and offers it within the dataspace.
- Asset Consumer: This refers to an entity or individual seeking to obtain or utilize an asset like a dataset or service.
- Compliance Services: These are systems in place to certify the integrity of the dataspace components and safeguard their ability to work together.
- Identity Services: These services are responsible for establishing and maintaining reliable identities within the dataspace, thereby fostering trust among its users.
- Catalogue: This is a directory where asset providers can list their offerings, enabling asset consumers to easily discover and access available assets.
- Data Exchange: These are the mechanisms that manage the interactions between providers and consumers, encompassing agreement formation, activity logging, and the transfer of data.

For a more concrete understanding of the Gaia-X dataspace framework, Figure 10 illustrates how these various components interact and integrate to form a coherent and compliant dataspace ecosystem.

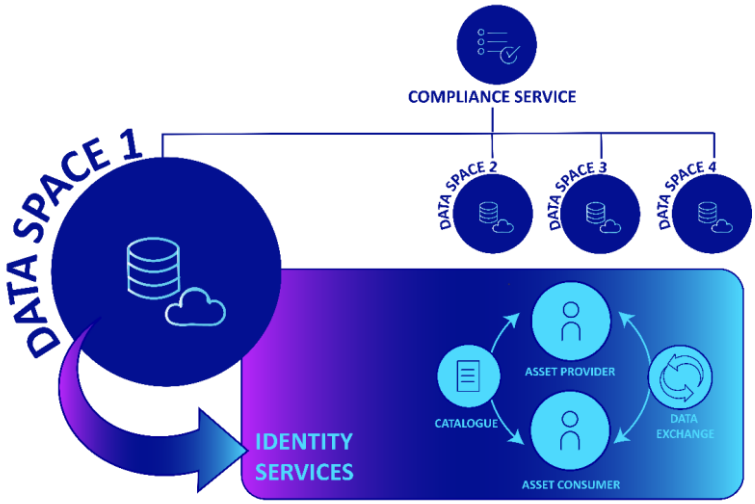

Figure 10. Data Space ecosystem proposed by Gaia-X

5.2.1.3. Eclipse Data Spaces Components

The Eclipse Foundation outlines a data space as a collaborative structure, encompassing both an agreement between organizations and the technological framework that underpins data exchange among various participants. Trust levels within a data space may vary—some participants may share an established trust from prior interactions, while others may be entirely new to each other, without any pre-existing trust, and this can include competitors.

The Eclipse Data Space Components (EDC) offer an integrated solution—including a conceptual design, architecture, codebase, and examples—that delivers a core suite of functionalities for dataspace deployments. These functionalities are both basic and advanced, and they are accessible for customization

and extension through the use of the framework's APIs. This ensures built-in interoperability. The EDC is aligned with the Gaia-X AISBL Trust Framework and the International Data Spaces Association (IDSA) data space protocol, leveraging their specifications.

This framework is composed of several modules, including a data space connector, a catalog for data discovery, a hub for managing identities, a registration service, and a management user interface for data oversight. The project is actively being developed under the Eclipse Foundation's supervision. While it is considered the most advanced initiative of its kind to date, there remains room for improvement and optimization. Helpful resources can be found on their official website [28] and their official GitHub channel [29].

Notably, to run the EDC instances on IONOS cloud, it is easier to use deployment scripts developed by IONOS. This also includes the IONOS S3 Extension for the EDC [30], so the data can be read/stored on S3 of IONOS. For more details, please check the corresponding GitHub repository [30].

5.2.2. Trust

5.2.2.1. Self-Identities Trust Tool

One of the pivotal objectives of Gaia-X/GXFS/XFSC and IDSA innovation is to prepare a reference framework for ensuring the highest level of trust among the different stakeholders and participants of the data-infrastructure ecosystem. Therefore, various components have been specified for developing a Trust framework in the integrated and interoperable system. Below we are going to provide a small summary of those tools and components:

- ***IDSA's Identity and Trust Management:*** To gain reliable information and for establishing trust between the participants IDSA [31] proposed the mechanism for Identity and Trust Management. According to them, each IDSA connector instance possesses its own identity. Remarkably, the distinctiveness of a combination of framework and service instance is tethered to an identifier for the service instance. Furthermore, the distinctiveness encompasses the framework (hardware, firmware, operating system), Connector Core Services software artifacts, configuration settings, and associated IDS Apps or services. The distinctiveness of every connector is unparalleled and is essential to the operation of the system. Importantly, every component within the IDSA possesses a unique identifier (C_UID), which is tethered to the service instance. The uniqueness of this identifier is ensured by the Identity Provider, and each C_UID is associated with a Connector Instance Key (CIK) pair, employed for TLS and potentially for data signing and other identity proofs. Whereas IDS certification process authenticates a blueprint of the entire stack including the platform and connector core services. Apart from the distinctiveness of IDS components, some supplementary information regarding the operating company and its software stack is required. To pave the way for dealing with reliable connections robustly, the exchange of Source Indicators (SI), commonly known as connectors, considers comprehensive identity data and constantly adaptive attribute Keys, referred to as Dynamic Attribute Tokens (DATs), around a central platform—Dynamic Attribute Provisioning Service (DAPS). This recognition procedure is steered around a set plan, acquiring authentication keys, appealing to be issued an authenticated Dynamic Attribute Token, and ensuring bonus points gained on these DATs on safe network connections known as TLS connections. Furthermore, formal procedures known as the "Component Lifecycle" rightly encompasses handling the evolution of every faucet of operational navigation elements of a system starting legitimately from the initial provisioning, passing through vital maintenance stages, and cruising indefectibly to meet the final decommissioning process. A fascinating factor permeating the spectrum of IDS acceptances evidently states that beneficiaries involved purposefully in the wing of IDS, to be precise various classifications embrace organizations of considerable multitude as well as individuals, each procuring distinctive identifiers alias 'Originating unique ID'— O_UID. Tagged prominently in the domain is this unbiased process of unmatched identity management responsible splendidly for the inception of array edges lined on private-public watts and lids in possession of identity proofs

orchestrating an open dance of fostering a systematic designation for each regulatory nexus on the sphere. Further components related to Trust Boots, together with chains commanding righteousness, auto flow vehemently pointing out towards systematic dissection and engaged acknowledgment acts coordinated adopting Intelligent Data Sources hedged under IDS connectors. Furthermore, these are categorized sensibly broadening horizons technically ranging notably from certificates urging the entitlement matrix spreading towards foundations decaled enigmatically as CAs. An added layer vital includes sectors of operation formally indicated as leads putting stacks of elemental rounds to establish structure certificates and related identity deciphers. In context lies a bounty treasure — IDS, Its exciting basket of essentials behind crafting peerless facilities programmed to construct a viable nexus focusing on exceptional security, take into account defying the hatches journeying from holding component-related facts, employment sprinting parallel across certificates based procedures, info consisting of smart data badges streaming tagging interaction patterns, subsystems advanced handling lifecycle recruiting and candidates onboard christened with unique key names, adding 'Originator unique identifier(O_UID)'. A feasible key pathway falls in line jotted subtly as 'e-validations logging trust chains'. The entirety resonates loud posture singing notes complimenting vastly the core of security, orchestrating blessed buttercup of control inside exchange pages gravitating deep into data structures. Figure 11 presents the interaction between IDS connectors and Identity Components.

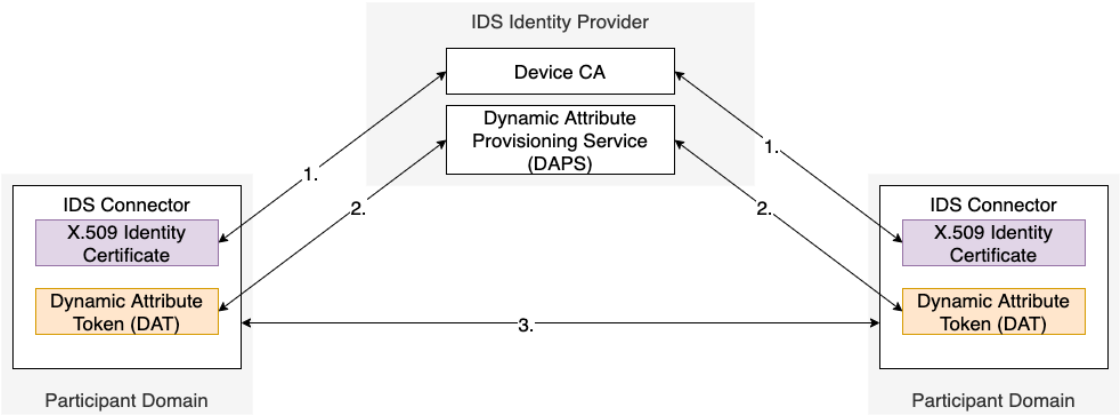

Figure 11. Interaction between IDS Connector and Identity Components

- **Gaia-X/GXFS/XFSC Identity and Trust:** The Gaia-X ecosystem is formed by integration of an Infrastructure and Data ecosystem, both connected via Federation services while the whole architecture is based upon Policy Rules and an Architecture of Standards. Notably, Gaia-X sets the overarching vision and framework, while GXFS (Gaia-X Federation Services) [32] and XFSC (Eclipse Cross-Federation Services) [33] work to translate this vision into tangible services, components, and tools that can be adopted by various stakeholders within the digital ecosystem. According to the Gaia-X, trust framework can be extended by a federation, which can add more requirements on the eligibility of the Gaia-X Trust Anchors and can add additional rules on top of the overall framework. Besides that, the Gaia-X Compliance Service uses the same principles for Issuer, Holder, Verifier, and Verifiable Data Registry as in the Verifiable Credential model. The framework under consideration puts the issuers, also known as **Trust Anchors**, squarely in the driver's seat in managing the Trusted Data Source, an entity endorsed by Gaia-X. This Anchor, also referred to as a Notary, is bestowed with the entitlement of scripting Trusted Data Sources into a lingua franca understood by the organization that looks after the Trusted Data Source. Central to Gaia-X's core tenet is the mechanism of managing decentralized identities and engendering digital trust for both identities and resources based on the Self-Sovereign Identity (SSI) paradigm. Believing in the power of W3C Verifiable Credentials and a Distributed Identifier (DID), Gaia-X lets its participants' commander personal digital identities. Inspired by Gaia-X, GXFS alongside XFSC, suite of services empowered by Identity and Trust veil to fortify federations' abilities in confirming both the signature and the validity of participants,

primarily by gauging the authenticity of their credentials. Meticulously known for its prominent role in sheltering ALL subsets in a federation with an encapsulating stratum of trust, an exclusive service symbolized as the '**Authentication & Authorization Service**' (**AAS**) comes to the fore. In the same orbit as the aforesaid ecosystem lies the **Personal Credential Manager** (**PCM**), which graciously shoulders the commonsensical role of an ardent representative of the consumer. Category-wise, PCM is the frontrunner in encapsulating every distributed identity credential it curates and identity attributes it hosts, in an ironclad layer thickened with affirmation. Steeped predominantly in enhancing users' fusion with the SSI-induced eco-sphere calling upon VCs and DIDs, PCM perfects the nuanced knack of infusing privacy into every interaction. Resolutely toeing the line on bolstering the trust matrix involving the antiphonal communication among various components within the Gaia-X system is the zealous work vision of the **Organization Credential Manager** (**OCM**). OCM takes pride in its centralized role of endorsing credentials to businesses and managing them methodically replete with patches of intricate detailing of trust rituals. Ensuring the dual functionalities of implementation of the codes of conduct per se usage of decentralized components of Gaia-X, and the regulation of trustworthy and mature nodes, are the **Trust Services** (**TRU**). Key to achieving this security balance is cryptographic validation of every detail coming stitched in as credentials from the ecosystem's periphery. Echoing stability in trust circles, the **Notarization Service** (**NOT**) authenticates given master data and transforms it into a W3C-compliant, digitally verifiable representation. These tamper-proof digital assertions about specific attributes are central to gaining the desired trust in provided self-descriptions of assets and participants.

5.2.3. Cloud IoT & Data Platforms

5.2.3.1. AWS IoT Core & IoT GreenGrass

AWS IoT Core and IoT Greengrass collectively form a powerful combination of cloud and edge computing services provided by Amazon Web Services (AWS) for Internet of Things (IoT) applications. These services enable seamless connectivity, data management, and edge computing capabilities, bridging the gap between cloud resources and edge devices.

AWS IoT Core serves as the central IoT hub in the AWS ecosystem. It manages device communication, authentication, and authorization, ensuring secure and reliable connections between IoT devices and the cloud. It leverages AWS's cloud infrastructure and security services. It is designed to work seamlessly with other AWS services such as AWS Lambda, Amazon S3, and Amazon Kinesis for scalable data processing, storage, and analytics.

IoT Greengrass extends AWS IoT capabilities to the edge of your network. It allows you to run code locally on edge devices, enabling real-time data processing and decision-making at the edge. Greengrass enhances the capabilities of IoT Core by providing local compute, messaging, and data caching. It relies on AWS Lambda for local computing, MQTT for local messaging, and integrates with AWS IoT Core for device management and communication. It is a pivotal component for extending AWS IoT capabilities to the edge.

While not directly integrated with IoT Core and Greengrass, container orchestration platforms like Kubernetes and OpenShift can complement these services. They provide containerized application management and orchestration, which can be used alongside IoT Core and Greengrass to enhance edge computing capabilities, especially in scenarios where containerization and microservices architecture are preferred.

AWS IoT Core and IoT Greengrass are designed to work in concert, with IoT Core serving as the cloud-based IoT management hub and Greengrass extending these capabilities to edge and far-edge devices. Edge devices can include a wide range of IoT sensors, cameras, industrial controllers, and more, while far-edge devices are often located in remote or resource-constrained environments. When combined with container orchestration platforms like Kubernetes or OpenShift, AWS IoT Core and Greengrass can effectively manage containerized workloads on edge devices, providing greater flexibility and scalability.

5.2.3.2. Google Cloud IoT Core

Google Cloud IoT Core is a managed service provided by Google Cloud for securely connecting and managing IoT devices. It allows to easily connect and manage large fleets of IoT devices while ensuring scalability, security, and flexibility. Here are some of the key features offered by Google Cloud IoT Core:

- Device Registry: IoT Core provides a device registry to store device metadata and configuration data, making it easy to manage and monitor devices.
- Device Authentication and Security: It offers robust device authentication mechanisms, such as public key infrastructure (PKI), to ensure the security of device connections.
- Cloud Integration: IoT Core seamlessly integrates with other Google Cloud services, including Google Cloud Pub/Sub for data ingestion and Google Cloud Functions for serverless processing.
- Horizontal Scalability: Google Cloud IoT Core is designed for horizontal scalability, enabling you to connect and manage many devices.
- MQTT and HTTP Support: It supports MQTT and HTTP protocols, allowing devices to communicate with the cloud using standard IoT protocols.

Google Cloud IoT Core can be integrated with Kubernetes and OpenShift for optimized resource management, scalability, and high availability in containerized environments. Google Cloud IoT Core is versatile and can be deployed in various environments, including cloud, on-premises, and hybrid settings. Its adaptability extends to integrating with multiple programming languages and technologies.

5.2.3.3. Azure IoT

Azure IoT is a comprehensive set of managed services by Microsoft for securely connecting, monitoring, and managing IoT devices. It offers a scalable and flexible platform for building IoT solutions that can be deployed in various scenarios, including industrial IoT, smart cities, and environmental monitoring. Among the notable features of Azure IoT are the following:

- Device Management: Azure IoT provides robust device management capabilities, allowing to register, monitor, and control IoT devices at scale. It offers features like over-the-air updates and remote device configuration.
- Security: Security is a top priority for Azure IoT. It offers device-to-cloud and cloud-to-device authentication, role-based access control, and integration with Azure Security Center for threat detection and prevention.
- Data Ingestion and Processing: Azure IoT seamlessly integrates with Azure services like Azure Stream Analytics and Azure Functions for data ingestion and processing. This allows to perform real-time analytics on IoT data.
- Azure IoT Hub: Azure IoT Hub is a core component, serving as the central message hub for bi-directional communication between IoT applications and devices. It supports protocols like MQTT, AMQP, and HTTP.
- Azure IoT Edge: Azure IoT Edge extends IoT capabilities to the edge, allowing to run cloud workloads locally on IoT devices. It's compatible with Kubernetes and OpenShift for containerized deployments.
- Integration with Azure Services: Azure IoT integrates with a wide range of Azure services, including Azure Machine Learning, Azure Time Series Insights, and Azure Maps, enabling advanced IoT applications.

Azure IoT is compatible with Kubernetes and OpenShift for containerized deployments and edge computing. Azure IoT Edge supports these platforms, enabling to run containerized workloads on edge devices. This integration optimizes resource management and enables scalability in containerized environments.

Azure IoT is versatile and can be deployed in various environments, including cloud, on-premises, and hybrid settings. Its adaptability extends to integrating with multiple programming languages and frameworks.

5.2.3.4. Spark Works IoT Platform

The Spark Works IoT Platform is a cloud-native flexible and scalable IoT Data Analytics platform that can handle unbounded streams of data in near-real-time and distribute them to multiple applications and services as needed. It is built using cloud-native technologies and can be deployed on multiple platforms ranging from AWS, Azure, and GCP to Docker based orchestrators like Docker Swarm or Kubernetes. The platform is comprised of different independent components, for data ingestion, transformation, processing, storage and distribution. Each component can operate on its own or in combination with the rest of the system. To facilitate communication between the components, all of them expose their functionalities using properly defined API. The core components of the platform are the following:

- Data Mappers: The Data Mappers are responsible for the conversion of incoming data streams to the common format used in the rest of the Spark Works IoT Platform. Each Data Mapper is implemented based on the underlying data sources. The Data Mapper operates receiving messages from the Data Broker and then reroutes the converted information again to the Data Broker for further processing and analysis.
- Data Broker: The Data Broker is the contact point for all data sources of the platform. It also acts as a central hub for the rest of the platform's components to exchange data. The actual software that implements this operation can be replaced based on the needs of each deployment. In most cases, we use the RabbitMQ message broker and the Advanced Message Queuing Protocol (AMQP).
- Data Manipulators: The Data Manipulators are the core components that implement the business logic of each application deployed in the Spark Works IoT platform. Each manipulator receives data messages from the Data Broker and generates the analytics needed. Once the required calculations are computed, the results of the computations are either sent for storage or used to trigger other actions on external services (e.g., sending notifications to end users).
- Data Storages: The Spark Works IoT Platform can use multiple types of storage engines and databases to store the results of the raw data received by the data sources, and/or the results of data analysis performed by the data manipulators.
- IoT Platform API: The Spark Works IoT Platform API is responsible for giving access to the collected data and their meta information to external services and applications. The API is defined as a RESTful API, with clients provided multiple development environments.
- WS API: The WebSocket API is used for receiving live feeds of the raw data and the calculated data analytics. It can be used in application user interfaces for real time and responsive user views.

5.2.4. Edge Operating Systems and Data Agents

5.2.4.1. Spark Works IoT Edge Agent

The Spark Works IoT Edge Agent is an application that can orchestrate IoT devices at the Edges of an IoT infrastructure and is capable of collecting, filtering and processing collected data or controlling available actuators autonomously based on the needs of the installed application. The agent is comprised of multiple components designed for collecting, cleaning, storing, and forwarding data to cloud services or even processing them at the edges of the network. Some of the components that are already available in the Spark Works IoT Edge Agent are the following:

- USB Device Monitor: A component for receiving data through a USB connected sensor device. The component opens a serial connecting and decodes incoming messages into data readings.
- Modbus Device Monitor: A component for connecting to a Modbus interface, either TCP/IP or RTU and monitor Modbus registers in a Modbus slave device. The collected data can be then converted to data readings.
- OS Monitor: A component for monitoring critical information for the operating edge device. Required for collecting information for its operation like the CPU and memory usage, disk usage, or other related information. Such information is then converted to data readings and can be used in the local processing or forwarded to the cloud monitoring services.

- Data Manipulator: A component used for processing data locally at the edges of the network. It can receive either the data generated from a Monitor component, or data generated from other Data Manipulators.
- Learner: A component used for performing Machine Learning operations at the edge of the network, like inference, anomaly detection or training in a federated learning environment.
- Buffer: The Buffer is a component designed for storing data readings locally until they can be forwarded to the cloud if needed.
- Forwarder: The Forwarder is a component for sending the stored data readings (either raw or processed) to the cloud services of the Spark Works IoT Platform.

All components communicate using a lightweight message exchange interface, in most cases an MQTT server (usually using Mosquitto MQTT).

The Spark Works IoT Edge Agent can be executed on many edge platforms and environments as it is designed to be executed using docker containers and lightweight execution environments with limited CPU and memory requirements. The software is also compatible with the AWS IoT GreenGrass for easier deployment and device management.

5.2.5. Message Brokers

5.2.5.1. RabbitMQ

RabbitMQ is a highly scalable open-source message broker software that facilitates message queuing between different components of distributed applications. It acts as an intermediary for processing and routing messages between sender and receiver applications, providing a reliable and efficient means of communication in various software architectures. Some of the RabbitMQ key features and components are the following:

- Message Queues: RabbitMQ utilizes message queues to store and manage messages between senders and receivers. Messages are stored in queues until they are processed, ensuring reliable delivery and decoupling of application components.
- Publish-Subscribe Model: RabbitMQ supports the publish-subscribe pattern, allowing multiple consumers to subscribe to a single message source. This pattern is useful for broadcasting messages to multiple consumers or for implementing event-driven architectures.
- Exchange: Exchanges in RabbitMQ act as message routing agents. Producers send messages to exchanges, which then route the messages to one or more queues based on specified rules or bindings. This provides flexibility in message routing and processing.
- Bindings: Bindings define the relationship between exchanges and queues. They determine how messages are routed from an exchange to specific queues based on message attributes, headers, or routing keys.
- AMQP Protocol: RabbitMQ's core technology is the Advanced Message Queuing Protocol (AMQP) protocol, which defines the format of messages and the rules for communication between the message broker and clients. This protocol enables compatibility with a wide range of programming languages and messaging clients.

RabbitMQ is a versatile message broker, functioning as a standalone service suitable for various deployment scenarios such as cloud environments, on-premises servers, and containers. Its adaptability extends to integration with multiple programming languages, frameworks, and technologies, making it an excellent choice for building distributed systems. RabbitMQ is typically deployed centrally to manage messaging in cloud, on-premises, or hybrid environments, serving as a critical resource for modern software architectures. Additionally, it seamlessly integrates into containerized environments managed by Kubernetes or OpenShift, leveraging container orchestration for streamlined resource management, scalability, and high availability in dynamic and cloud-native settings.

5.2.5.2. Apache Kafka (SPA)

Apache Kafka is a distributed event streaming platform designed for building real-time data pipelines and streaming applications. It excels in handling high-throughput, fault-tolerant, and scalable data streams, making it a popular choice for managing and processing large volumes of data in real-time.

Apache Kafka follows the publish-subscribe pattern, where producers publish messages to topics, and consumers subscribe to these topics for real-time data consumption. It is designed as a distributed system, enabling horizontal scalability and high availability. Kafka consists of brokers, topics, partitions, and replication for fault tolerance. Kafka stores data in an immutable, ordered, and partitioned event log, ensuring durability and allowing consumers to replay and process data at their own pace. Kafka Connect offers a framework for integrating Kafka with various data sources and sinks, simplifying data ingestion and egress. Kafka Streams provides a stream processing library for building real-time applications that can process, transform, and analyze data streams.

Enabling technologies for Kafka include its distributed architecture, log-based storage, extensive integration ecosystem, and compatibility with container orchestration platforms like Kubernetes and OpenShift, which offer enhanced resource management and scalability. Kafka is typically deployed as a distributed cluster of brokers in cloud, on-premises, or hybrid environments, serving as a central data hub for streaming data between various components. It is valuable for managing data between microservices, building real-time analytics pipelines, and handling large-scale event-driven applications. When combined with container orchestration platforms like Kubernetes or OpenShift, Kafka can further optimize its resource utilization and scalability in containerized environments.

5.2.6. State Manager

The **State Manager** is an entity which will internally implement a state migration mechanism, applicable to stateful applications and/or for implementing caching strategies. This mechanism will decide when and where to migrate the service state by relying on user mobility and edge servers network conditions to maintain user's experience optimizing resource allocation on edge servers as well. As the user moves away from its current edge server, the latency will increase causing service interruption and, therefore, affecting the user's experience (e.g., downgrading QoS) and the edge server performance. To solve this task, we adopt a Redis store as the main component, alongside a Reinforcement Learning (RL)-based approach that learns the optimal migration policy. The **State Manager** will involve a "trajectory prediction step" prior to running the RL-based method. More specifically, the following modules are considered:

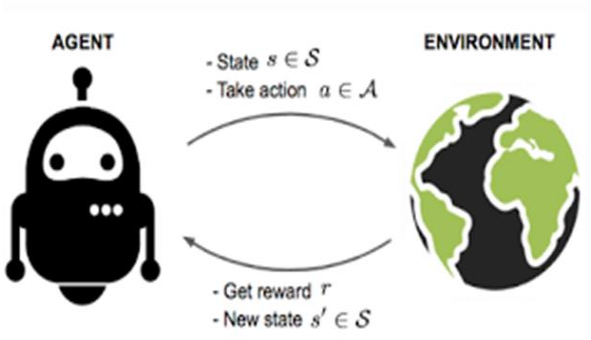

Figure 12. RL scheme: An agent interacts with the environment making smart actions that maximize reward signals [34]

- A Redis store: Redis is an in-memory database, which means it stores data in memory. This storage system allows Redis to deliver high-speed read and write operations, which can be applied in cache and state management applications.
- Trajectory predictor: In the context of prediction task, we will employ a robust technology stack that includes **Python**, **PyTorch**, **TSAI** time series prediction library, **Weights & Biases** (wandb) for

experiment tracking, and **Lightning** for streamlined pipelining and prototyping.

- Node selector: The State Manager leverages an RL-based decision algorithm to find the optimal destination server from the list of servers that are proximal (e.g., k-NNs) to the predicted location of the user to reduce the migration delay and related cost (e.g., computational). As shown in Figure 12, RL involves three elements: a) state set S, b) action set A , c) reward signals r obtained by action $a \in A$ in state $s \in S$.

For the implementation of the RL-based decision mechanism, we will exploit several python libraries that implement some state-of-the-art algorithms (see Table 2). Here is a list:

- **KerasRL:** a Deep RL Python library. It implements some state-of-the-art RL algorithms, and seamlessly integrates with Deep Learning library Keras. Also, it works with OpenAI Gym out of the box.
- **Tensorforce:** it is one of the best open-source Deep RL libraries built on Google's Tensorflow framework. It's straightforward in its usage and has been designed to be modular component based. Moreover, the RL algorithms are agnostic to the type and structure of inputs (states/observations) and outputs (actions/decisions), as well as the interaction with the application environment.
- **TFAgents**: it is another promising RL library designed to implement, deploy, and test RL algorithms more easily due to its modular structure and components that can be easily modified and extended.
- **MushroomRL:** its modularity allows the use of well-known Python libraries for tensor computation and RL benchmarks providing classical and deep RL algorithms under a common interface in order to run them without doing too much work.

Table 2. Number of SOTA RL algorithms implemented in each RL library [35].

KerasRL	**Tensorforce**	**TFAgents**	**MushroomRL**
Deep Q-Learning (DQN) and its improvements (Double and Dueling)	DQN and its improvements (Double and Dueling)	DQN and its improvements (Double)	Q-Learning
Deep Deterministic Policy Gradient (DDPG)	DDPG	DDPG	DDPG
Continuous DQN (CDQN or NAF)	Continuous DQN (CDQN or NAF)	Twin-Delayed Deep Deterministic policy gradient (TD3)	TD3
Cross-Entropy Method (CEM)	Actor Critic (A2C and A3C)	REINFORCE	Fitted Q-iteration (FQI)
Deep SARSA (State-action-reward-state-action)	Trust Region Policy Optimization (TRPO)	PPO	PPO, TRPO
	Vanilla Policy Gradient (PG)	Soft Actor Critic (SAC)	SARSA

	Proximal Policy Optimization (PPO)		DQN

5.3. Local Management System

In AC^3, CECCM has to interact with different LMS to deploy micro-services over the Cloud Edge Continuum Infrastructure. Each LMS is specific to the underlaying technology and infrastructure. In this section, we will review some well-known LMS that manage and handle computing and networking infrastructure.

5.3.1. Local Management System for Computing

Over time the technology for cloud native platform evolved from VM based platform (which used to provide both *Infrastructure as a Service* (IaaS) and *Platform as a* Service (PaaS)) into a 2-tier architecture in which the IaaS is provided using virtualization while the PaaS is using container as its core platform technology. In the past this technology was used only for Linux workloads ("containers are Linux") and for non-Linux workloads virtualization provided PaaS, but today when containers can run VMs in Kubernetes the separation between virtualization for IaaS and containerization for PaaS is becoming the de-facto standard. In AC^3 project, as a CECC platform, we concentrate on the PaaS layer as we manage workloads along the cloud edge continuum.

5.3.1.1. Container Orchestration

One of the reasons that the containerization technology took control over the PaaS layer is the emergence of Kubernetes (k8s) as the de-facto standard for containers scaling and orchestration (there are more reasons) - k8s provides a simple (relative to the problem size), extensible and robust framework for managing the cluster in a declarative way, making the lives of the cluster managers and developers much easier than previously. For example, K8s APIs for changing the cluster configuration are simple CLI commands (applying or modifying YAML file) and do not require any code for executing the APIs (It is possible to call the APIs programmatically but it is not required). Since k8s has become the de-facto standard for cluster management and scaling, some additional technologies emerged that use k8s management technology to manage non-k8s platforms or to imitate k8s in smaller environments (up to single node edge devices). In this section we introduce some of these technologies.

5.3.1.1.1. Kubernetes

Kubernetes commonly abbreviated K8s, is an open-source container orchestration platform designed to automate and manage the deployment, scaling, and operation of containerized applications. Google originally developed it and is now maintained by the Cloud Native Computing Foundation (CNCF) [36]. Kubernetes provides

- **Container Orchestration:** Kubernetes simplifies the management of containerized applications by automating tasks such as container deployment, scaling, load balancing, and health monitoring.
- **Cluster Management:** It organizes containerized applications into clusters, abstracting the underlying infrastructure to ensure high availability, resilience, and resource optimization.
- **Service Discovery and Load Balancing:** Kubernetes provides tools for automatic service discovery and load balancing to ensure that containers can communicate with each other efficiently.
- **Self-Healing:** Kubernetes monitors the health of containers and, if necessary, automatically restarts or replaces failed containers to maintain application availability.
- **Declarative Configuration:** Administrators and developers describe the desired state of an application using configuration files (YAML), and Kubernetes works to ensure the actual state matches the desired state.
- **Extensibility:** Kubernetes is highly extensible, allowing users to add custom resources and controllers to tailor the platform to their specific needs.

A working Kubernetes deployment is called a cluster, which is a group of hosts running Linux containers. We

can visualize a Kubernetes cluster as two parts: the control plane and the compute machines or nodes (either physical or virtual machines). Nodes execute pods, which are composed of containers, and the orchestration is managed by the control plane.

The control plane is responsible for maintaining the desired cluster state, including which applications are running and which container images they utilize. This orchestration system integrates various services to automatically determine the most suitable node for a given task. It decouples work definitions from pods, ensuring that service requests reach the correct pod, even if it migrates within the cluster or is replaced. Kubernetes operates on top of an underlying operating system and manages containers within pods distributed across the cluster nodes.

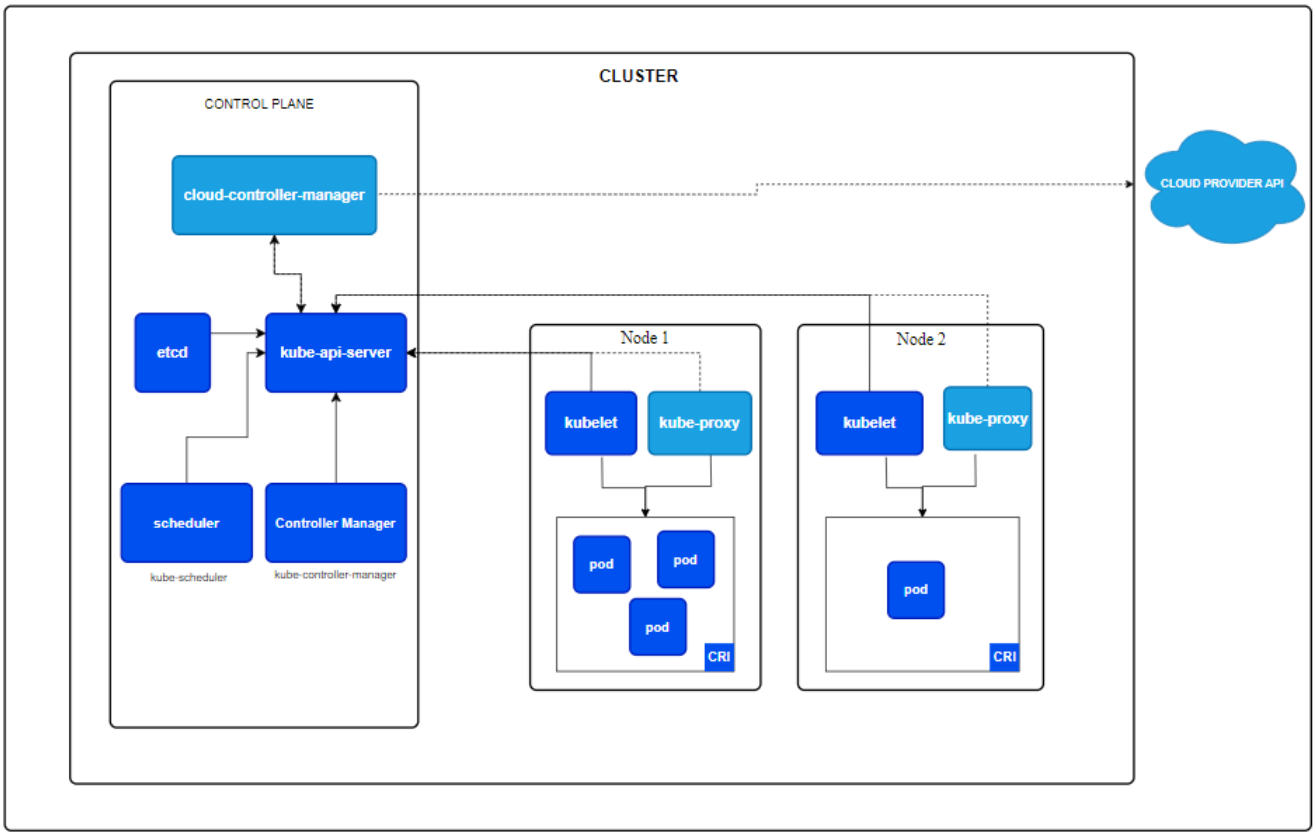

Figure 13. Kubernetes cluster architecture [37]

5.3.1.1.2. OpenShift

OpenShift is an open-source container orchestration platform which is based on Kubernetes but extends its capabilities to compose a comprehensive container application platform. OpenShift simplifies deploying, scaling, and managing applications through container technologies. It encompasses tools and services for the entire application lifecycle and focuses on creating a developer friendly environment, simplifying the complexities of containerisation and allowing them to focus on writing code rather than deployment or scaling procedures.

Core Features:

- Operators are designed to automate the management of complex tasks by utilising Custom Resource Definitions (CRDs) and they encapsulate application specific knowledge to automate tasks such as deployment, scaling, and maintenance which improves efficiency and reduces manual intervention.
- DevOps practices can be seamlessly integrated as OpenShift supports CI/CD pipelines, enabling agile development and release cycles.

- Robust security features are provided such as role-based access control, image scanning, and network policies which ensure a secure application environment.
- A source to image (S2I) tool is available which allows developers to build and deploy containerised applications directly from the source code repository, helping streamline the build process.
- Multi-cloud environments are supported, providing flexibility in deploying applications across multiple cloud providers or on premises machines which provides scalability and redundancy.

5.3.1.1.3. Small form factor Kubernetes

The small form factor Kubernetes platforms are used to manage smaller computing devices by the k8s / OpenShift control plane. These platforms are usually deployed on the edge and far edge and many times on a single node (making this edge node a separate cluster from management perspective) - below is a list of small form factor k8s that might be used in the AC³ project

- K3s
- MicroShift
- Single Node OpenShift (SNO)

K3s:

K3s, is a lightweight, open-source Kubernetes distribution designed for edge and resource-constrained environments. It is a highly simplified and streamlined version of Kubernetes, which aims to make it easier to deploy and manage Kubernetes clusters on smaller hardware, IoT devices, or edge computing scenarios. K3s offers the following technical characteristics:

- **Lightweight:** K3s has a reduced memory and CPU footprint compared to a standard Kubernetes distribution, making it suitable for edge devices with limited resources.
- **Simplified Installation:** K3s provides a straightforward installation process with a single binary, and it can be deployed on a wide range of Linux distributions, including some specifically tailored for IoT platforms.
- **Bundled Components:** K3s includes essential Kubernetes components and dependencies, such as containerd, CoreDNS, and Flannel, to reduce the complexity of setting up a cluster.
- **Security Features:** It includes built-in security features, such as automatic transport layer security (TLS) certificate generation, role-based access control (RBAC), and network policies, to help secure edge deployments.
- **High Availability Options:** K3s supports multi-node clusters and high availability configurations to ensure reliability even in edge environments.
- **Simplified Operations:** K3s abstracts some of the complexities of Kubernetes, making it more accessible to users with limited Kubernetes experience.

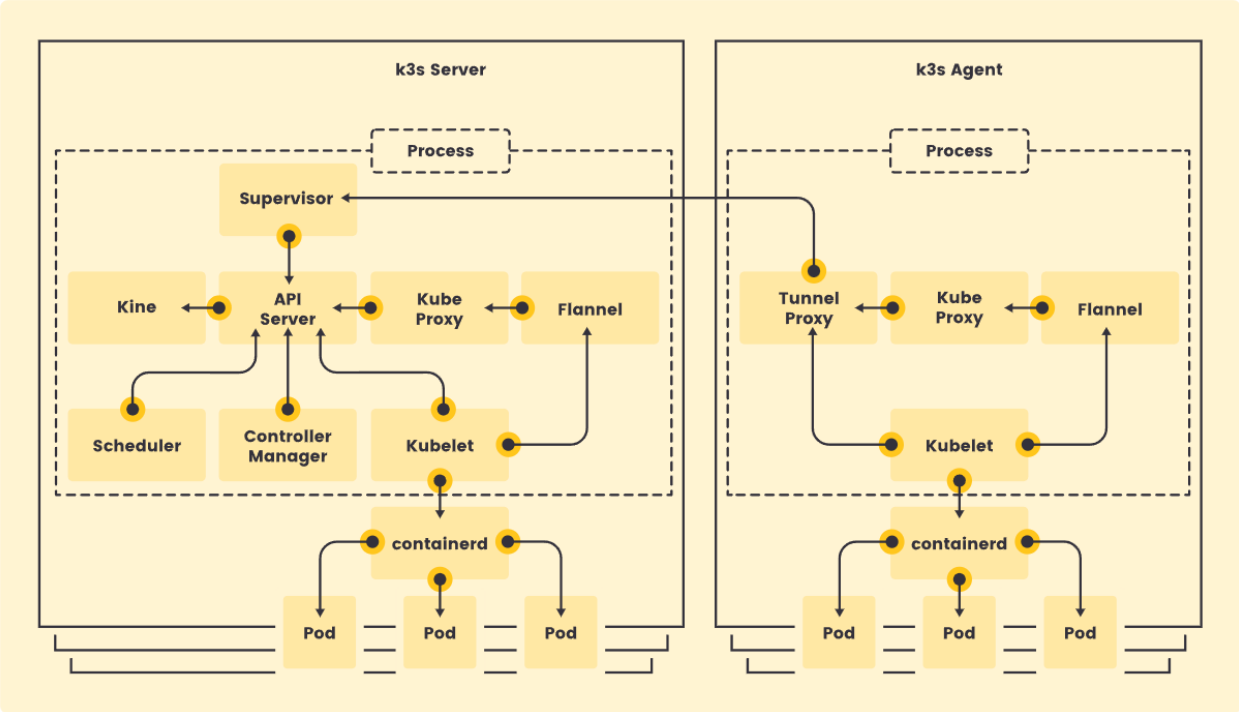

Figure 14. The difference between K3s server and K3s agent nodes [38]

MicroShift:
MicroShift is a project aimed at exploring how OpenShift and Kubernetes can be optimized for small form factor and edge computing.

- **Features**:
  - **Optimized for Edge**: Designed specifically for edge environments, focusing on the specific needs and constraints of these deployments.
  - **OpenShift Compatibility**: Leverages the capabilities of OpenShift, providing a familiar environment for users already using OpenShift.
  - **Resource-Efficient**: Focuses on reducing resource usage to run effectively on low-capacity devices.

Single Node OpenShift (SNO):
SNO is a version of OpenShift designed to run on a single node. While traditional OpenShift clusters have multiple nodes, SNO brings the capabilities of OpenShift to environments where only a single server or virtual machine is available.

- **Features**:
  - **Compact Cluster**: Provides the OpenShift experience in a single-node model, ideal for small, remote, or edge locations.
  - **Lower Resource Requirements**: Designed to operate with lower compute and memory resources compared to a standard OpenShift cluster.
  - **Simplified Management**: Eases the complexity of managing a full-scale OpenShift cluster, suitable for smaller operations or single-use cases.

5.3.1.2. Multi Cluster Manager OCM

Open Cluster Management focuses on simplifying the management of Kubernetes clusters across differing types of infrastructure such as cloud, on-premises, hybrid cloud, and edge. OCM addresses the challenges of managing multiple clusters efficiently by providing a comprehensive framework and set of tools for multi-cluster management. These tools include the following functionality:

- Cluster lifecycle management: Streamlines the creation of clusters, manages scaling, and decommissions unnecessary clusters.
- Application Management: The capability to deploy, update, and scale applications in a coordinated fashion.
- Observability and Monitoring: Provides insights into the health and performance of clusters, enabling administrators to detect issues, optimise resource usages, and improve reliability.
- Policy and Governance: Defines and enforces security and compliance policies to ensure clusters adhere to governmental standards.

5.3.1.3. Energy-aware Container Manager EACM

Energy-aware Container Manager is a tool for the resource orchestration of cloud and edge computing domains that accommodate multiple applications with different SLA requirements. Specifically, it is a framework based on Deep Reinforcement Learning (DRL), which targets to optimize the domains' resource utilization and minimize the energy consumption, while guarantying the applications' SLAs. EACM contains the following key components:

- A custom openAI Gym environment: it emulates domains that accommodate multiple applications.
- A DRL agent: It observes the environment's state (actual resource demand) and takes the appropriate resource allocation decisions in order to optimize a reward function.
- An innovative reward function: the reward function's design targets, on the one hand, to minimize the over-allocation of resources and the energy consumption, and on the other hand, to ensure that the amount of SLA breach events is acceptable.

5.3.1.4. Kubernetes like Control Plane KCP

KCP goal is to provide a single entry point for managing k8s-like applications across multiple clusters, so while the user sees a single management plane, the workload can be distributed across multiple clusters and can be moved between clusters. This is a central feature for managing CECC, where the CECC is composed of multiple clusters and workload is expected to move between clusters as a feature of the platform.

KCP can be a building block for SaaS service providers who need a massively multi-tenant platform to offer services to many fully isolated tenants using Kubernetes-native APIs. The KCP API supports standard Kubernetes types and Custom Resource Definitions (CRDs) for enhanced customisation. It tackles the first challenge of multi-tenancy through Workspaces, providing full isolation and simplifying cluster creation. KCP also addresses the issue of managing multiple clusters by introducing a Syncer and Scheduler solution.

Sync Targets and Locations, defined as CRs, enable administrators to group physical clusters and present them as a unified unit to end-users. KCP's internal schedulers facilitate workload distribution across clusters, aspiring to treat multiple clusters as a seamless computing resource.

The goal is to be useful to cloud providers as well as enterprise IT departments offering APIs within their company. KCP provides the following benefits in security and extensibility:

- KCP's architecture inherently supports strong security measures. The isolation between tenants ensures that data and resources are not inadvertently shared, reducing the risk of data breaches and helping with compliance of data protection regulations.
- KCP can be tailored to specific organizational needs. Its extensibility allows for the integration of custom resources and services, enabling organizations to add specific functionalities unique to their operational requirements.

5.3.1.5. OpenShift Control Plane HyperShift

HyperShift is, to some extent an alternative to KCP that manages Openshift clusters. It is a middleware for hosting OpenShift control planes at scale that solves for cost and time to provision, as well as portability cross cloud with a strong separation of concerns between management and workloads.

The ability to create fully compliant OpenShift Container Platform (OCP) clusters is a key feature of HyperShift and ensures that clusters adhere to defined standards and provide a consistent and reliable environment for containerised applications while maintaining compatibility with OpenShift and Kubernetes toolchains, allowing for seamless integration with existing workloads.

Simplification of deployments and management of OpenShift clusters is achieved by emphasizing compatibility and adherence to industry standards and HyperShift plays a crucial role in optimising resource utilisation, promoting scalability, and enabling an agile and responsive infrastructure for organisations or teams leveraging OpenShift technology.

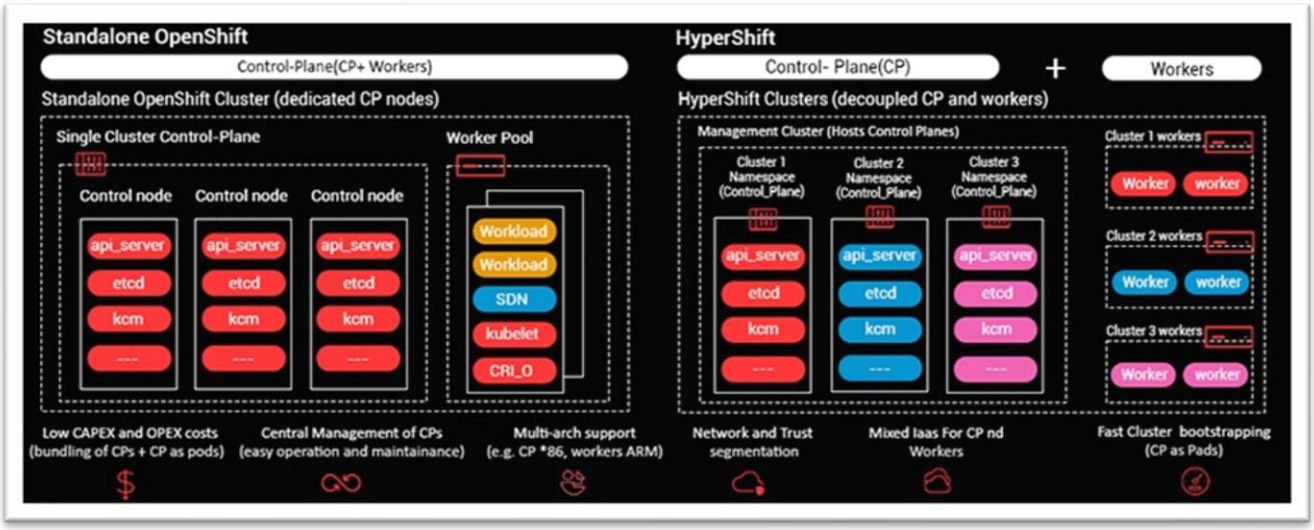

Figure 15. OpenShift vs HyperShift Architecture

5.3.1.6. IONOS Data Center Designer (DCD)

DCD stands as a distinctive instrument designed for the construction and administration of virtual data centers. The graphical user interface of DCD simplifies the process of configuring data centers, offering an intuitive approach. Users have the ability to utilize a drag-and-drop mechanism to arrange and configure the infrastructure components of a data center.

- **APIs/Role**: As is the case with a physical data center, one can use the DCD to connect various virtual elements to create a complete hosting infrastructure. The same visual design approach is used to make any adjustments later. One can log in to the DCD and scale their provisioned infrastructure capacity on the go. Alternatively, one can set defaults and create new resources when needed through DCD.
- **Core Functionalities**: The DCD grants clients the capability to oversee and manipulate a suite of services offered by IONOS Cloud, which include:
    - Virtual Data Centers: Clients can craft, modify, and eliminate entire data centers, establish interconnections between various VDCs, and customize user accessibility throughout their organization.
    - Dedicated Core Servers: Customers have the facility to configure, suspend, and reboot virtual instances, which come with adjustable options for storage, CPU, and RAM. These instances are scalable in accordance with usage demands.
    - Block Storage: Clients are empowered to upload, modify, and remove their private images, or opt for utilizing those provided by IONOS Cloud. They can also generate or preserve snapshots intended for future instance deployment.
    - Networking: Users can allocate and organize static public IP addresses, as well as establish and configure private and public LANs, inclusive of firewall configurations.
    - Basic Features: The service allows the saving and administration of SSH keys, connections through Remote Console, instance initiation via cloud-init, networking recording through

flow logs, and the monitoring of instance usage via monitoring software.

- **Environment compatibility**: As a web application, the DCD is supported by the following browsers:
  - Google Chrome™: Version 30+
  - Mozilla® Firefox®: Version 28+
  - Apple® Safari®: Version 5+
  - Opera™: Version 12+
  - Microsoft® Internet Explorer®: Version 11 & Edge
  - Recommended for using Google Chrome™ and Mozilla® Firefox®.

For more details it is recommend visiting to the official documentation of IONOS DCD [39].

5.3.1.7. IONOS Cloud Developers tool

The developer tool is composed of Cloud APIs, Config Management tools, and Software Development Kits (SDK).

- **Cloud APIs -** IONOS Cloud offers Enterprise-grade Infrastructure as a Service (IaaS) solutions that are manageable through the Cloud API, which serves as an adjunct or a substitute to the browser-based "Data Center Designer" (DCD) tool. Utilizing uniform concepts and features, both the API and the DCD provide equivalent robustness and adaptability. They facilitate a wide range of management operations such as integrating servers, expanding storage volumes, and configuring network settings. A detailed specification of IONOS Cloud API can be found on the official site [40].
- **Config Management tools -** Configuration Management tools enable the automation of Virtual Data Center management and associated infrastructure. Through the utilization of common cloud libraries, software development kits (SDKs), and application programming interfaces (APIs), DevOps teams implement these principles of infrastructure automation. They leverage infrastructure as code to enhance scalability, monitoring, and efficiency. To facilitate direct access to IONOS cloud resources via remote terminals, IONOS provides a command-line interface tool named IonosCTL. This tool incorporates the Cobra and Viper libraries to handle commands and configurations. Cobra not only serves as a library for developing robust command-line interfaces but also as a tool to generate applications and command files, and it's widely used in numerous Go projects alongside the Viper library. For more details one can visit the official documentation site [41].
- **Software Development Kits -** IONOS is offering SDKs for the various cloud services such as – Compute SDK wraps the Cloud API, DBaaS SDKs (PostgreSQL and MongoDB), Auth SDKs, Certificate Manager SDKs, Container Registry SDKs, Data Platform SDKs (Managed Stackable Data Platform), Cloud DNS SDKs, and Logging Service SDKs (where users are able to push and aggregate their system or application logs). Detailed information can be found at the official documentation site [42].

5.3.1.8. Next Generation of Cloud Server (NGCS) API

NGCS platform has been built with interoperability and flexibility in mind. Every functionality has an API to allow automation, integration and, in general, a programmatic approach. Specifically, the API is a programming interface that allows easy access to all functions of the NGCS. This programming interface follows the RESTful API design. This API uses standard HTTP methods to perform queries and operations and allows you to integrate the functions of the NGCS into any applications and services.

- **APIs/Role:**
  - **Cloud Servers:** replicates the actions of the Cloud Panel NGCS. The endpoint depends on the customer, but methods are the same for all [43].
  - **Metadata API**: allows the customers to access programmatically to the internal configuration of each single server. This is useful especially to provide scripts that will be launched with the creation of a new server (using the CloudInit feature) that can make use of variables like IP, DNS, hostname or internal server id [44].
- **Core Functionalities:** NGCS allows the customer to both control and manage the following services provided by Arsys:

- **Cloud Servers:** Running over the latest Intel technology with VMWare, deployed in 30 seconds. A Cloud Server composed of three independent resources: CPU/RAM/Disk. It is possible to resize independently each resource. Advanced options are available like clone servers, Cloud Init or Metadata API. *Cloud Servers work seamlessly with Dedicated Servers allowing hybrid installations.*
- **Block Storage:** Allowed values are from 1 GB to 1000 GB. It can be resized at any time. It is possible to attach or detach a Block Storage to a server at any time. *Only for Cloud Servers.*
- **Dedicated Servers:** 100% dedicated hardware. Multiple models are available with Intel Xeon or AMD processors offering a high performance. The dedicated servers support Virtualization. Also, a recovery mode for troubleshooting purposes is available. In addition, we offer bandwidth up to 10Gbps and 25Gbps and Network redundancy for top models. *Dedicated Servers work seamlessly with Cloud Servers allowing hybrid installations.*
- **Shared Storage:** It can be attached to several servers at the same time, and it has read/read&write permissions can be resized at any time. The size can be increased or decreased (only if there is free space available). The maximum offered is 10TB.

- **Images:** Apart from creating an image from a server it is possible to import an external one to create a new server or ISO from it for Cloud Servers. http/https/ftp protocols are supported including main file-sharing services (Dropbox, OneDrive, S3, and more).
- **Firewall Policies:** By default, any new server has a firewall policy automatically set depending on the software installed in the image. IPv4 and IPv6 allowed. Available protocols: TCP, UDP, TCP/UDP, ICMP, IPSEC, GRE, ANY
- **Load Balancers:** We offer balanced traffic among servers. It is provided using F5 technology and allows custom rules per port and advanced configuration.
- **Public IPs:** IPv4 and IPv6 Public IPs and subnets are supported. Both can be assigned indistinctly to either Cloud or Dedicated servers and can be exchanged from one server to another. In addition, reverse DNS for the IPs can be configured.
- **Networks, Routing & NAT:** Creation of networks with private or public addressing for communicating the servers and establishing public connection to the Internet. We build dedicated environments to meet advanced network needs. In addition, other features such as Routing and NAT are offered.
  - **VPN:** NGCS platform offers a secure connection between desktop and servers. SSL and IPSEC are available. Standard configuration via OpenVPN.
  - **Monitoring:** It works with and without agents. Measures taken are - CPU, RAM, SSD, Transfer, Ping, Ports responding, Processes running A monitoring center is available to see graphics with resources from multiple servers unified. Email notifications configured by customers are available when any of the two thresholds (warning/critical) is reached.
  - **Backup:** Powered by Acronis. Offering the creation of security copies for any device: physical and virtual servers, workstations, and mobile devices. Two locations available: USA and Europe
  - **SSH Keys:** Storing a public key in cloud panel is possible, so that it can be used in the servers during the server provisioning. There are two options to create a key, create a Key Pair and Import Public Key.
  - Basic Features: Logs to check all actions performed with possibility to filter certain timeframe, User & Roles with the possibility of granular roles to grant access in the panel, Usages are shown so the resources usage can be measured, Multicurrency and multilanguage with a Hight availability of 99.9% monthly.
  - Security: 2FA, Intrusion Prevention System, DDoS protection, SIEM
- **Environment compatibility:** As a web application interface provided by the Cloud Panel, the NGCS platform is compatible with:
  - Google Chrome™: Version 30+
  - Mozilla® Firefox®: Version 28+
  - Apple® Safari®: Version 5+

- Opera™: Version 12+
- Microsoft® Internet Explorer®: Version 11 & Edge
- Recommended for using Google Chrome™ and Mozilla® Firefox®.

5.3.2. Local Management System for Networking

The Networking layer provides several features and capabilities that are critical for the proper operation of the CECC. These include, but are not limited to:

- seamless connectivity to services that may be spanning from core cloud to Edge and Far Edge
- expose performance metrics and issues, which in turn allow the CECC and its components, i.e. the AI LCM, to identify SLA violations or risks and take appropriate actions to adapt the workloads accordingly, i.e. perform migration, or horizontal scale-in / scale-out
- provide network programmability to the adaptation and federation layer
- efficiently handle spikes and deluge in incoming traffic
- provide security against application level attacks

The remainder of this section examines in depth the State of the Art (SotA) tools and products that will provide the networking building blocks for the CECC architecture as part of the AC^3 project.

5.3.2.1. Virtual application network

Virtual application network (VAN) is a concept which takes the VPN tunneling from the OSI network layer (L3) which facilitates communication and routing between devices on different networks and moves it to application layer (L7) which interacts directly with user applications and provides network services to the user. This reduces the scaling and management challenges of L3 VPNs, since the L7 scaling and management is a challenge well handled by today's internet-scale applications and infrastructure.

The VAN approach integrates seamlessly into the application lifecycle management. Through technologies like Helm charts or equivalent tools, VAN becomes an integral part of the application's ecosystem. This not only simplifies the network structure but also ensures an app-centric focus. The VAN is readily available to the end app-developer, streamlining the development process and enhancing overall simplicity.

The extension of the VAN exclusively to the application network, rather than the broader L3 network, offers a significant security advantage. This targeted application-centric security model ensures a higher level of fool-proofing compared to a conventional L3 tunnel by narrowing the scope to the application layer, the VAN minimizes potential vulnerabilities and fortifies the overall security.

Considering the context of a federated and occasionally low trust architecture like CECC, the application-centric VAN emerges as the preferred choice. Setting up a VAN within a CECC framework is notably easier and preferable to establishing a fully-fledged L3 network. This not only aligns with the specific requirements and constraints of a federated architecture but also underscores the practicality and efficiency of an app-centric VAN.

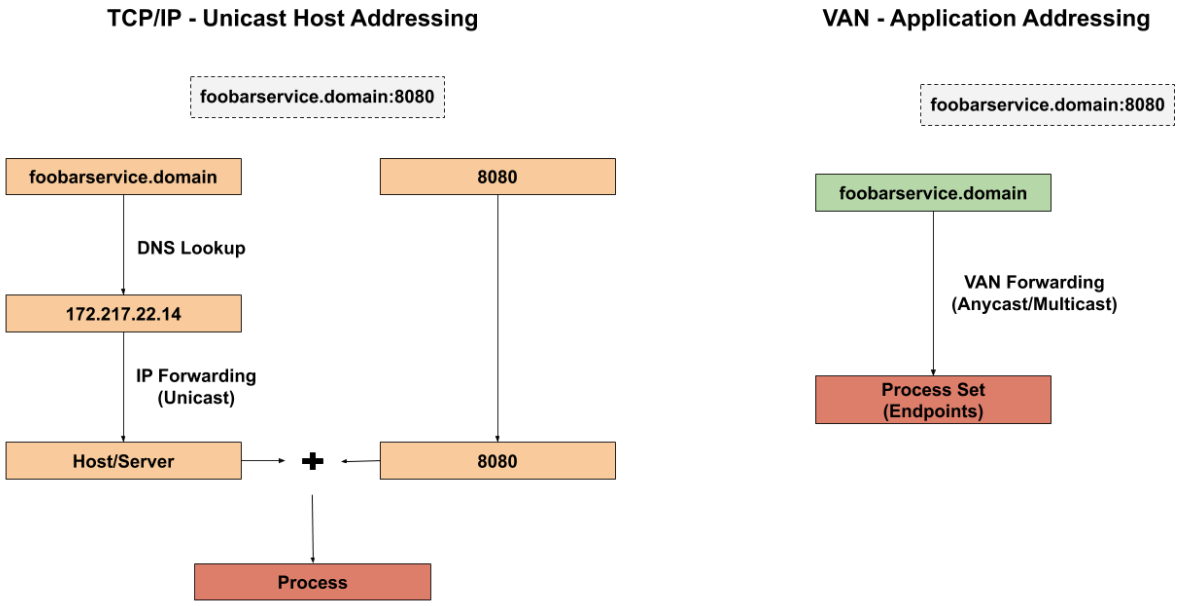

Figure 16. Virtual Application Networks implementation with Skupper [45]

The VAN implementation we have chosen is the Skupper project which aligns with the objectives of our VAN implementation by offering a scalable solution for application networking in dynamic infrastructures when at least one cluster is running in Kubernetes. Through high-level abstraction, Skupper reduces the complexity associated with cross-cluster communication, even in diverse platforms like AWS and GCP and allows for connections between Kubernetes applications and systems running on bare-metal machines or VMs.

5.3.2.2. L2/3 Virtual Private Network (VPN) for k8s clusters

Submariner bridges the gap between higher level Virtual Application Networks (VANs), typically implemented with SD-WAN products, and application-level networks, which are anticipated to be implemented with Submariner in the AC^3 project. It provides a VPN implementation of sorts that can connect overlay networks across different Kubernetes clusters connected to a common public network, without a dependence on a site-to-site connectivity. Towards this end, Submariner can be used to connect disjoint application clusters across multiple sites and datacenters, without implicitly forcing a trust relationship between these datacenters as well.

Submariner provides seamless bridging of networks across disparate Kubernetes clusters and designed to be CNI (Container Network Interface) agnostic which delivers flexibility in its integration while also supporting both encrypted and non-encrypted tunnels, catering to security requirements between connected clusters.

Submariner provides advantages with regards to dynamic resource allocation. Its intelligent load balancing ensures optimised utilisation of resources across connected clusters and its comprehensive traffic management features empower administrators to prioritise and control the flow of data between clusters based on predefined policies. This allows for the tailoring of networking configurations based on workload requirements and operational priorities.

Submariner consists of several main components that work in conjunction to securely connect workloads across multiple Kubernetes clusters, both on-premises and on public clouds:

- Gateway Engine: manages the secure tunnels to other clusters.
- Route Agent: routes cross-cluster traffic from nodes to the active Gateway Engine.
- Broker: facilitates the exchange of metadata between Gateway Engines enabling them to discover one another.
- Service Discovery: provides DNS discovery of Services across clusters.

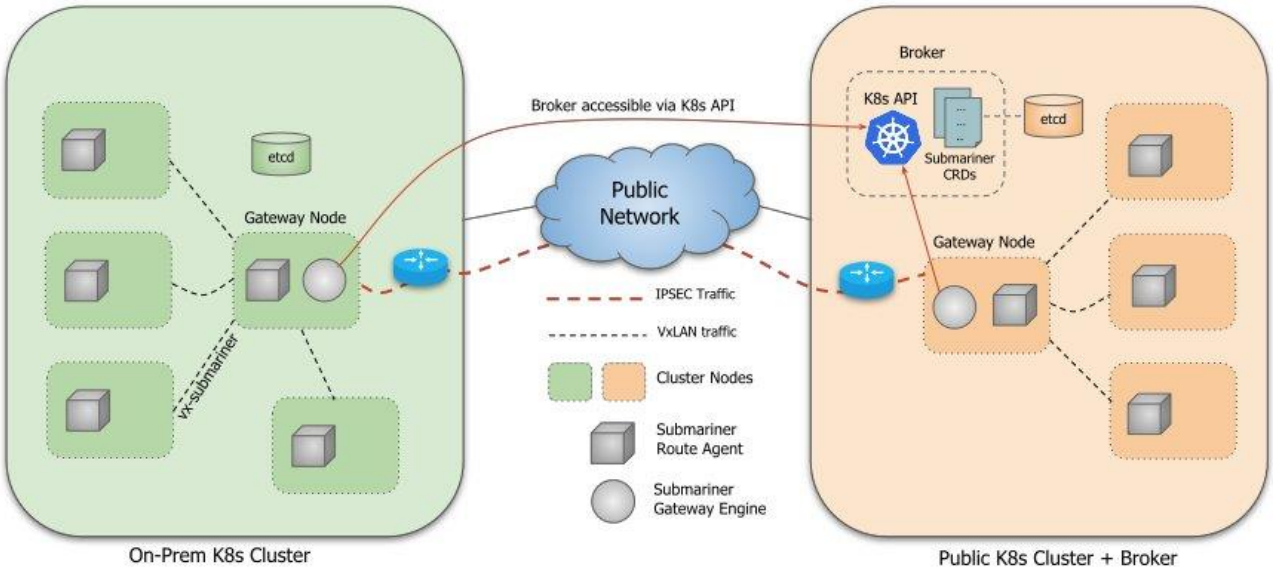

Figure 17. Submariner Architecture

5.3.2.3. NetScaler CPX

NetScaler CPX is the container-based form factor of Cloud Software's Group industry leading Application Delivery Controller (ADC). It can be deployed in any environment that provides support for containers, including but not limited to standalone Docker hosts, Rancher's K3s lightweight Kubernetes (K8s), public clouds managed K8s offerings, Redhat's Openshift and more.

At its core, the NetScaler CPX is an ADC. It is positioned "in front" of applications and intercepts all incoming traffic.

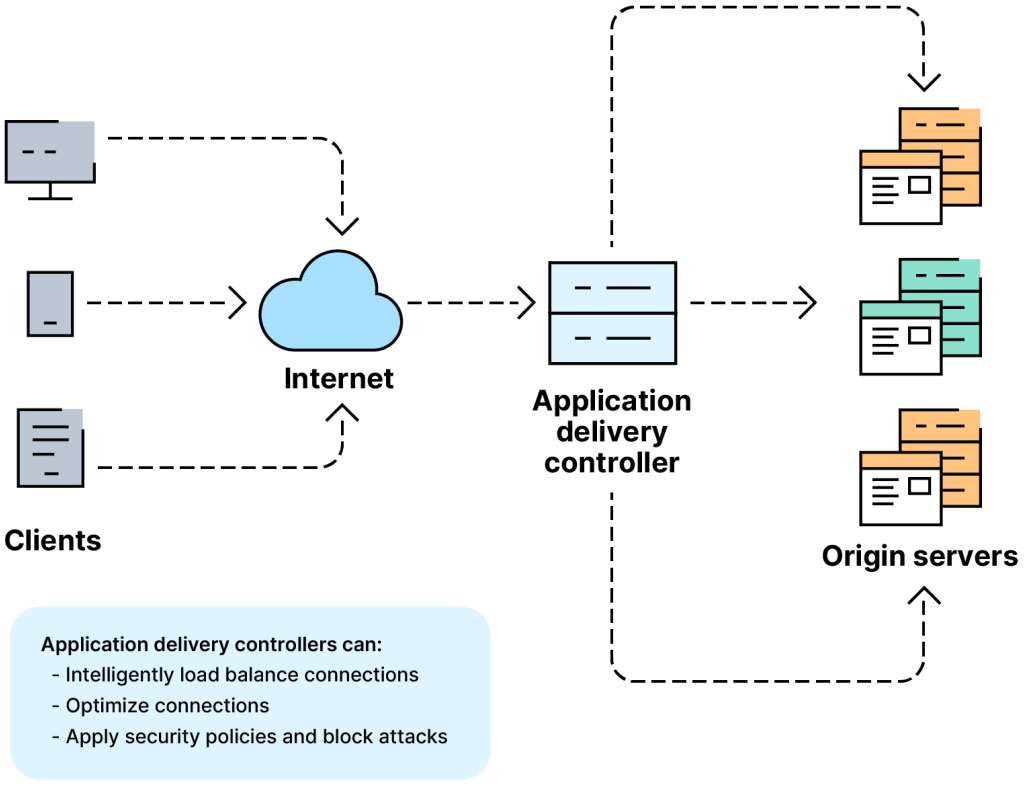

Figure 18. Overview of an Application Delivery Controller

This section examines in further depth the Application Delivery features and capabilities that NetScaler CPX provides to applications and microservices running within the Local Management System (LMS). It should be noted that deeper integration of CPX with Kubernetes based environments, such as support for service mesh or sidecar deployments, typically requires the NetScaler Ingress Controller and are considered in the following section.

Load Balancing

Load balancing is the most basic function of an ADC. Note that while most LMS provide a somewhat basic load balancing solution, however this is typically insufficient to handle the demands and needs of modern applications. The reason is that rudimentary load balancing solutions, such as k8s' kube-proxy, operate at Layer-4 and provide only basic round-robin load balancing. Lack of visibility into the application layer is a significant limitation, since it may lead to unequal load balancing, routing of requests to unhealthy origins, overload of origins and more. As an ADC, NetScaler CPX is a full Layer-7 load balancer and hence alleviates all of these limitations.

Firstly, NetScaler provides full visibility into the application payload. For encrypted applications, this includes terminating the TLS connection on the NetScaler, and optionally re-encrypting requests to the origin.

Having access and "understanding" of the actual application payload, NetScaler can apply far more sophisticated load balancing and application routing. Firstly, it can load balance incoming traffic based on the actual L7 requests, rather than just the L4 connections. Over time L4 connections are a somewhat good proxy of the incoming load, however elephant flows or aggressive or malicious clients may send an overly large number of requests over a single connection and overload a backend origin. By performing load balancing at a L7, NetScaler ensures a fairer distribution of load across origins and eliminates potential "hot spots" that would degrade the user experience. Secondly, it can perform more complex load balancing decisions, i.e. route traffic across different origin sets. A typical historical example has been to route requests for dynamic content across a certain set of origins, whereas requests for static content (i.e. images, stylesheets, etc.) across another set of origins. A more modern example is canary distribution, where a number of users, i.e. based on a Cookie or another L7 request indicator, are routed to a different set of origins that run a newer microservice / application version, to assess whether said version introduces any issues.

Application performance

By virtue of being a full-fledged application delivery controller, NetScaler can significantly increase application performance and efficiency.

One such aspect of such improvement is already hinted above. By providing for more fair load balancing at the application rather than the connection layer, one can avoid hot spots at one or more of the origins. Reducing hot spots not only improves end-user experience, but can also avoid premature horizontal scale-out, that would lead to increased utilization of resources.

NetScaler also contributes to improved performance via its multiplexing capabilities. A rudimentary L4 load balancer will typically map frontend and backend connections in a 1-1 manner. Instead, NetScaler only maintains and reuses a small pool of connections with each origin. Over this small pool of connections it can route L7 traffic from an arbitrarily large number of clients. This not only improves the performance of the origin. It also leads to more predictable utilization, since the performance of the origin becomes a somewhat linear function of the application load. In addition, it indirectly improves the utilization of the underlying LMS, reducing the application cost and improving the density of applications that the infrastructure can hold.

Finally, NetScaler can improve application performance via its more sophisticated server monitoring and load balancing capabilities. Server monitoring can expose deeper insights about the origin status compared to the simple [un]healthy/[not]ready that k8s applications provide, including but not limited to more complex health checks, app response time, bandwidth utilization, concurrent requests, application overload and additional application metrics and insights. These can be leveraged to perform more sophisticated load

balancing algorithms and allow for more fair and more efficient load balancing, which implicitly leads to improved application performance and application utilization.

Application security

Being positioned as the entry point to an application and having full visibility of the application payload, enables NetScaler CPX to address and mitigate threats and vulnerabilities. Threat and attack mitigation typically consists of the following:

- Implementing basic Layer-3 allows and/or deny lists, so as to limit application exposure only to trusted clients.
- Improve encryption security posture, thanks to advanced TLS termination capabilities such as TLS 1.3 support, HTTP Strict Transport Security (commonly known as HSTS), OCSP stapling to provide for certificate revocation and TLS session reuse and ticket, which reduce the impact of TLS encryption while not compromising on security.
- Enforce client authentication
- L4 protection, against TCP spoofing attacks
- Basic Layer-7 allow and/or deny lists, so as to limit application exposure only to well-defined routing endpoints
- Rate-limiting, so as to protect origin servers from a Denial of Service attack, or simply a temporary spike in traffic that it cannot handle until it can scale out horizontally
- Web Application firewall, with the NetScaler inspecting in depth both packet and application headers and payloads for potentially malicious requests, like SQL injection, cross-site scripting or payloads matching an attacks' signature database, and automatically flagging and/or dropping such malicious requests.

Note that for most of its capabilities NetScaler CPX can be configured to operate in a positive or a negative security model. In the latter, more traditional configuration, all traffic is allowed except one that matches a deny rule. In the former, more secure and strict model, all traffic is dropped by default, except one that matches an allow rule.

Application insights

NetScaler CPX can provide extremely rich application insights. At a high level, application insights can be broken into three broad categories:

1. **Events**: these are one-off "signals". They may correspond to an event, i.e. an origin server came offline or back online, an application threshold was crossed, an anomaly was detected, a security violation matching a specific signature was identified, etc.
2. **Time series data (Metrics)**: metrics are exported periodically (typically every few seconds) and provide aggregate stats on a per application and per-origin level. Typically for each metric three values are exported, total, delta (since last report) and rate (per second). These metrics can include, but are not limited to, application requests, successful requests, errors, per error metrics (different counters for 4xx and 5xx errors), throughput, L4 connections and more. Note that NetScaler exposes more than 3,000 entity-level and more than 15,000 system level counters, hence an extensive treatise is beyond the scope of this document.
3. **Transaction logs**: one or more transaction logs may be generated per application request. These logs include tens of fields, with L2-L7 details, such as incoming network interface, source and destination IP addresses and UDP/TCP ports, TLS version and cipher groups utilized, URL details (application FQDN, path, query string parameters), detailed timings (time to establish connection, time to first byte, time to retrieve content, connection duration) and more.

The NetScaler CPX is agnostic of how these insights will be utilized. Typical use cases include, but are not limited to, application monitoring, lifecycle management, reporting and auditing. The NetScaler observability exporter, outlined below, supports ingesting the aforementioned insights to popular monitoring and observability tools. As part of the CECC architecture implemented in AC^3, it is expected that application

insights will be ingested into the “Application and Resources Management” Layer to allow for the following main two functions:

- App developers to monitor Key Performance Indicators of their application

The lifecycle management module to verify that the applications’ SLA is being met, anticipate potential upcoming SLA violations and make appropriate lifecycle decisions if and as needed

5.3.2.4. NetScaler ingress controller for Kubernetes

While NetScaler is a powerful application delivery controller in all its form factors, including the CPX containerized form factor. However, its deployment and configuration can be quite cumbersome, even for a seasoned network admin. Deployment and configuration becomes even more complex in a Kubernetes (k8s) based environment, where all elements of an application, including but not limited to ingress load balancing, ingress security rules, encryption settings, external IP address and FQDN allocation and more, application routing rules are defined in a single “application deployment” file of sorts, and stored as a desired state configuration in the k8s control plane, without an option of independently deploying and configuring independent containers.

The NetScaler ingress controller offers seamless integration of NetScaler CPX in such environments. The following diagram provides an overview of k8s ingress:

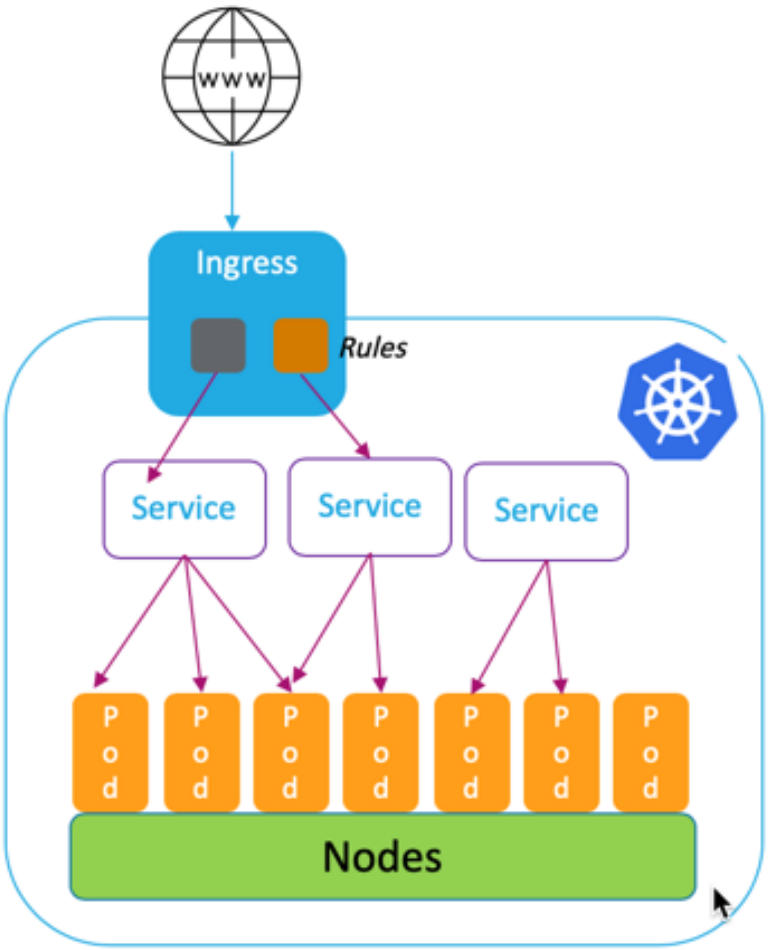

Figure 19. Kubernetes Ingress Overview

- **Ingress** is basically a k8s resource that allows one to define rules to access applications from outside

the cluster and stores the desired state configuration of these rules.

- **Ingress controller** is a microservice application that converts said rules into an appropriate load balancing configuration.
- **Ingress device** is a load balancing application implementation, such as NetScaler CPX.

The NetScaler Ingress Controller is essentially a microservice that communicates with the Kubernetes control plane API and monitors Ingress objects. In case it identifies a new Ingress object that requires a NetScaler CPX, instead of the default or another vendor's CRD, it spins up a new CPX instance for the respective k8s service. It also monitors for Ingress objects changes, such as modifications to update the respective CPX instance configuration or deletions to remove the associated CPX instance.

The advanced capabilities of NetScaler CPX were outlined extensively in the previous section. The remainder of this section focuses on how NetScaler Ingress Controller allows for easier integration of CPX in Kubernetes environments and integration with applications running in such environments.

Supported environments

NetScaler Ingress Controller provides support for all k8s environments on bare-metal or self-hosted on public clouds. It's also been verified to work with all major public clouds managed k8s offerings (Google GKE, AWS EKS, Azure AKS), Google Anthos, Red Hat Openshift, Pivotal Container Service, VMWare Tanzu and more. It is expected to properly integrate with any LMS that provides a k8s compatible API and functionality.

The NetScaler Ingress Controller can be deployed in a number of ways to the target k8s environments listed above, including but not limited to YAML and the kubectl k8s management tool, Helm charts, and Kubernetes Operations (kops).

Features and use cases

The NetScaler Ingress controller supports integration with the k8s environment to automatically deploy CPX instances to handle a variety of use cases.

- **Microservice horizontal scaling**: in case the CECC lifecycle manager takes any action that affects the state of an application and its respective microservices, Ingress Controller automatically updates the CPX configuration to reflect this change. This includes, but is not limited to scenarios like the following:
    - Horizontal scale-out to handle increasing load: CPX is appropriately updated to steer traffic to new instances of the microservice, and quickly mitigates the impact that the increase would have
    - Scale-in, to reduce resource utilization in case of decreasing load: CPX is appropriately updated to stop steering traffic to now invalid instances of the microservice
    - App or microservice migration (due to SLA miss, preference for green datacenter, etc.): this is equivalent to a combination of scale-out and scale-in event, in that the old origin needs to be removed from the CPX' load balancing pool and the new origin added
- **North-South traffic**: the most typical use case is to insert NetScaler CPX as a replacement to the standard ingress proxy offered by Kubernetes. This allows the application administrator to leverage the advanced capabilities outlined in the section above, inserting the NetScaler as an entry point for incoming traffic. Communication between microservices, typically known as East-West traffic, is still being handled by native k8s offerings.

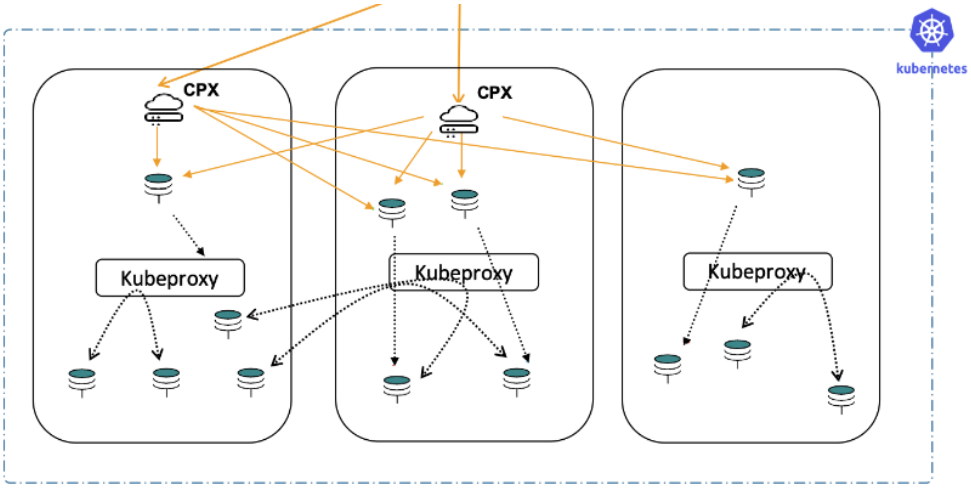

Figure 20. Kubernetes north-south traffic distribution with NetScaler CPX

- **East-West traffic:** commonly known as "service mesh", Ingress Controller allows inserting the NetScaler CPX in a manner that intercepts not only incoming traffic from the internet, but also microservice-to-microservice traffic. This exposes NetScaler capabilities like mutual TLS and SSL offload, HTTP content based routing and filtering, advanced load balancing and rich application insights not only to outside-in traffic but also traffic between microservices. This capability is more critical for modern applications, that are typically composed of multiple tiers, with only a single "frontend" tier being exposed to the internet.

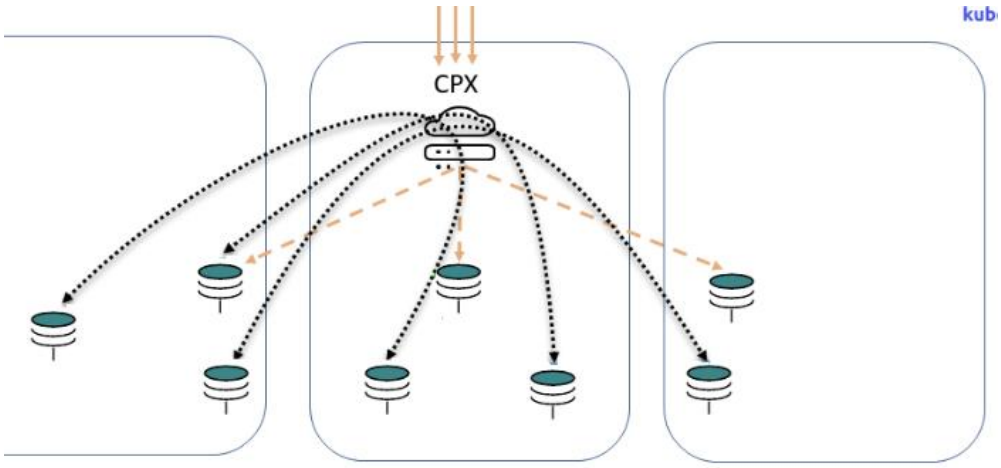

Figure 21. Kubernetes East-West traffic distribution with NetScaler CPX

- **Openshift integration**: NetScaler Ingress Controller supports all basic OpenShift router types for North-South traffic patterns, namely unsecured routes (unencrypted), edge termination (traffic to internet encrypted by CPX, internal traffic unencrypted), re-encryption (CPX terminates and re-encrypts traffic to internal microservices) and passthrough.
- **Canary deployments**: the Ingress Controller stitches together all components of continuous deployment to make canary deployment easier for developers, namely Spinnaker for CD, Kayenta as a plugin to perform statistical analysis of application metrics from the canary pods and Prometheus as the source of the metrics.

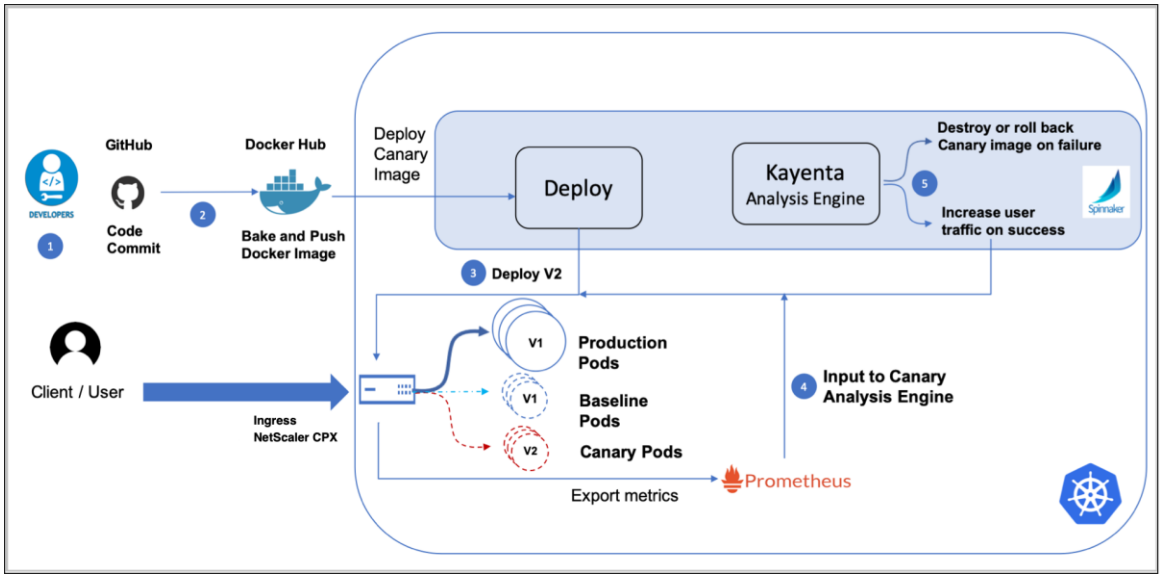

Figure 22. Canary Deployment of microservices in Kubernetes with NetScaler, Kayent and Spinnaker

- **Support for most NetScaler CPX features**: most critical NetScaler features for microservices can be implemented using Kubernetes configuration annotations and applied by the Ingress controller, including but not limited to HTTP, TCP and TLS settings, TLS certificate support, support for HTTP content routing, IP address and DNS record management and more.

5.3.2.5. NetScaler Observability Exporter

As outlined in section 5.3.2.3, NetScaler CPX provides rich application insights, in the form of events, metrics and transactions. NetScaler observability exporter consumes metrics and transaction logs from the NetScaler CPX instances deployed within the Kubernetes cluster. It then integrates with one or more external observability solutions to export said metrics and transaction records.

A typical deployment is illustrated below:

- One or more NetScaler CPX containers push transaction records to the NetScaler observability exporter
- The observability exporter converts said records to a JSON format suitable to Elasticsearch and ingests them using the appropriate protocol

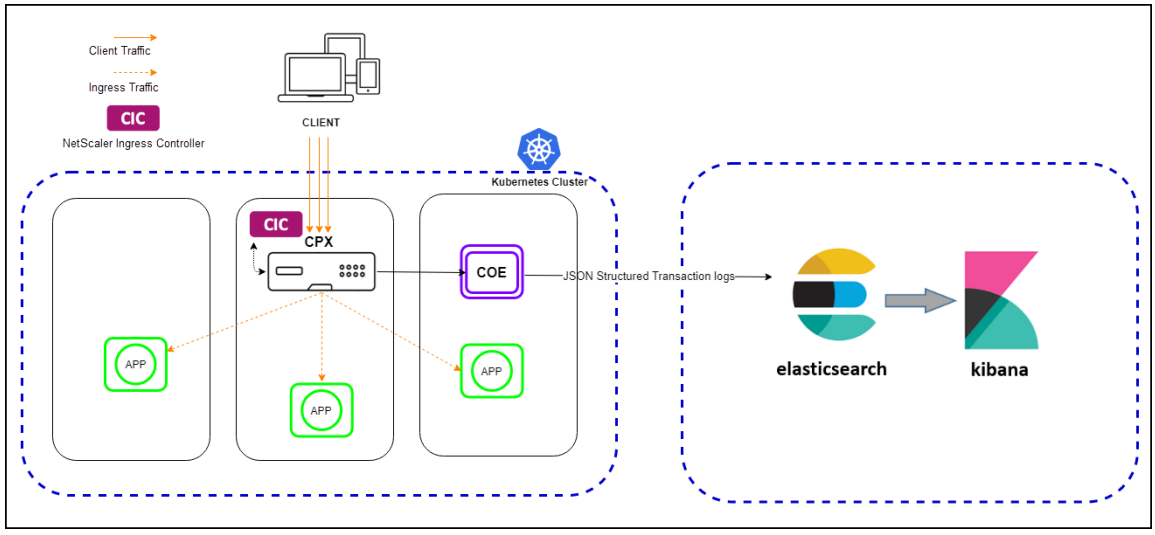

Figure 23. Citrix Observability Exporter architecture

The Observability exporter supports the following:

- Elasticsearch, Kafka and Splunk enterprise for ingestion of structured transaction logs
- Prometheus and Grafana for ingestion of time series data (metrics)
- Zipkin, for ingestion of Opentelemetry trace logs

5.3.2.6. NetScaler service graph and ADM service (CTX)

NetScaler Application Delivery management service graph provides an end-to-end holistic view of microservices applications running within a Kubernetes cluster. Through a single pane of glass, the application admin can quickly identify:

- If the application operates properly or not, either at this moment or at a specific time window in the past
- If not, which of the microservices involved are problematic
- Identify the nature of the problem, i.e. errors, latency / response time issues or both
- View detailed statistics for the faulty microservices

The service graph is of most value when NetScaler CPX is deployed as a service mesh, intercepting both north-south traffic, as well as traffic between microservices.

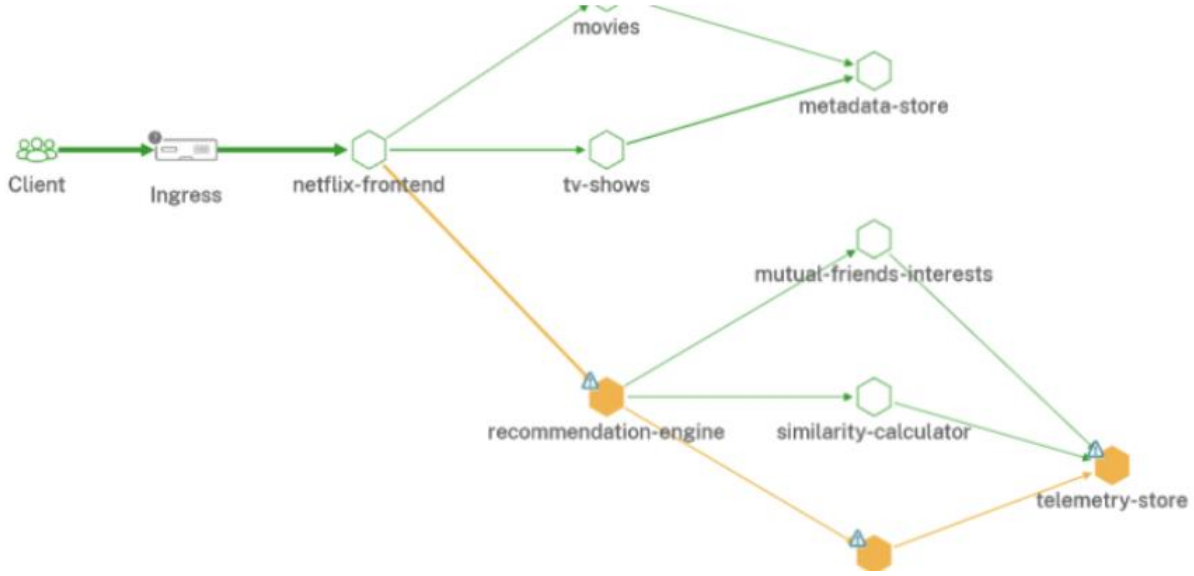

Figure 24. NetScaler App Delivery Management Service Graph

Having a holistic view of the application, the app admin can then easily "focus" on the offending microservices and proceed to appropriate further actions, i.e.

- identify when the problem started occurring and potentially correlate (using an observability platform) it with potential changes to the application or the infrastructure
- assess the severity / impact of the problem, i.e. what percentage of requests were affected and over what time period, what was the latency and response time increase compared to the baseline
- triage the issue to specific microservices and potentially specific root causes, i.e. increase in traffic above a threshold
- consider potential remediations, etc.

5.3.2.7. EveryWAN

EveryWAN is a flexible and open-source Software Defined Wide Area Network (SD-WAN) solution. EveryWAN's architecture aims to simplify the management and operation of wide area networks by

decoupling networking hardware from its control programs and using software and open APIs to abstract the infrastructure, manage connectivity, and deliver services. EveryWAN offers flexibility in choosing WAN connections, making it possible to leverage various transport services, including low-cost broadband, MPLS, LTE/5G, and broadband internet services. The architecture is designed to be highly adaptable to different needs, supporting various network behaviors and routing approaches.

EveryWAN adopts an approach in which traditional IP protocols coexist seamlessly with a SouthBound (SB) interface in the edge device to configure it. The SD-WAN edge device architecture combines Programmable IP Forwarding Engine (P-IPFE), an IP routing daemon (IPRE) and a Southbound API (SB API), enabling coexistence between traditional routing like BGP and programmable routing. Edge devices can be deployed on any server with computing, storage and networking capabilities, to replace CE equipment.

Both the Controller's Northbound and Southbound APIs are implemented using gRPC protocol. The agent at the SD-WAN controller (also called EverEdgeOS) processes the Northbound API request and, depending on the type of message sent by the orchestrator, forwards it to the appropriate manager, e.g. if the request concerns the creation of an Overlay network, it forwards the request to the Overlay-Manager. The Orchestrator exposes the SD-WAN functionalities as a REST API. Both the EverEdgeOS controller and EverEdge devices require a Unix-based system and are developed using Python and uses different libraries to communicate with the linux kernel.

It employs a Zero Touch Provisioning (ZTP) to facilitate various functions, including the download of the routing daemon's bootstrap configuration and device authentication with the controller. Additionally.

EveyWAN can be used in the context of the CECC for: **-** Optimizing the WAN connectivity between CECC sites. **-** Load-balancing between different overlay connections **-** Dynamically adapt application traffic (could be classified depending on parameters defined by the CECM) to maximize SLA satisfaction. **-** Monitor the overlay infrastructure of the CECC.

5.3.2.8. FlexiWAN

FlexiWAN is another open-source Software Defined Wide Area Network and Secure access service edge (SASE) solution. It aims to provide an open architecture, vendor-agnostic approach to SD-WAN, unleashing the power of flexibility and control for enterprises and service providers.

Key features of FlexiWAN include an auto full-mesh and hub & spoke tunnel creation for selected devices, zero-touch provisioning, automatic detection and failover for multiple WAN connections, load balancing across multiple tunnels, automated deployment behind NAT setups, constant monitoring with automatic issue correction, and multi-tenant management options.

From a technical architecture perspective, FlexiWAN consists of a software-based edge device called flexiEdge and a central management system, flexiManage. FlexiEdge is composed of router infrastructure using FD.io VPP, a routing control plane based on Free Range Routing, and the FlexiWAN Agent for secure communication with the management plane. FlexiManage operates on a scalable web server, facilitating network management, device and network statistics collection, and providing updates. However, it lacks support for Virtual Routing and Forwarding (VRF) and does not provide network analytics or flow telemetry.

FlexiWAN can be used in the context of the CECC for:

- Load-balancing between multiple tunnels
- Monitor the overlay infrastructure of the CECC.

6. Gap Analysis

After reviewing the existing tools in Section 4 and Section 5 and establishing the link with the AC³ architecture components, this section analyses the gap in terms of missing tools to implement the CECCM components. The missing tools will be implemented in WP3 and WP4. A summary of the tools and gaps in the architecture is provided in Table 3.

Table 3. Summary of the technological tools

Architecture plane	Architectural component	Available technological tools

User Plane	Application Gateway	- EURECOM's web portal:
		  o User Interface
		  o NST - Application Descriptor
		  o Allow to specify the application instantiation order
		  o Allow the configuration of service CM, IN ...
		  o Visualisation of KPIs and metrics
		- MAESTRO SO – front-end interface:
		  o User Interface
		  o Application graph for communication between applications
		  o Dashboard on resources usage
		- **Gaps:**
		  o Lack of Analytics

	Service Catalogue	- Data catalogue:
		  o IDS Metadata Broker
		  o XFSC Catalogue
		  o EDC Catalogue
		  o Piveau Catalogue
		- Deployment Catalogue:
		  o Databases used to store the application descriptors.
		- **Gaps:**
		  o No unified view on applications blueprints, software, ML models and Data. (Need for interface design for the service catalogue)
		  o No catalogues for software and blueprint of micro-service-based applications.

	Ontology & Sematic aware Reasoner	- Ontology editors:
		  o Protégé
		  o Web Protégé
		- Semantic Reasoners:
		  o Pellet
		  o HermiT
		  o Fact++
		  o RecerPro
		- Frameworks to manage ontologies:
		  o Apache Jena

		○ OntoStudio - **Gaps:** ○ Knowledge base for reasoning techniques need to be provided or constructed.
Management Plane	App & Resources Mgmt	- EURECOM's CLiSO: ○ Multi cloud management ○ Supports different cloud management technologies: Kubernetes, Openshift, K3s ○ Resources allocation, and check available resources ○ Update service ○ Monitoring of metrics and logs - MAESTRO: ○ Translation of SLA and resources requirements to intent ○ Select the network and deployment location based on the SLA ○ Provide monitoring of resources usage - **Gaps:** ○ Limited AI based resources management ○ No models for profiling applications and resources.
	Monitoring	- EURECOM's monitoring: ○ Provide a scalable data pipeline for metrics collection ○ The monitoring systems covers the collection of network, infrastructure and application's metrics - **Gaps:** ○ Need to be adapted to fit AC³ purpose in order to collect also data produced by the applications
	Data Management	- Spark Work's IoT Platform - Spark Works IoT Edge Agent - Cloud based solutions: ○ Azur IoT ○ Google Cloud IoT Core ○ AWS IoT Core & GreenGrass - **Gaps:** ○ How to include all the solutions in one system ○ How to provide applications with access to the Data
	Adaptation and Federation Layer	- EURECOM's CLiSO ○ Interacts with different LMS using the notion of plug-in that uses the LMS NBI

		○ Provide the functionalities of the adaptation agents and adaptation gateway - MAESTRO: ○ integrated with Kubernetes LMS ○ Can provide functionalities of adaptation agent for Kubernetes based LMS - **Gaps:** ○ Resource Broker functionality is not fully provided by the proposed solutions. ○ Resources Discovery over a federated infrastructure using the NIST CFRA model is not implemented by the tools.
Infrastructure or CECC plane	LMS for computing	- Several types of LMS available: ○ Kubernetes ○ Openshift ○ K3s, microshift, SNO - Vendor specific tools: ○ IONOS Data Center Designer ○ IONOS Cloud Developers ○ Next Generation of Cloud Server - **Gaps:** ○ Need a Unified interface for monitoring information exposure.
	LMS for networking	- Networking solutions: ○ VAN ○ L2/3 VPN ○ NetScaler CPX ○ Netscaler Ingres controller ○ Netscaler Observability Exporter ○ EveryWAN, FlexiWAN SDN controllers

Based on the analysis of the technological tools and their functionalities we conclude the following:

- There need to be an Integration of the deployment and data catalogues to provide a unified view of software, services, ml models and datasets, this view can be provided by designing a common interface for the Service Catalogue.
- The applications LCM orchestrators proposed does not provide AI solutions for applications and resources management. Thus, the machine learning modules should be implemented in the project including the application and resources profilers.
- The proposed User Interfaces that can be used to implement the application gateway do not include data analytics, and the visualisations that they provide are mainly about resources usage and not the data generated by data sources, applications and IoT Devices.
- For data management, the project needs to provide a method to access the data flow of hot and cold data sources from the deployed applications by CECCM.
- For Ontology & Semantic aware reasoning, the proposed tools allow the ontology editing. Therefore, the custom solutions for applications and resources management need to be implemented, and a knowledge base for reasoning techniques need to be either built and provided to the reasoner or there should be an implementation of models to construct this knowledge base.

- The solutions provided as Local Management Systems for computing and networking cover all the technological requirements needed to manage services over the CECC infrastructure. However, in order to manage those infrastructures via the CECCM, AC^3 should provide solutions and integration APIs for cloud federation management following the IEEE SIIF implementation of the NIST CFRA. These solutions include: The resources broker and the selections algorithm to choose the infrastructure where to deploy a service; The resources discovery module that collects information about the federated resources.

7. Conclusion

In this deliverable, D2.3, we have presented a comprehensive overview of the essential technological tools for the CECC (Cloud Edge Computing Continuum) architecture, and particularly for the CECCM within the AC^3 (Agile and Cognitive Cloud-edge Continuum management) project. These tools have been meticulously assessed, considering their roles, compatibility with the AC^3 architecture, and feasibility of adaptation.

These tools are pivotal components that underpin the CECC framework's functionality and its ability to manage cloud-edge resources effectively. This document, related to task T2.3, has also provided an initial glimpse of how these tools will be integrated into the AC^3 CECCM (CECC Management) architecture, setting the stage for future development and refinement.

These technological tools are not static components but integral building blocks of the CECC framework, designed to enhance its functionality, adaptability, and performance. Moreover, this deliverable analyzed the gaps that require adaptation and the development of new components to lay the groundwork for the initial end-to-end integration of these tools into the CECCM architecture.

In summary, this deliverable is a key reference for WPs 3 and 4 for the implementation of the different building blocks of the AC^3 architecture avoiding implementing them from zero. Furthermore, the output of this deliverable will ease the integration process of the different functional blocks of the architecture in WP5.

References

[1] "S. Arora, K. Boutiba, M. Mekki and A. Ksentini, "A 5G Facility for Trialing and Testing Vertical Services and Applications," in IEEE Internet of Things Magazine, vol. 5, no. 4, pp. 150-155, December 2022, doi: 10.1109/IOTM.001.2200206".

[2] "S. Arora, A. Ksentini and C. Bonnet, "Lightweight edge Slice Orchestration Framework," ICC 2022 - IEEE International Conference on Communications, Seoul, Korea, Republic of, 2022, pp. 865-870, doi: 10.1109/ICC45855.2022.9838854.".

[3] "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; VNF Descriptor and Packaging Specification".

[4] "protégé," [Online]. Available: https://protege.stanford.edu/.

[5] "WebProtégé," [Online]. Available: https://webprotege.stanford.edu/.

[6] "Apache Jena," [Online]. Available: https://jena.apache.org/.

[7] "OntoStudio," Semantic Web Standards - World Wide Web Consortium (W3C), [Online]. Available: https://www.w3.org/2001/sw/wiki/OntoStudio.

[8] E. P. B. G. B. C. K. A. &. K. Y. Sirin, "Pellet: A practical OWL-DL reasoner," *Journal of Web Semantics,* pp. 5(2), 51–53. https://doi.org/10.1016/j.websem.2007.03.004, 2007.

[9] B. H. I. M. B. S. G. &. W. Z. Glimm, "HerMiT: an OWL 2 reasoner.," *Journal of Automated Reasoning, 53(3), 245–269.,* no. https://doi.org/10.1007/s10817-014-9305-1, 2014.

[10] D. &. H. I. Tsarkov, "FACT++ Description Logic Reasoner: System description," in *In Lecture Notes in Computer Science (pp. 292–297)*, https://doi.org/10.1007/11814771_26, 2006.

[11] V. &. M. R. Haarslev, "Racer: A Core Inference Engine for the Semantic Web. Ontology-based Tools," http://ceur-ws.org/Vol-87/EON2003_Haarslev.pdf, 2003.

[12] H. Lan, "SWRL : A semantic Web rule language combining OWL and ruleML," W3C Member Submission. https://ci.nii.ac.jp/naid/10028232353, 2004.

[13] M. &. B. H. Kifer, "RIF Overview (Second Edition)," W3C Working Group Note. https://www.w3.org/2013/pdf/NOTE-rif-overview-20130205.pdf, 2013.

[14] "IDS Whitepaper," [Online]. Available: https://github.com/International-Data-Spaces-Association/IDS-RAM_4_0/tree/main.

[15] "GXFS catalogue," [Online]. Available: https://gaia-x.gitlab.io/data-infrastructure-federation-services/cat/architecture-document/architecture/catalogue-architecture.html.

[16] "XFSC repository," [Online]. Available: https://gitlab.eclipse.org/eclipse/xfsc/cat/fc-service.

[17] "GXFS Specs," [Online]. Available: https://www.gxfs.eu/download/1740/.

[18] "IONOS gx catalogue repository," [Online]. Available: https://github.com/Digital-Ecosystems/gx-catalogue-ionos.

[19] "eclipse respositories," [Online]. Available: https://github.com/eclipse-edc.

[20] "piveau," [Online]. Available: https://www.piveau.de/en/.

[21] "DCAT," [Online]. Available: https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/11. [Accessed 2023].

[22] "piveau doc," [Online]. Available: https://doc.piveau.eu/general/introduction/.

[23] "Grafana," [Online]. Available: https://grafana.com/.

[24] "Kibana," [Online]. Available: https://www.elastic.co/kibana.

[25] "FluentD," [Online]. Available: https://www.fluentd.org/.

[26] "Elasticsearch," [Online]. Available: https://www.elastic.co/elasticsearch.

[27] "IDSA website," [Online]. Available: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/layers-of-the-reference-architecture-model/3-layers-of-the-reference-architecture-model/3_5_0_system_layer/3_5_2_ids_connector#ids-connector.

[28] "EDC website," [Online]. Available: https://eclipse-edc.github.io/docs/#/.

[29] "EDC Repository," [Online]. Available: https://github.com/eclipse-edc.

[30] "EDC IONOS extension," [Online]. Available: https://github.com/ionos-cloud/edc-ionos-s3. [Accessed 2023].

[31] "IDSA," [Online]. Available: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/perspectives-of-the-reference-architecture-model/4_perspectives/4_1_security_perspective/4_1_2_identity_and_trust_management.

[32] "GXFS," [Online]. Available: https://www.gxfs.eu/set-of-services/.

[33] "XFSC," [Online]. Available: https://projects.eclipse.org/projects/technology.xfsc/reviews/creation-review.

[34] "RL overview," [Online]. Available: https://lilianweng.github.io/posts/2018-02-19-rl-overview/.

[35] "RL tools," [Online]. Available: https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python .

[36] "CNCF," [Online]. Available: https://www.cncf.io/.

[37] "Kubernetes architecture," [Online]. Available: https://kubernetes.io/docs/concepts/architecture/.

[38] "K3s," [Online]. Available: https://k3s.io/.

[39] "IONOS DCD doc," [Online]. Available: https://docs.ionos.com/cloud/getting-started/data-center-designer.

[40] "IONOS Cloud API doc," [Online]. Available: https://api.ionos.com/docs/cloud/v6/.

[41] "Config management tools," [Online]. Available: https://docs.ionos.com/reference/config-management-tools/config-management-tools.

[42] "IONOS SDK," [Online]. Available: https://docs.ionos.com/reference/software-development-kits/sdk-bundles.

[43] "Cloudbuilder API doc," [Online]. Available: https://api.cloudbuilder.es/documentation/v1/en/api/documentation.html.

[44] "Cloudbuilder metadata API doc," [Online]. Available: https://api.cloudbuilder.es/documentation/v1/en/metadata_api/documentation.html.

[45] "TCP/IP vs VAN Addressing," [Online]. Available: https://itnext.io/virtual-application-networks-van-for-multi-cloud-multi-cluster-and-cloud-edge-interconnect-1f63a8081f41.