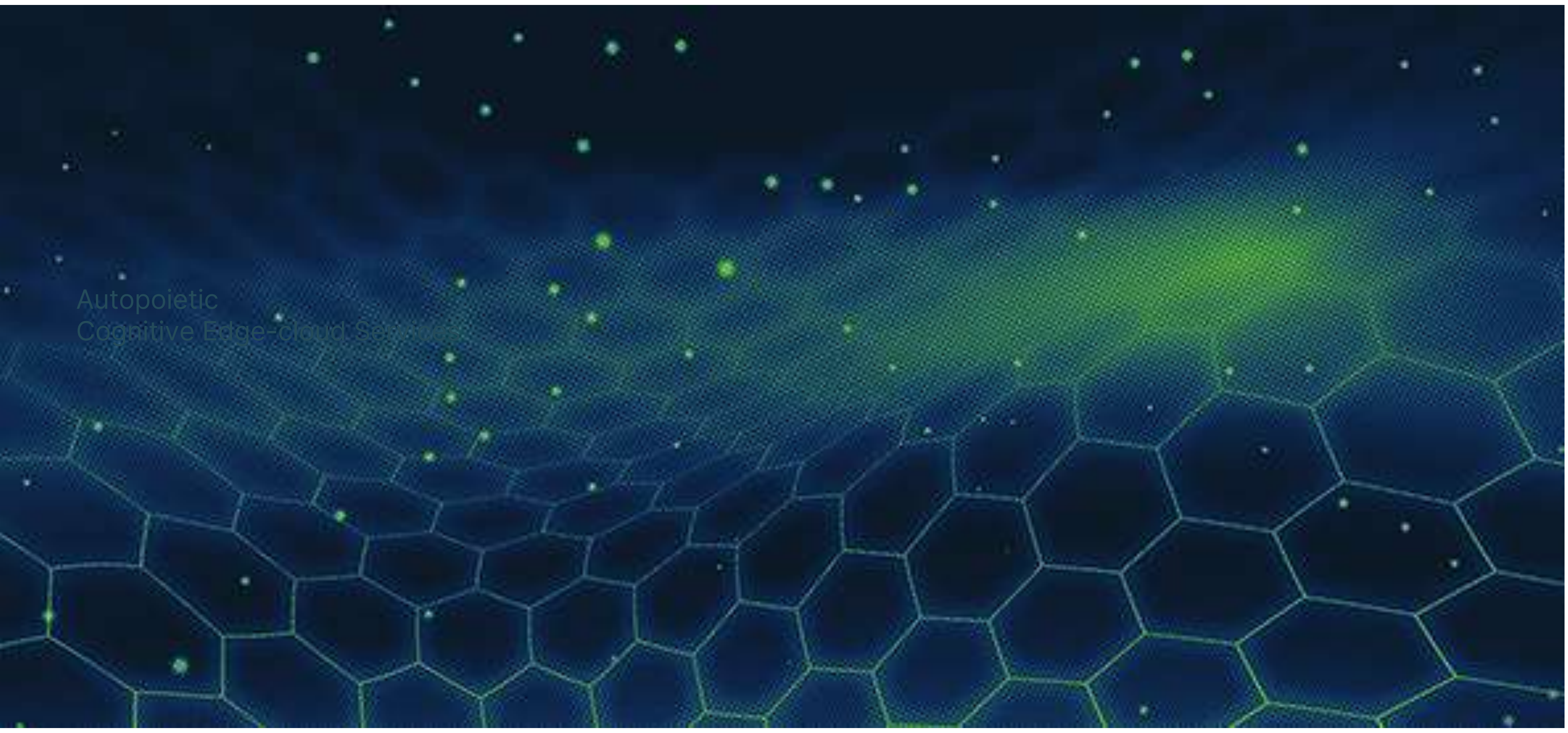

Deliverable 2.1 ACES Architecture Definition

Grant Agreement Number: 101093126

Autopoietic Cognitive Edge-cloud Services	
Project full title	Autopoietic Cognitive Edge-cloud Services
Call identifier	HORIZON-CL4-2022-DATA-01
Type of action	RIA
Start date	01/01/2023
End date	31/12/2025
Grant agreement no	101093126

Funding of associated partners

The Swiss associated partners of the ACES project were funded by the Swiss State Secretariat for Education, Research and Innovation (SERI).

D2.1 – ACES Architecture Definition	
Author(s)	Yiannis Georgiou, Fred Buining, Rui Min, Melanie Schranz, Fernando Ramos, Cláudio Correia, Thien Duc Nguyen, Félix Cuadrado, Loris Cannelli, Theodoros Grigorakakis, Nikolaos Kanakaris, Timotej Gale, Melanija Vezocnik, Nabil Abdennadher
Editor	Yiannis Georgiou
Participating partners	HIRO, LAKESIDE, INESC ID, TUDa, UPM, SUPSI, IPTO, UL, MARTEL and SISXQ
Version	1.0
Status	Completed
Deliverable date	M12
Dissemination Level	PU - Public
Official date	31 December 2023
Actual date	29 December 2023

Executive Summary

The ACES project aims to develop a highly decentralised autopoietic and cognitive framework for edge-cloud computing built around AI/ML and swarm intelligence. This deliverable outlines the ACES architecture and its key components.

The document starts by presenting its motivating use cases, elucidating how ACES supports its edge computing scenarios. The use-case requirements delve into the nature of data and knowledge, its decentralised storage, networking, scalability, and security requirements, providing a comprehensive understanding of the challenges ACES aims to address.

An in-depth exploration of the ACES architecture follows, presented from different angles: functional, component, tools, and hardware. This architectural overview highlights the key features and innovations of the project, offering a high-level view of how ACES can reshape edge computing by integrating cutting-edge technologies and concepts.

The subsequent chapters delve into the core aspects of the project, covering resource management, data management, orchestration, networking, monitoring, cognition, and security. Each section provides insights into these domains' background, design requirements, and innovations, showcasing ACES' commitment to addressing some fundamental challenges in the edge-cloud continuum context.

Finally, the deliverable explores the integration design of ACES components, interfaces, and interactions to provide a holistic view of the framework.

Disclaimer

The ACES consortium consists of the following partners:

No	PARTNER ORGANISATION NAME	ABBREVIATION	COUNTRY
1	INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC ID	PT
2	HIRO MICRODATACENTERS B.V	HIRO	NL
3	TECHNISCHE UNIVERSITAT DARMSTADT	TUD	DE
4	LAKESIDE LABS GMBH	LAKE	AT
5	UNIVERZA V LJUBLJANI	UL	SI
6	UNIVERSIDAD POLITECNICA DE MADRID	UPM	ES
7	MARTEL GMBH	MAR	CH
8	SCUOLA UNIVERSITARIA PROFESSIONALE DELLA SVIZZERA ITALIANA	IDSIA	CH
9	INDIPENDENT POWER TRANSMISSION OPERATOR SA	IPTO	EL
10	DATAPOWER SRL	DP	IT
11	SIXSQ SA	SIXSQ	CH

Document Revision History

DATE	VERSION	DESCRIPTION	CONTRIBUTIONS
08/10/2023	0.1	Table of contents	Rui Min (HIRO)
10/10/2023	0.11	Some changes to chapter structure (1 new chapter added) and first proposal for ownership	Fernando Ramos (INESC)
10/10/2023	0.12	Added requirements for the use cases	Theodoros Grigorakakis (IPTO)
10/11/2023	0.8	Added drafts for architecture diagrams	Yiannis Georgiou (HIRO)
12/11/2023	0.9	Updates in all sections	All partners
8/12/2023	0.95	Initial draft ready for review	Yiannis Georgiou (HIRO)
15/12/2023	0.98	Corrections	All partners
21/12/2023	0.99	Final version	Yiannis Georgiou (HIRO)
29/12/2023	1.0	Version after final review	Fernando Ramos (INESC)

Authors

AUTHOR	PARTNER
Yiannis Georgiou, Rui Min, Fred Buining	HIRO
Melanie Schranz	LAKE
Fernando Ramos, Cláudio Correia	INESC ID
Thien Duc Nguyen	TUDa
Félix Cuadrado	UPM
Loris Cannelli	SUPSI
Theodoros Grigorakakis, Nikolaos Kanakaris	IPTO
Rui Min, Yiannis Georgiou	HIRO
Timotej Gale, Melanija Vezocnik	UL
Nabil Abdennadher	SixSq

Reviewers

NAME	ORGANISATION
Loris Cannelli	IDSIA
Thien Duc Nguyen	TUDa
Fernando Ramos	INESC-ID

List of terms and abbreviations

ABBREVIATION	DESCRIPTION
ACES	Autopoiesis Cognitive Edge-cloud Services
AI	Artificial Intelligence
ML	Machine Learning
EMDC	Edge MicroDataCenter
SI	Swarm Intelligence
SDN	Software-defined Networking
QoS	Quality of Service
FL	Federated Learning
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
MEC	Mobile Edge Computing
TSO	Transmission System Operator
UC	Use Case

Table of contents

1 Introduction

Current edge-to-cloud architectures typically have a hub and spoke design, in which devices move their data to a distant datacentre (hub) with IaaS and PaaS management systems to manage and process the data with knowledge of workload resource requirements and their availability. The results are then pushed back out to the edge devices. Hyperscale and large datacentres use their economies of scale (incl. overprovisioning) and cognition in workload management to create stability and performance despite the fluctuations in the demand, and the availability of resources.

Edge infrastructures (FOG, MEC Edge MicroDataCenters) are more challenged in their stability and performance because of more stringent latency and autonomy requirements, distribution across multiple sites, their local limited size, multi-tenancy and multi-operators, local management, with components being concurrent and asynchronous.

This challenge to edge infrastructures is growing rapidly due to the increasing i) number of connected devices and their data-producing and data-consuming capabilities, ii) intelligence embedded in edge devices, iii) atomization of monolithic applications, iv) scale, speed, and complexity of edge device interactivity in a zero-trust environment.

The Autopoiesis Cognitive Edge-cloud Services (ACES) general aim is to research an evolution of cloud computing towards a hybrid edge-cloud continuum to effectively manage disaggregated computational resources while enabling the execution of complex modern data analytics applications. This is proposed through an edge systems software stack enabling decentralization and hierarchical intelligence, with specific autopoiesis and cognitive behaviours, to manage and automate a computing platform, network fabric, and storage resources, along with the analytics to be executed, to increase resilience while managing simultaneous service constraints.

Autopoiesis and cognition will be infused on different levels of the workload placement framework, service and resource management, data and policy management to manage their own stability with knowledge of its distributed component deployment, their state of health along with the knowledge of best practices to deal with fluctuations.

1.1 Objectives of the deliverable

The objective of the deliverable is to define the architecture of the ACES framework. This is expressed as a set of different views, namely the functional view, the components view, and the tools' view. To effectively design the architecture of ACES we brought forward the characteristics and requirements of ACES use cases which are used as the basis for our analysis and the definition of the different requirements and needs in terms of deployment and platform control. Based on this architecture a set of specific research areas and derived components are proposed and analysed. This list of research areas and components will be refined in the following months of the project while particular studies and developments will be started to address the different design specifications.

This document provides the generic architecture of ACES which will follow the project until the end. However, specific updates may be needed and if this is the case this will be reported in different intermediate deliverables such as the D2.2a ACES kernel components planned for M18, the D5.1 Validation and Demonstration plan planned for M19, along with the final deliverable of the project, D5.2 Use case Validation and Demonstration report, planned for M36.

1.2 Structure of the document

The remainder of this deliverable is as follows. The use case requirements are presented in Section 2 which are considered as inputs to create the architecture of ACES. The architecture along with the different views of the architecture and the initial concepts are then described in detail in Section 3.

Then, the various areas of interest of the ACES project are presented one by one in a different section. Section 4 discusses Resource Management, Section 5 introduces Data Management, Section 6 presents Orchestration and Section 7 brings forward Networking. Then, Section 8 presents Monitoring and Observability, Section 9 describes the ACES cognition framework, and Section 10 discusses Security and Privacy. Finally, Section 11 involves Integration Design, and Section 12 concludes this document.

2 Use-case requirements and related service design

The three use cases of ACES target the Energy sector. Companies from this sector operate in a complex environment characterized by restrictive regulations, increasing competition, geopolitical influences, and changing stakeholder expectations. To be able to respond dynamically to these changes, ensuring open, automated communication and real-time operation of the energy system, managers need to transform companies digitally. While the digital transformation of energy companies is inevitable, limited access to specific resources can slow down or, in extreme cases, prevent the process of digitalization [1].

On the other hand, climate change regulations like the Green Deal are pushing the current Power Grid Infrastructure to its limits. Electric vehicles and renewable energy sources have reduced emissions and energy costs but due to their distributed and inverter-based nature, they also decrease the power quality and pose serious threats to the power network stability. Sustaining the power quality and creating a network capable of operating in a highly distributed way creates a serious challenge for the Transmission System Operators (TSOs). In this highly anticipated and clean energy era, the vast penetration of metering and control devices all over the network has paramount importance but further observing the state of the grid is not enough to achieve our goal. Sophisticated algorithms gaining insight into the generated measurements and quickly responding to incidents will increase the robustness of the network.

Thus, the increase in produced and transmitted data puts a heavy burden on their respective computing and networking resources. This growth in produced data and the necessity to further process them form a new landscape for the energy sector. Centralised Cloud computing architecture although simpler to adopt falls short in many requirements of the future vision of a smart grid [2]. To complement cloud computing there is a need for a solution that meets the strict latency requirements in an efficient and distributed way to cover the vast geographical area of the power grid. Edge computing leverages computing resources closer to sensors and users to carry out data analytics. It gains advantages for its ability to effectively reduce system delay, lighten the burden of cloud computing centres, improve system scalability and availability, and protect data security and privacy [3]. The ACES solution follows this trend, enabling microgrid deployments and a highly distributed energy grid.

2.1 Use-case descriptions

The ACES platform aims to demonstrate its advanced competencies through three pilots in the energy sector. In this section we describe the Use Cases (UCs) that will be deployed in the Greek Power Transmission System.

UC1 – Market place and assets distribution

The Energy Market improves the efficiency of the energy power grid and provides lower prices for the electricity customers. The TSO in Greece (IPTO) must check whether the energy transactions that take place are feasible. In order to carry out this task, the topology of the (power) network is examined. Taking as input the network topology with its constraints, the supply and the demand, an optimal power flow algorithm give us a rough estimation. Secondly, prices and other characteristics like ramp-up time or minimum-power are taken into account to solve a Market Clearing Algorithm. Finally, the prices are settled and the amount of each generation and consumption at which price point is set. This process takes place every 15 minutes according to Greek Energy Market Regulations.

The output may also include power network information (voltage in buses, current flow in lines, generator production etc.). To assess the network dynamics, there are (power) load sensitivity algorithms that tests thousands of simulation scenarios and assesses how much load the network can withstand in each different part of the system. As long as the predicted load falls within the specified limits then we can be confident that the voltage will also remain within the desired limits ensuring constant power supply to the consumers.

UC2 - Distributed process management

The penetration of Distributed Energy Resources (DERs) requires more fine-grained monitoring and control of the grid. As with the previous UC the first stage of data processing stems from optimal power flow. The input may vary from SCADA to PMU data. In combination with the network topology characteristics, many insights can be given through processing. Feeding these data into machine learning algorithms can provide both present and future estimations on the Power Grid. ML Health index algorithms provide an estimator for the state of the topology under monitoring. Enhancing this algorithm with rule-based processing of the data can constitute a Digital Twin of the Network.

Proactive Automatic Generator Control (AGC) simulations can assist in proactively adjusting power production, which is especially useful for production units that require lead time before changing their production levels. Monitoring of data gives us leverage to establish alarms and set thresholds further aiding the AGC units that are registered in the network.

Demand prediction algorithms calculate the expected demand in the upcoming timeframes (some hours ahead) which can solve the power flow ahead of time. The calculated generator contributions could be used to control the generator production ahead of time increasing the stability of the network.

UC3 – IoT-based asset monitoring and management

Power Network operators have periodic planning cycles for assets-maintenance. Periodic on-site inspections can be replaced by advanced metering, sensor data and GIS systems for real time outage detection, prediction, and more reliable investment planning and deferral.

Taking as input SCADA data, the ML Health index Anomaly Detection algorithm identifies anomalies such as cases of low/high reactive power, high voltage instance etc. Anomalies could be used by the operators to analyse unexpected instances and design mitigation actions for the future, improving the health of the grid. The outputs of the algorithms include the list of anomalies that could trigger alerts for the operators.

The typical data that are expected to be used for UC 3 include SCADA data (Active Power, Reactive Power, Voltage and Current) in per minute values. Outputs includes list of anomalies that trigger relevant alerts for the operators.

2.2 How ACES supports the use cases

Massive volumes of data are generated every second in smart grids. Advanced data analytic algorithms are required to transform the data into information and knowledge, which can be further utilized for grid operations and services. Generally, these data analytics depend on information and communication technologies (ICTs), which perform a critical role in data collection, transmission, and processing. Among the major functions of ICTs, computing determines how grid data analytics are executed and thus, it becomes the foundation for grid operations and services. Centralized cloud computing prevails as a feasible solution for the grid computing paradigm. In cloud computing, geo-distributed devices and equipment are connected to cloud data centres, making centralized decisions, and issuing control orders. Nevertheless, it suffers from several weaknesses, such as limited bandwidth resources, heterogeneous environments, and privacy concerns. To tackle this problem, another solution, edge computing, pushes the frontier of computation applications away from centralized nodes to the communication network's extremes. Edge computing leverages computing resources closer to sensors and users to carry out data analytics. It gains advantages for its ability to effectively reduce system delay, lighten the burden of cloud computing centres, improve system scalability and availability, and protect data security and privacy [4].

Following this paradigm shift, the Digital Transformation in the Energy Sector aims to facilitate its seamless operation, further exploiting both communication and computation advantages to increase efficiency.

UC1 – Market place

The decentralised architecture of ACES is an enablement opportunity for decentralised market management in the energy network. Local simulations will run market algorithms for various parts of the network that will communicate with each other trading flexibility and network characteristics. The edge capabilities of ACES provide the opportunity for calculating power flows in the different edge servers in parallel as well as selecting the optimal allocation of the computational workload.

With the utilisation of a distributed edge computing architecture, we expect an improved performance of the marketplace algorithm as the optimal power flow simulations of the grids will be calculated in parallel across the various servers of the network. Additionally, we expect that we will be able to run additional and more complex modules within the 15-minute intervals which may prove useful in increasing the stability of the network.

Expected behaviour of the ACES platform in this UC:
- Seamless operation of the 3 markets (Attica, Crete, Cyclades)
- Upscaling of performance through the other nodes
- Expandability (adding extra regions)
- No single point of failure
- Safe (no disruption) and secure (data privacy) operation

UC2 - Distributed Process Management - Automatic generator control

Distributed Process Management can happen locally in the edge servers taking advantage of the decentralised architecture and its proximity to the data generation. The computationally expensive algorithms (Digital Twin and ML Grid Health) will run in parallel for different parts of the network and the outputs for each grid will be combined to cover the configurations of the whole energy grid.

Expected behaviour of ACES platform in this UC:
- Host the monitoring and control functions application in the platform
- Meet the latency requirements (seconds, sub-seconds) of the workloads
- Scalability
- No single point of failure
- Safe (no disruption) and secure (data privacy) operation

UC3 – IoT-based Asset Monitoring and Management

The decentralised architecture of ACES provides the opportunity for parallel calculation of ML grid health algorithms, for the various parts of the energy grid. Trained models can be transferred across the various edge components enabling the transfer of patterns learned from the data. Processing can happen locally satisfying latency and efficiency requirements as the data will not need to be transferred to centralised cloud systems for further processing but only for historical storage. This could enable sub-second performance in the future as the incoming SCADA data streams become more granular.
Expected behaviour of ACES platform in this UC:
- Host the monitoring and asset management applications in the platform
- Meet the latency requirements (seconds, sub-seconds) of the workloads
- Scalability
- No single point of failure
- Safe (no disruption) and secure (data privacy) operation

2.3 Use-case requirements

Operating in a Utility Industry, TSOs need safe and seamless operation of the power grid. The EMDC that will be acquired through the project will not completely replace the current infrastructure but will enable it to enhance the capabilities of a TSO to sustain and fortify its uninterrupted operation of the grid. The Use Cases demonstrated in the ACES platform should meet certain requirements along the following dimensions like benefits, security, availability, scalability, redundancy and latency. Those are described in the list below:

1. (For all UCs) The solution should provide availability of at least 99.9%.
2. (For UC1) The 3 market regions (Athens, Crete, Cyclades) should be fully operable in the ACES platform and have horizontal communication.
3. (For all UCs) There should be no single point of failure.
4. (For all UCs) The data used must remain private and secure.
5. (For all UCs) The solution should be scalable so that in the future we can add additional locations across the energy grid as well as include additional assets in the network grid.
6. (For UC1) When one EMDC fails other nodes must be used as hot swaps.
7. (For all UCs) The latency of communication should be close to sub-second to enable real-time responses.
8. (For UC1) the data should remain stored for at least 1 year.
9. (For UC2 and UC3) the data should remain stored for at least 3 months.
10. (For all UCs) The solution should be able to cater to diverse workloads, such as periodic, non-predetermined time intervals, or near-real-time workloads.

In Table 2.1 we describe the various use case components that will be available in ACES. We describe their type (periodic/occasional/continuous), we specify the frequency of running those components (real-time/every few hours/weekly, etc), and we indicate the timeframe for each component.

Table 2.1: ACES processes including type, frequency, and timeframes

PROCESS	TYPE	FREQUENCY	TIMEFRAME
Optimal Power Flow	**Periodic:** workload spaced at regular time intervals – static load	Seconds	Seconds
Load sensitivity analysis	**Occasional** workload with no (or very small) changes or fluctuations	Could run for each different load every time the specifications of the network change (addition/change of a generator / line etc)	Elastic
Market Algorithm -	**Periodic** workload spaced at regular time intervals – static load	Minutes-Run every 15minutes	Strict deadline
Optimal power flow using demand prediction input	**Periodic** workload spaced at regular time intervals – static load	Hours-Run every 7h and / or every 48h Run after demand prediction module is run	Elastic
ML Health Algorithm - Machine Learning Based Anomaly Detection	Training: Periodic workload spaced at regular time intervals – static load Scoring: Continuous scoring as SCADA data is retrieved	Days-Training: run every week Seconds-Scoring: run real time	Seconds (Inference) Elastic (Training)

Most Use Case components are considered periodic workloads spaced at regular time intervals. Their load is considered static and similar to the previous runs. The frequency of the runs varies from every 15 minutes for power flow simulations to every 1 or 2 months for the retraining of the demand prediction algorithms. For the Anomaly Detection algorithm, the scoring needs to happen near real-time as new data becomes available. There are also occasional workloads that occur in non-predetermined time intervals. For example, the load sensitivity analysis needs to run only when the system parameters change. Those workloads are not very frequent and not scheduled.

The consideration of the above use case requirements enabled us to design and refine the ACES architecture we present next, accordingly.

3 Overview architecture

This chapter provides the overview architecture of the ACES platform. This is defined by considering the use case requirements, as presented in the previous chapter; by aligning to the generic architecture of ongoing EU working groups and past or ongoing Edge-Cloud EU projects; and by considering the motivation to go beyond the state-of-the-art in aspects such as distributed control of disaggregated hardware resources, multi-cluster orchestration, decentralization, network programmability, and swarm intelligence. These aspects are mainly pushed by the increasing heterogeneity, complexity, and disaggregation of computational resources at the edge.

In the ACES context, we consider that each Edge MicroDataCenter (EMDC) will be composed of several independent clusters, to provide better fault tolerance for sensitive workloads while enabling resource segregation and the usage of different clusters for single applications through seamless executions. Different EMDCs may be distributed geographically, to address lower latency and fewer data transfers through data locality. On this basis, ACES enables the seamless execution of single applications even upon the clusters of different EMDCs (east-west) while the connection from edge to Cloud (south-north) is also supported.

Furthermore, the possibility to enable automated control and self-maintenance of the system, bringing autopoiesis capabilities for EMDCs along with the AI/ML lifecycle management and the combination of AI/ML with the edge is another important focus of ACES, which is reflected through its various architectural components and integration choices.

3.1 ACES concepts and background

Following a bottom-up approach, ACES concepts are represented by the need to support the resource management of disaggregated consumable resources at the edge. The de facto standard in resource management from edge to cloud is currently Kubernetes. Hence, for simplification and standardization purposes, we will adopt this tool. However, this imposes some architectural constraints such as the fact that complete decentralization will be a challenging task [5]. The control of multiple Kubernetes clusters, in the context of one or multiple EMDCs, can be performed, enabling decentralized control by adopting and improving techniques [6] proposed in the context of Kubernetes federation (deprecated since the end of 2023) and the various new projects which are inspired by it and continue in similar paths (Nuvla, Karmada, Open Cluster Management, etc). Following these techniques, we will be able to also provide the connection between edge and cloud data centres.

Figure 1.1 provides a high-level overview of the principal concepts of the ACES platform. In particular, at the bottom of the figure, we can see how the control of disaggregated resources will take place within one single Kubernetes cluster through the traditional centralized, hierarchical way, potentially adopting swarm intelligence scheduling policies; whereas in the higher layer, for the control of multiple clusters of one or multiple EMDCs, we opt for a fully distributed control of resources, bringing more innovation related to decentralization in all layers of resource management such as networking, storage, and scheduling based on swarms.

To allow this to take place the ACES software stack is composed of a number of innovative systems software. Following the bottom-up view in Figure 3.1, the Operating System, the Resource Management, and the Container Management Interfaces will allow for better control of the different hardware instances (network interfaces, persistent volumes, etc) as disaggregated resources through containerization and correct partitioning. This will be completed by the single cluster orchestration which, as mentioned before, will be performed using Kubernetes standard with extensions in the scheduling layer to allow swarm intelligence algorithms to take place. Previous work [7] showed very promising results but to the best of our knowledge, to this date, no real implementation of swarm-based scheduling has been implemented in Kubernetes.

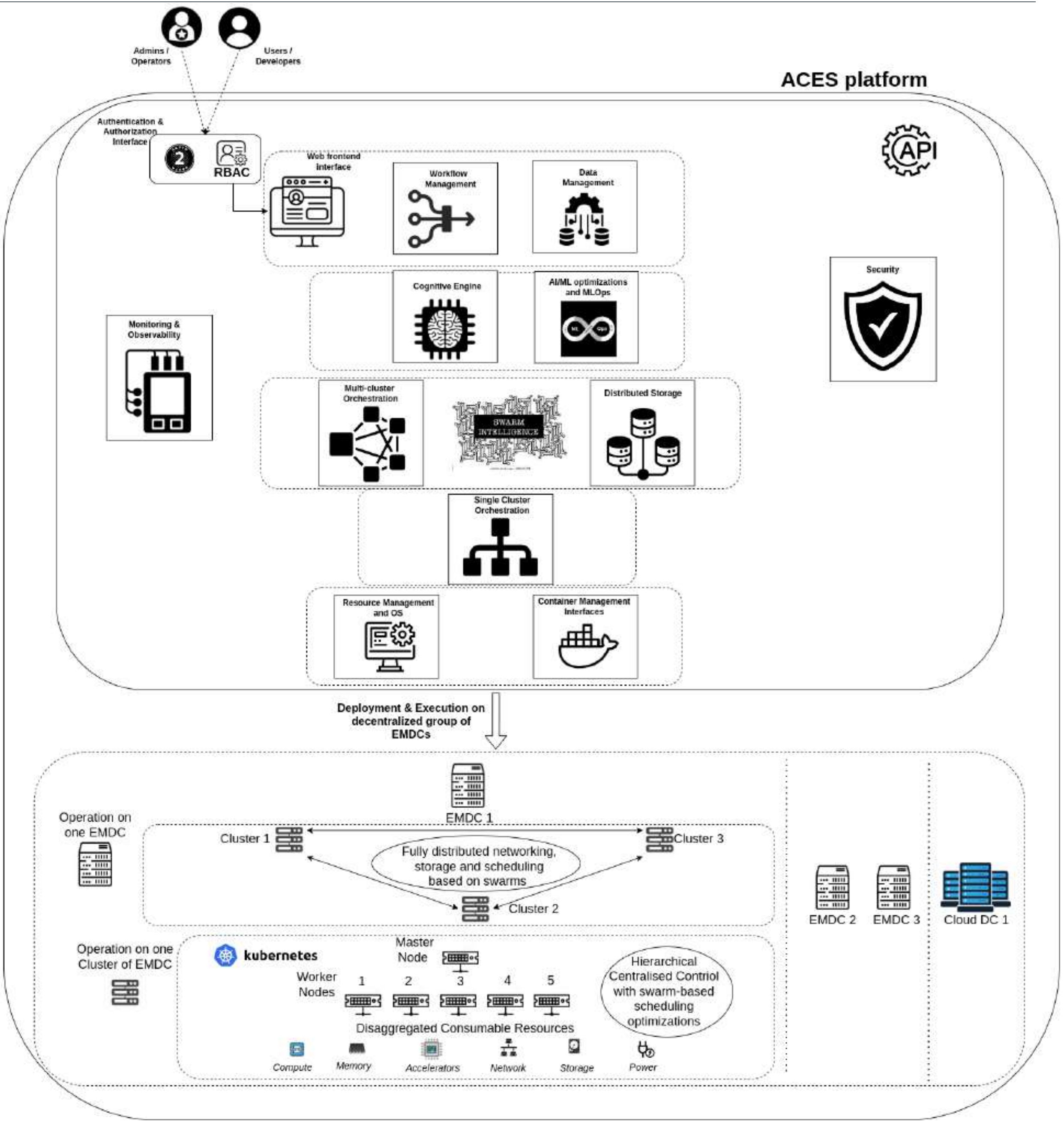
Admins / Operators
Users / Developers
ACES platform
Authentication & Authorization Interface
RBAC
Web frontend interface
Workflow Management
Data Management
API
Security
Cognitive Engine
AI/ML optimizations and MLOps
Monitoring & Observability
Multi-cluster Orchestration
SWARM INTELLIGENCE
Distributed Storage
Single Cluster Orchestration
Resource Management and OS
Container Management Interfaces
Deployment & Execution on decentralized group of EMDCs
EMDC 1
Operation on one EMDC
Cluster 1
Cluster 3
Fully distributed networking, storage and scheduling based on swarms
Cluster 2
Operation on one Cluster of EMDC
kubernetes
Master Node
Worker Nodes
1
2
3
4
5
Hierarchical Centralised Control with swarm-based scheduling optimizations
Disaggregated Consumable Resources
Compute
Memory
Accelerators
Network
Storage
Power
EMDC 2
EMDC 3
Cloud DC 1

Figure 3.1: High-level overview of the ACES platform

Moving higher, to enable the execution of applications across multiple clusters, we need software to enable multi-cluster orchestration along with decentralized storage and networking. These aspects need to be taken into account, and even if various techniques exist as open-source solutions, there are open research issues and possibilities for improvements, especially taking into account the disaggregation of hardware and the particularities of the applications.

An important aspect to be taken into account for optimal orchestration is the monitoring of resources and the observability of the application executions, as shown in the top left of the figure. These features will provide the necessary information and metrics to enable optimal resource selection and dynamic workload placement. Furthermore, security and privacy issues, at the top right of the figure, are primordial to be addressed in the context of ACES, including container, network, and ML security.

Moving towards the top of the figure, an important aspect to be taken into account is the Cognitive Engine, which will provide autopoietic capabilities, enabling self-maintenance and other self-* properties (detailed in Deliverable D4.2), particular enhancements to ML models for the edge, usage of edge computing advantages for ML models along with everything related to the ML lifecycle management of the platform. This engine will enable the techniques for MLOps for all the systems' internal AI/ML needs and will also offer the necessary services for application development.

To enable the design of applications to be executed on the platform, ACES will need to provide the right tools to allow users to prepare their applications. This is shown at the top level of the figure. For this to be feasible we opt for workflow/graph-based application design which is a typical way to enable the expression of scientific and data analytics calculations. The application design also needs to consider data management. Finally, a frontend interface connected to an authentication/authorization panel is needed to enable the connection, the permission controls, and the high-level view of the different services offered to users and admins.

3.2 High-level view of the architecture

Following the initial concepts and background, this section provides a high-level view of the architecture from different angles. First, we provide the architecture explaining the different functionalities that are needed on each layer. Then, we translate this to generic components, per layer. Finally, we describe the type of tools to be explored. These views are then completed by a closer look at the hardware architecture and the application structure.

3.2.1 Functional architecture

The functional architecture, presented in Figure 3.2, provides the physical layer with the hardware components and their different characteristics. On the cluster layer, we can see the Resource Management and Single-cluster Orchestration blue boxes representing both the node and single cluster level functionalities. The important aspects we expect here, besides the typical node resource management features such as cgroups, are related to the ways containers will be deployed upon the nodes, how persistent volumes will make use of different partitions of nodes' disks, and how networking will be virtualized to be used in the context of containers communication across nodes. Furthermore, in terms of orchestration, the way resources will be selected for particular workloads and the way tasks will be placed upon the resources, along with different constraints/SLAs/SLOs policy usage, are particularly important to better use of the underlying heterogeneous resources in the context of edge-cloud related platforms.

The multi-cluster layer, the green box in the figure, provides the multi-clustering tools, and how the different clusters will be connected and controlled to enable decentralized storage, P2P networking, and adapted distributed workload placement. These features will allow the connection of different clusters between one EMDC or more EMDCs, along with possibly the connection towards the Cloud.

On the left side of the figure, we can view the monitoring and observability box, which defines the need for collecting the metrics from different software and hardware per node, aggregated on the cluster level and per cluster aggregated for multiple clusters. On the right side of the figure, we express the needs for security and privacy spanning across all layers and features of the platform.

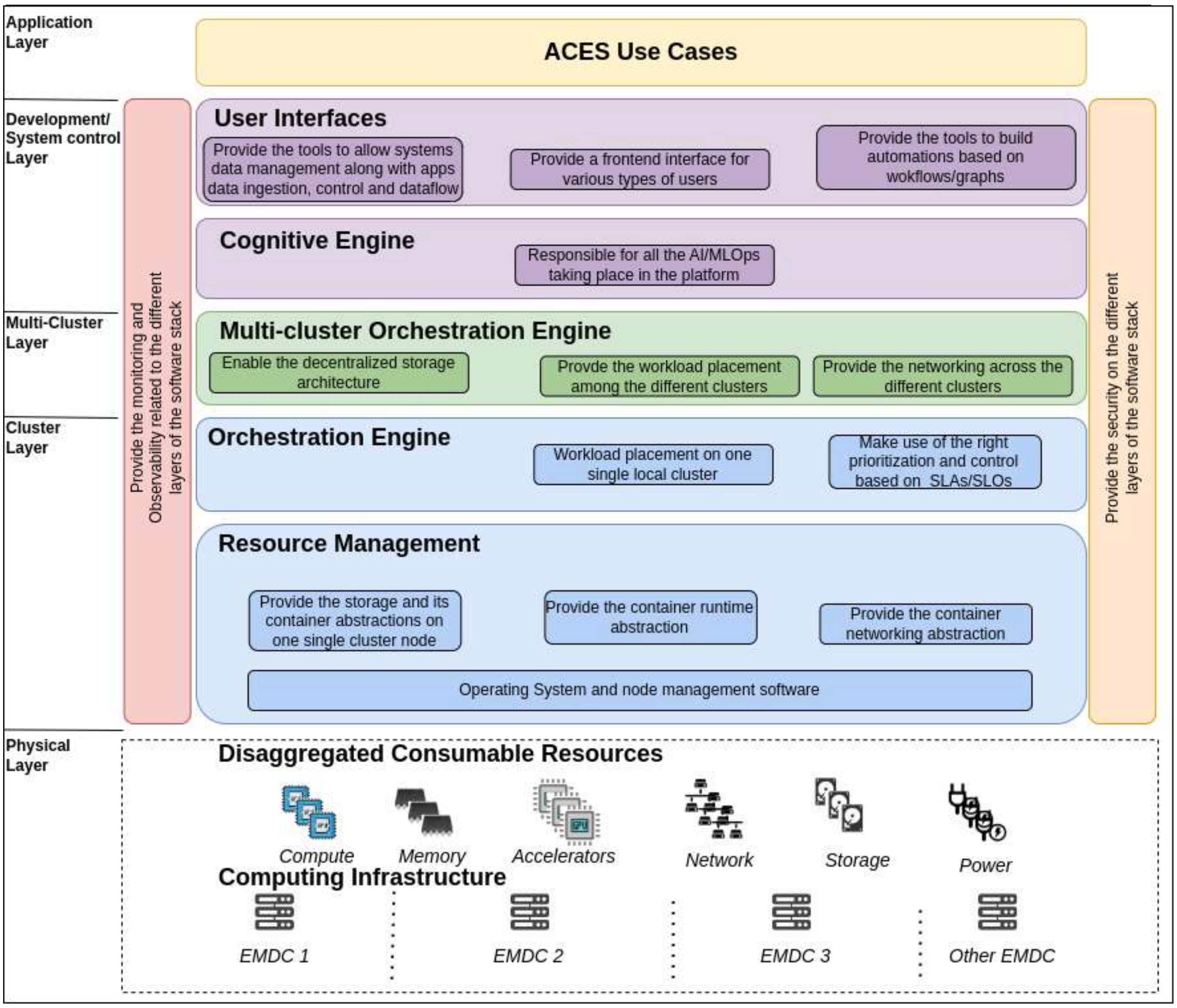
Application Layer
ACES Use Cases
Development/ System control Layer
User Interfaces
Provide the tools to allow systems data management along with apps data ingestion, control and dataflow
Provide a frontend interface for various types of users
Provide the tools to build automations based on wokflows/graphs
Cognitive Engine
Responsible for all the AI/MLOps taking place in the platform
Multi-Cluster Layer
Multi-cluster Orchestration Engine
Enable the decentralized storage architecture
Provde the workload placement among the different clusters
Provide the networking across the different clusters
Cluster Layer
Orchestration Engine
Workload placement on one single local cluster
Make use of the right prioritization and control based on SLAs/SLOs
Resource Management
Provide the storage and its container abstractions on one single cluster node
Provide the container runtime abstraction
Provide the container networking abstraction
Operating System and node management software
Provide the monitoring and Observability related to the different layers of the software stack
Provide the security on the different layers of the software stack
Physical Layer
Disaggregated Consumable Resources
Compute
Memory
Accelerators
Network
Storage
Power
Computing Infrastructure
EMDC 1
EMDC 2
EMDC 3
Other EMDC

Figure 3.2: ACES functional architecture overview

Moving higher at the layer of Development and System Control layer, the purple box, the Cognitive Engine expresses the needs to provide functionalities to enable the AI/MLOps of the platform, which will be used not only by the internals but also offered as a service to be used by the ACES users. Furthermore, this layer offers the higher level of user interfacing by expressing functionalities such as the frontend interfaces, the ways to build automations based on workflows, and the user and system level data management.

Finally, the application layer shows the level of ACES use cases which define how they will be expressed as ACES applications to be executed upon the platform.

3.2.2 Components-based architecture

Based on the previous Functional architecture, Fig. 3.3 provides the different components that are derived from the functional needs of the platform. There is a one-to-one mapping to the functionalities mentioned in the relevant boxes of the functional architecture. It is interesting to see how monitoring and security boxes are perpendicular to the different layers since they are related to all different aspects of the system software. This architecture focuses on the most important components of the ACES edge-cloud software stack and provides the principal areas upon which some innovation will be brought in ACES.

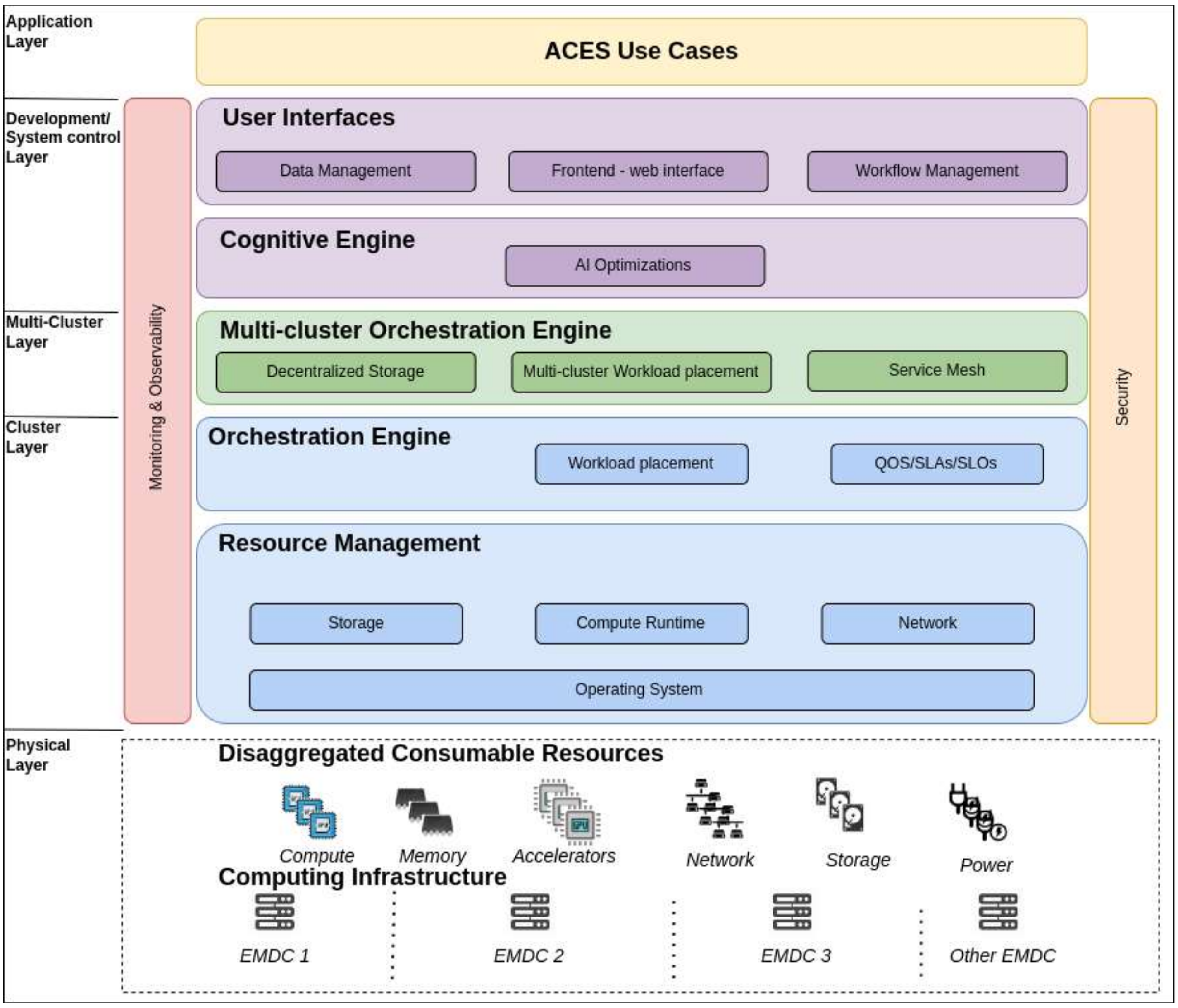

Figure 3.3: ACES components-based architecture overview

This component-based architecture diagram will be used as a reference moving forward in ACES, and an update of this architecture will be provided in the upcoming deliverable D2.2a.

3.2.3 Tools-based architecture

Based on the previous components-based architecture, Fig. 3.4 provides the tools-based architecture, which shows the view from the angle of possible tools to be used to cover the functionalities demanded by the components.

In particular, it is interesting to see that while Linux OS takes the place of the node operating system, Kubernetes brings the cluster operating system and covers both the resource management and the single cluster orchestration engine. Of course, different independent tools will be used to cover each functionality, such as containerd or CRIO to provide the container runtime interface, or Cilium for the Container Network Interface.

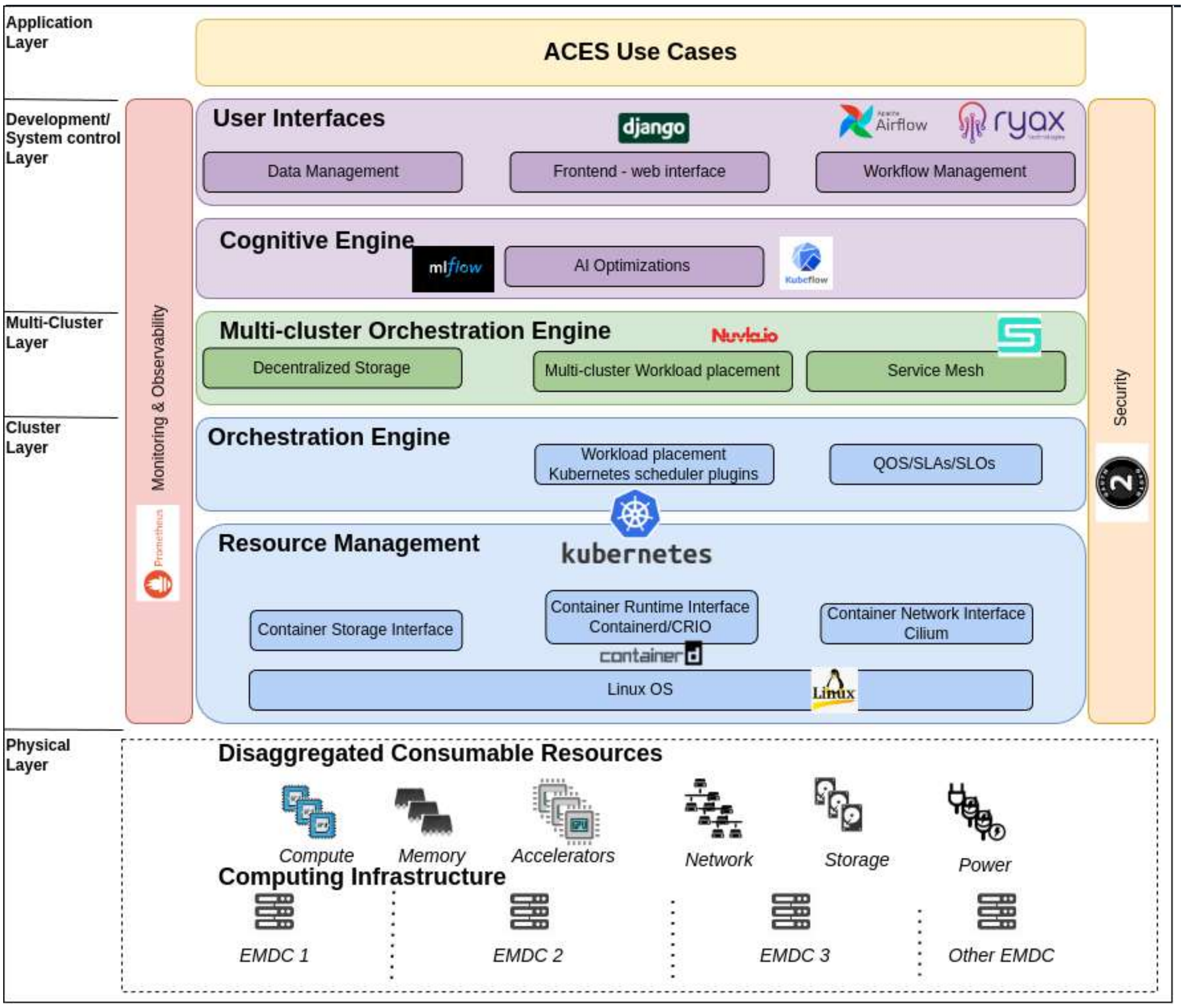

Figure 3.4: ACES tools-based architecture overview

In a similar way, workload placement will be brought by different Kubernetes schedulers. ACES will innovate by bringing specific swarm-based scheduling policies, as will be discussed in upcoming sections. Furthermore, multi-cluster control will be brought by the open-source software nuvla.io, which will be enhanced with swarm-based scheduling policies for workload placement. On the multi-cluster networking and service mesh side, tools such as Submarriner or Istio will enable container communications, in the case of workflows which span their execution across different clusters with different networks. Monitoring and observability will be based upon Prometheus, which will also cover multi-clustered aggregations through tools such as Thanos.

On the security side, various software will be used and enhanced in the context of ACES, as we will see in Chapter 10. For example, techniques such as OAuth2 will be used to enable the secure access to users on the platform. Several innovations will be brought forward in ACES in this space, including in-network security advances, increasing the robustness of ML training and inference, among others.

The above tools-based architecture shows the main guidelines, but the definition of the components and tools to be implemented will be refined in the upcoming deliverable D2.2a on ACES kernel components.

3.2.4 EMDC and hardware architecture

Technology providers are continuously innovating to reduce the complexity of tightly linked hardware components and trade-offs that need to be made because of these tight links. More loosely linked hardware components allow a wider application of swarm technologies. The recent technological innovations relevant to powerful edges are:

- Storage: NVMe (Flash storage) over Fabric, creates a low latency storage pool from all NVMe flash storage within an EMDC and within a network of EMDCs connected via WANs and LANs. This storage pool can be accessed by any server within the EMDC and EMDC network at low latency. NVMe/TCP is the latest addition to NVMe-oF and makes it possible to use NVMe-oF across a standard Ethernet network without having to make configuration changes or implement special equipment. (Note: the management of NVMe/TCP storage can be offloaded from the CPU to a DPU to free up the CPU for other work).
- Composable infrastructure with CXL: Disaggregated CXL 1.1 memory will ship with Intel Sapphire Rapids Xeon Scalable, AMD's fourth generation Epyc Genoa and Bergamo processors, and enables memory to be attached directly to the CPU over the PCIe 5.0 link. Vendors including Samsung and Marvell are already planning memory expansion modules that slot into PCIe like GPU and provide a large pool of additional capacity for memory-intensive workloads.

In light of recent and future technologies and given the limited local capacities of an EMDC, the scarce resources that need optimized utilization are, primarily, the various cores (CPU, FPGA, GPU, Custom ASIC) of the heterogeneous EMDCs. Second, memory and caching storage. Third, the internal network/ bandwidth. Additionally, the objective of ACES is to adopt CXL as much as possible and build this technology in edge data centres roadmap, and develop the intelligence to manage CXL efficiently.

Following the initial ACES concepts, Fig. 3.5 provides a zoom on the high-level hardware architecture, featuring the way we envision that EMDCs will be structured internally. On one side, by providing three different independent clusters connected in a fully distributed manner; on the other side, by different disaggregated hardware resources to be used in the context of a single Kubernetes cluster.

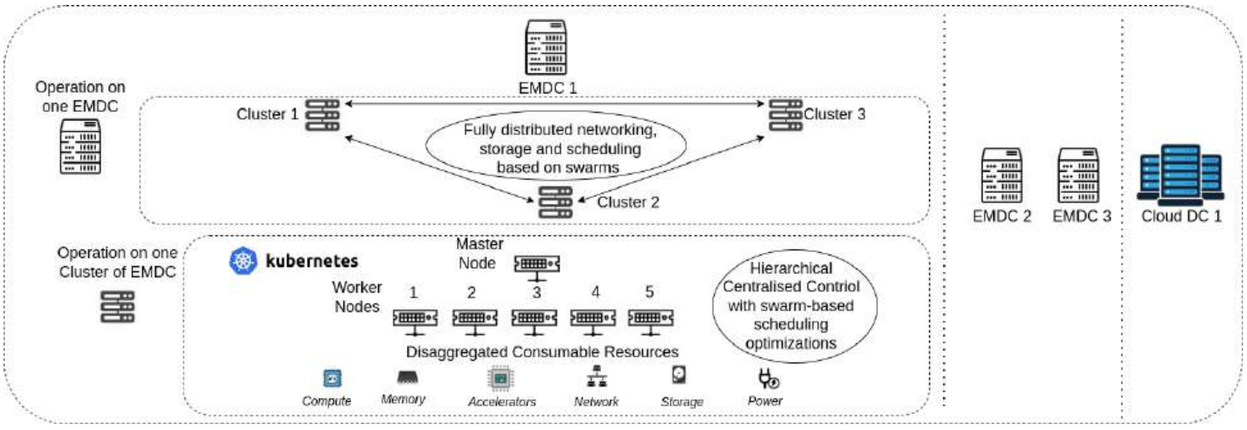
EMDC 1
Operation on one EMDC
Cluster 1
Cluster 3
Fully distributed networking, storage and scheduling based on swarms
Cluster 2
EMDC 2
EMDC 3
Cloud DC 1
Operation on one Cluster of EMDC
kubernetes
Master Node
Worker Nodes
1
2
3
4
5
Hierarchical Centralised Control with swarm-based scheduling optimizations
Disaggregated Consumable Resources
Compute
Memory
Accelerators
Network
Storage
Power

Figure 3.5: ACES tools-based architecture overview

The architecture will be similar on the other EMDCs. The connection between different EMDCs will be performed following similar techniques such as those for internal EMDCs.

3.2.5 ACES application structure

The applications in the context of ACES are represented by workflows, which are graphs representing a sequence of tasks or microservices to be executed upon the computational resources. Initially, simple sequence workflows will be supported, but if the use cases-related workflows evolve towards more complicated workflows (e.g., with loops or parallel phases) then this will need to be supported through the workflow management system. Figure 3.6 provides the structure of the typical ACES application, composed of different microservices which are connected through specific dependencies among them. This is the typical and most common way that data centre-based data analytics and scientific calculations are expressed.

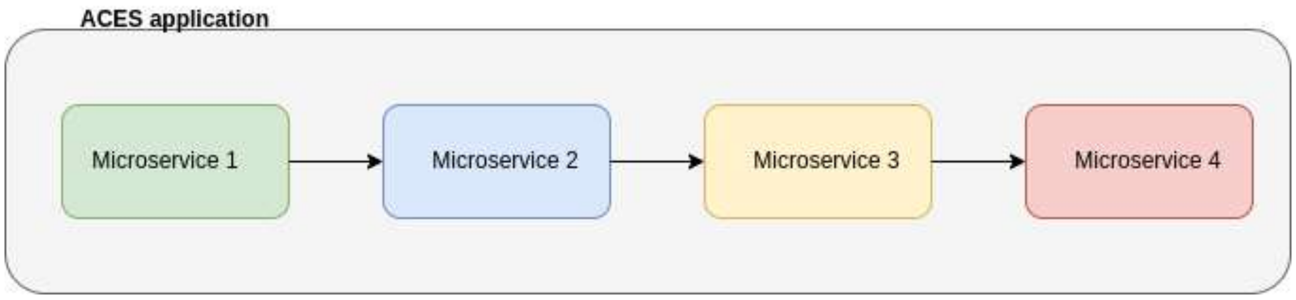

Figure 3.6: ACES application structure

Figure 3.7 provides the instantiation of the execution of an application upon the pool of consumable resources of the Edge Micro DataCenter. Each microservice will be allocated several consumable hardware resources such as CPUs, Memory, GPUs, etc., based on their needs, and once each microservice execution is performed its outputs are used as inputs for the following microservice to be completed. The dependencies between them can be defined following various patterns and based on the hardware architecture we provided previously; ACES will allow the execution of microservices to take place on different EMDC clusters. This will be enabled through the ACES multi-cluster networking capabilities.

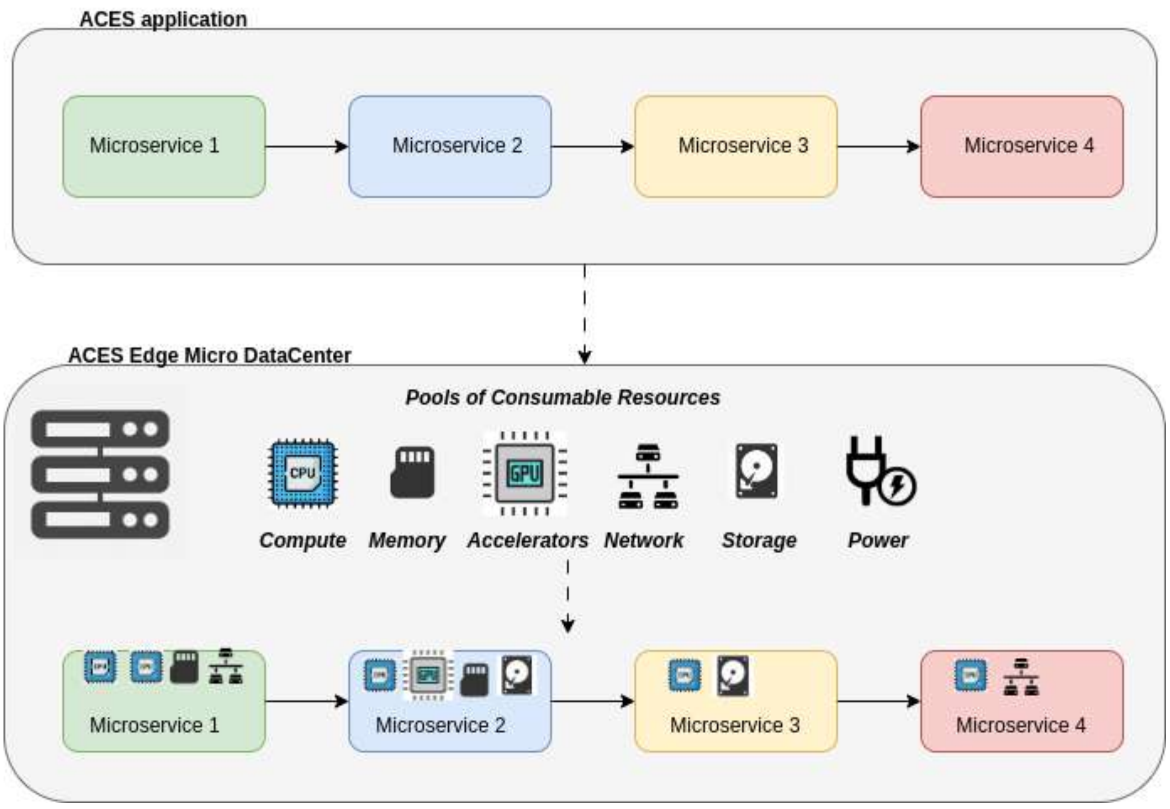

Figure 3.7: ACES application executed upon an EMDC

Since ACES adopts Kubernetes as the cluster operating system, we align to the concepts used in K8S. In particular, a microservice in ACES will be expressed by a K8S pod which will contain one or multiple containers executed upon one single node. Once the execution is instantiated, specific resources will be allocated to allow the pod to be executed correctly. Figure 3.8 provides the view of an ACES microservice along with the different internals, to better understand the connections among the different concepts.

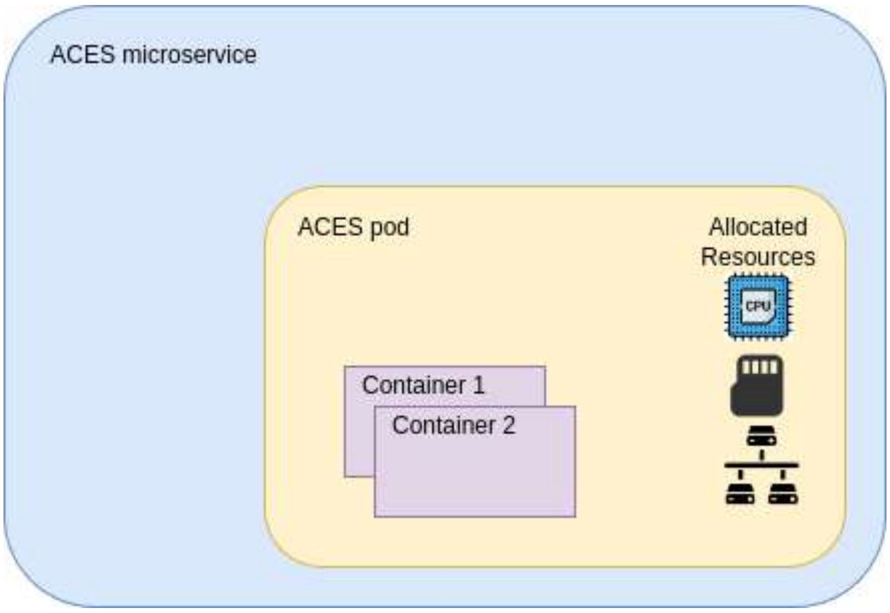

Figure 3.8: ACES microservice

3.3 Overview of features and innovations

In this section we present an overview of the features and expected project innovations, grouped by topic.

Resource Management

On the Resource Management side, ACES will bring features related to the optimal control of Edge Micro DataCenters single clusters through Kubernetes, adapted for the disaggregated resources and the workflow-based applications of ACES. This entails making use of specific Kubernetes adopted CRI, CSI, CNI tools and device plugins to cover the needs for resources heterogeneity and application complexity. Furthermore, it will offer control of multiple Kubernetes clusters by making use of tools beyond Kubernetes federation, enhanced with decentralization and adapted scheduling.

Concerning innovations, the decentralized control, and the implementation of swarm intelligence for scheduling on single cluster, along with high availability and scalability of the control-plane, and the optimization of autoscaling to better function at the edge, are some directions upon which the focus will be driven. Furthermore, in a multi-cluster setting, sophisticated scheduling optimizations are needed to manage deployments across clusters efficiently.

Data Management

The ACES data management component is critical for maintaining the operation of the platform, handling all the data required for orchestration and AI reasoning, which gives the system its self-managing qualities. It works with the ACES knowledge model, which encapsulates supply, demand, and runtime information, all essential to represent the architecture's resources, application requirements, and dynamic status. Key requirements for this component include handling various data types and relationships efficiently within a knowledge graph, accommodating time-series data, utilizing decentralized storage solutions for scalability and locality, ensuring data consistency, availability, fault tolerance, and security across geographically distributed environments.

With regard to innovations, the ACES data management component embodies a novel approach to distributed data handling that supports advanced cloud-edge computing environments. Notable is the potential use of knowledge graphs for representing data models, which can simplify complex information management and provide agility in querying and retrieving data. The emphasis on decentralized storage accentuates a cutting-edge model for data distribution that enhances scalability and preserves privacy. Another innovative perspective is the synchronization within a decentralized storage system, which aims to maintain consistency of the knowledge base without full replication, allowing data to remain local but universally accessible and up-to-date. Additionally, the seamless interplay with other ACES components, like swarm intelligence, highlights a dynamic and adaptive ecosystem built for high performance and resilience in cloud-edge orchestrations.

Orchestration

For the orchestration of workloads in an EMDC, we propose a decentralized approach that presents a key innovation: central to our approach is the use of swarm agents, representing demand and supply entities on different hierarchies. These highly decentralised agents collaborate within an EMDC environment, orchestrating processes such as workload placement, storage management, and caching optimization. The interaction between the agents is orchestrated through swarm intelligence algorithms. For example, demand swarm agents autonomously seek out the most suitable node for workload placement, while supply swarm agents determine the optimal workload to process based on available resources and capacity. This collaborative decision-making process enables the system to efficiently allocate workloads to nodes, optimizing processing, latency, and resource utilization.

Thus, in ACES we propose the edge continuum with its characteristics and limitations as a novel field of application for swarm intelligence leading to a distributed, emergent scheduler.

Networking

In terms of features, the ACES network architecture emphasizes intelligent, scalable, and secure networking within an edge-computing framework driven by a set of key requirements. It focuses on service connectivity with QoS prioritization, high network observability through advanced telemetry, and optimized throughputs and latencies for responsive edge computing capabilities. The design incorporates a closed-loop network control mechanism that harnesses machine learning analytics for dynamic network adaptability, ensuring that ACES can efficiently scale up as the edge infrastructure expands. It also features decentralized control to avoid single points of failure, thus enhancing network reliability. Moreover, integration with Kubernetes and programmable network switches aid in the management of multi-cluster networking, allowing for dynamic scaling and pod mobility to ensure service continuity across various edge environments.

Concerning innovations, ACES will leverage a software-defined networking (SDN) model, with the control plane dynamically orchestrating the multi-cluster network, integrated with machine learning to establish a closed-loop control system. The network emphasizes fault tolerance through replicated SDN controllers, scalability by distributing control functions, and security through AI-driven mechanisms. Furthermore, specific programmable data plane devices will be employed for network function acceleration, advanced telemetry, and executing swarm-based orchestration mechanisms directly on network devices, showcasing a shift towards utilizing in-network computation for enhanced edge intelligence. These data plane innovations offer an adaptable, high-performance network capable of sophisticated monitoring and real-time data analysis, as well as a more intuitive control over the complex interactions between network components and edge services.

Monitoring and Observability

The ACES monitoring and observability component will enhance the supervision of complex, decentralized systems with disaggregated hardware resources. It will include a storage component for time-series data, a Retrieval Worker for pulling metrics, and an API server for data retrieval. The architecture will incorporate a Push Gateway for alternative data ingestion from ephemeral workloads, a Data Forwarder for communicating with other ACES components or third-party tools, and an Alert

Manager for managing notifications based on anomalies detected by an Anomaly Detection submodule. The Service Discovery will automate the identification of targets for monitoring, while Data Analysis, Export and Visualization tools will facilitate the analysis and display of data. These components ensure that all ACES resources can be effectively monitored across the various layers, and types of software. The architecture will enable the handling of a variety of data such as traces, logs, and metrics while providing a framework for both push and pull data collection methods.

Concerning innovations, the ACES monitoring and observability framework will provide a particular handling of network data and event-driven monitoring. It may leverage on the computational capabilities of network devices such as programmable switches and SmartNICs/DPUs to perform fine-grained, flow-based metric computations directly within the data plane, facilitating richer telemetry data without traditional sampling limitations. This approach will enable network operation tasks, such as traffic engineering or intrusion detection, to be more accurately and promptly performed. The framework also plans to integrate event-driven monitoring through a novel change detector primitive using memory-efficient sketches, enabling rapid and space-efficient change detection in network traffic. This capability ensures the monitoring system can react responsively to significant changes in network conditions, in contrast to the fixed periodic monitoring intervals, while enhancing operational awareness and responsiveness within the ACES platform.

ML-based cognition

In terms of features, the Cognitive Framework in the ACES project offers a suite of advanced features to enable autopoietic capabilities in edge data centers (EMDCs), promoting self-maintenance and adaptive management of resources. It integrates continuous learning, predictive analytics, and machine learning (ML) lifecycle management, allowing systems to learn from the environment and user interactions for optimal performance. The framework's feedback loops allow for proactive, data-driven decision-making, aligning with broader organizational goals. Techniques such as federated and split learning support collaborative, privacy-preserving model training, while the integration of swarm intelligence with AI/ML enhances distributed resource management. Explainable AI (XAI) increases transparency and trust in automated decisions, and the MLOps component ensures efficient and standardized ML model lifecycle management across the platform.
In terms of innovation, the ACES platform introduces novel approaches such as employing autopoiesis for self-sustaining systems, leveraging edge-specific, lightweight ML models that navigate computational and latency constraints, and embracing collaborative methodologies like federated learning for data privacy. The combination of AI with swarm algorithms exemplifies a cutting-edge computational method for optimizing distributed systems, and integrating XAI provides transparency in automated decision-making. The platform's MLOps strategies reflect a forward-thinking approach in managing the complex and evolving landscape of ML models, emphasizing efficiency, reuse, and consistent monitoring throughout the ML model lifecycle.

Security and Privacy

The main features related to security include a multi-layered security and privacy architecture designed to protect cloud-edge services. Key focus areas include robust authentication processes with anonymous schemes and pseudonyms, ensuring the availability and integrity of services through fault-tolerant replication and auditing tools, network and hardware security using ML-based attack detection and innovative network switches, node and container security with advanced anomaly detection models and defence strategies, and machine learning security to counter data/model poisoning and inference attacks. The system emphasizes compliance with strict privacy regulations like GDPR and incorporates components like zero-trust environments, non-revocation proofs, and secure two-party computation to safeguard data privacy.

The main innovations in ACES revolve around enhancing security in cloud-edge systems against sophisticated cyber threats. Noteworthy advancements include the development of anonymous authentication methods that comply with privacy laws and accommodate non-revocation proofs, and an innovative auditing tool that uses cryptographic proofs to pinpoint data locations with high precision. The incorporation of an ML-based attack detector that operates within a network switch stands out as a performance-centric measure to detect zero-day attacks. Furthermore, the container security framework employs deep-learning-based dynamic anomaly detection to protect against new threats, and sophisticated strategies to defend ML systems, like model clustering and secure two-party computation, showcase novel approaches to fortify systems against complex attacks such as backdoors and inference breaches in distributed environments.

4 Resource management

Resource management holds a very important place in the software stack of distributed systems since it is responsible for providing the necessary computing power to user jobs based on their needs and the resources availabilities. The advent of Cloud and Big Data systems along with the usage of microservices and containerization brought the needs of environment provisioning and auto-scaling. Hence, the management of applications' lifecycle orchestration became an integrated part of resource managers. Traditional HPC resource managers such as Slurm and PBSPro do not provide integrated support for environment provisioning and hence no orchestration is feasible. However, state-of-the-art resource managers such as Mesos, Yarn and Kubernetes enable the deployment of containers and allow the applications' lifecycle management.

4.1 Background and principal concepts

Some studies on orchestrators discuss the various advances made in scheduling [8]. Kubernetes [9] and Mesos [10] are two of the most advanced open-source orchestrators. Kubernetes performs the resource management of a cluster of computational nodes and simplifies the deployment and management of containerized applications. It is based on a highly modular architecture which abstracts the underlying infrastructure and allows internal customizations such as deployment of different software-defined networking or storage solutions. It supports various Big Data frameworks such as Hadoop MapReduce, Spark, and Kafka and has a powerful set of tools to express the application lifecycle considering parameterized redeployment in case of failures, auto-scaling, state management, etc. Furthermore, it provides advanced scheduling capabilities and the possibility to express different schedulers per job.

Kubernetes orchestrator enables the support of Software Defined Infrastructures and resource disaggregation by leveraging on container-based deployments and particular drivers based on standardized interfaces (Container Runtime Interface [11], Container Storage Interface [12], Container Network Interface [13] and the device plugins framework [14]). These interfaces enable the definition of abstractions for finer-grain control of computation, state and communications in multi-tenant environments along with optimal usage of the underlying hardware resources. Open-source solutions such as k3s [15] where Kubernetes heavyweight internal procedures have been stripped down are more adapted for the edge. Another open-source alternative that could be interesting for the deployment of individual autonomous edge resources is Canonicals' microk8s [16] which can be evaluated for the mobile edge resources case, needing to orchestrate tasks and workloads autonomously when disconnected from the network. In a similar way, the multi-cluster special interest group (SIG) community of Kubernetes has been working on the federation v2 project [17] on integrating multiple clusters under a federation while providing a generic scheduling engine that, based on policies, is able to make decisions on how to place arbitrary Kubernetes API objects. While this project has recently been deprecated, systems such as Nuvla, Karmada and Open Cluster Management enable features of the federation which, along with networking techniques such as Submarriner and Istio systems, can enable the execution of applications across multiple clusters. An interesting solution combining edge system and multi-cluster control is provided by Oakestra [18], a hierarchical, lightweight, flexible, and scalable orchestration framework for edge computing. Oakestra features a federated three-tier resource management, delegated task scheduling, and semantic overlay networking proposed as an alternative for Kubernetes.

Asuncao et al. [19] studied resource management challenges regarding hybrid deployments including IoT and Edge. They consider that managing task scheduling and allocation of heterogeneous resources along with adapting an application to current resource and network conditions will require the development of new schedulers and that allocations have to be dynamic enough to support migration.

Resource management in a container-based cluster environment is a complex but essential aspect of ensuring that applications and services deployed across the cluster perform optimally and reliably. Containers have emerged as the standard unit of deployment in edge-clouds; hence, managing the resources they consume (such as CPU, memory, storage, and network bandwidth) becomes crucial for both the stability and efficiency of the entire system. Containers are lightweight, usually ephemeral, and they often run in dense multi-tenant environments. Due to their transient nature and the dynamic workloads they typically support, effective resource management strategies must be used to address the varying demands of different applications while utilizing the underlying infrastructure resources efficiently.

Kubernetes is the de facto standard for Cloud or Edge resource management and container orchestration, with a rich set of features for managing all types of heterogeneous resources. It automatically schedules containers based on resource requirements and availability, and handles the lifecycle of containers, enabling the configuration of resource limits and requests for containers, while providing auto-scaling and self-healing capabilities. In addition, it provides storage and network orchestration allowing storage to be managed dynamically while defining how services communicate among them.

ACES will adopt Kubernetes and its APIs to control the resources on a single cluster level as shown in the high-level view of the architecture. The resource management will be performed considering 3 main blocks or resources: Computing, Storage and Networking. Each of these plays a vital role, and interfaces like Container Runtime Interface (CRI), Container Storage Interface (CSI), and Container Network Interface (CNI) help abstract and manage these resources efficiently.

4.1.1 Runtime

Runtime resources in Kubernetes entail managing the lifecycle of containers and ensuring they have the necessary compute resources to function, such as CPU and memory allocations. The orchestration of these containers is crucial to the smooth operation and scaling of applications. The CRI abstracts the container runtime from the kubelet (the primary agent that runs on each node), allowing Kubernetes to use different container runtimes without requiring integration into the Kubernetes codebase. It defines a set of RPC calls for functionalities such as container and image operations. The kubelet uses CRI to manage container lifecycle events like starting and stopping containers, as well as handling their resource usage by adhering to the specified resource limits and requests.

4.1.2 Storage

Kubernetes storage management is related to the provisioning, attaching, and managing of the lifecycle of persistent storage used by applications. Automating these tasks and providing high availability of data is essential in cloud-native environments. CSI standardizes and abstracts the way storage providers interact with Kubernetes clusters. Through the standard API, the different storage plugins provided allow Kubernetes to work with a wide array of storage solutions. CSI facilitates volume provisioning, de-provisioning, mounting, unmounting, and snapshot operations initiated by Kubernetes. This enables Kubernetes to manage persistent data storage across different storage providers in a unified manner.

4.1.3 Networking

The Network management in Kubernetes ensures seamless container-to-container communications, Pod-to-Pod communications across different nodes, and external access to services running inside the cluster. A reliable and secure networking setup is fundamental to distributed applications' architecture. The CNI abstraction provides a standard for configuring network interfaces for Pods. By using a plugin-based architecture, CNI allows for various networking solutions that can be used with Kubernetes, which enables it to support different network models in alignment with organizational needs and policies. CNI plugins manage the assignation of IP addresses to Pods, set up network routes, and configure network namespaces. This allows Kubernetes to abstract the complexity of the underlying network topology and operations from the users and operators of the cluster.

4.1.4 Linux OS

Kubernetes makes use of cgroups and namespaces, Linux kernel features that enable the isolation and allocation of the CPU, memory, block I/O, and network resources for containers. Furthermore, it provides different security contexts which allow the definition of privilege and access control settings for Pods and containers, leveraging underlying Linux security features like SELinux, AppArmor, and seccomp. Finally, the Kubelet agent which runs on each Kubernetes worker node monitors its consumption and availability.

4.2 Resource selection and workload scheduling

In the ACES context we opt for decentralization. Thus, a possible direction would be to completely decentralize Kubernetes as studied in [5]. However, enabling this in Kubernetes is hard, with arguable value at this stage. A better technique has been adopted by the researchers of the Oakestra system [18], who introduced a new resource manager and orchestrator inspired by Kubernetes but built from the ground up for the edge. This study highlights the advantages in terms of scalability at the edge, when segregating the disaggregated resources across a larger number of clusters and enabling the deployment of applications across different clusters. This actually justifies and motivates further our design to opt for multiple clusters in one EMDC.

Even if Oakestra seems a very adapted solution for the edge, in ACES we will use Kubernetes as a single-cluster resource manager, since we are interested in the standardization capabilities. However, our goal is to adopt swarm intelligence algorithms to perform resource selection and workload placement. Related work [20] demonstrated impressive speedup; however, these were limited to simulation-based implementations and analyses. To the best of our knowledge, as of the present date, there has been no tangible realization of swarm-based scheduling within the Kubernetes framework.

Our approach entails collecting different historical and real-time monitoring data related to the EMDC resources, and combining them with the applications and microservices requirements, to perform the resource and workload matching. We give some detail on the swarm-based policy we plan to explore in Chapter 6, and how to integrate this policy as a scheduling plugin within Kubernetes.

4.3 High availability

High availability for the Kubernetes control plane is primordial to guarantee continuous management of cluster resources, even in the event of individual component or node failures. Achieving such reliability requires a specifically designed architecture where the critical control plane components—such as the kube-apiserver, etcd datastore, kube-controller-manager, and kube-scheduler—are deployed in a highly available configuration. This can involve running multiple replicas of each component across several nodes to mitigate the risk of correlated outages.

In the context of Kubernetes single cluster resource management, the API server acts as the gateway to the control plane, managing and persisting the state of the cluster in the etcd database. Etcd must also be reliably replicated and consistently maintained. A high-availability setup often includes a load balancer that directs traffic to the API server instances evenly, ensuring that the loss of a single server does not prevent access to the control plane. The controller-manager and scheduler, even if they are less stateful than the API server and etcd, also need replication to provide resilience against individual process failures.

When deploying a high-availability control plane in a decentralized edge computing environment, the complexity increases due to broader physical distribution and possibly variable network conditions. Research in this area requires innovations to ensure that the distributed control plane components efficiently reach consensus while respecting the latency and bandwidth constraints typical of edge networks. This may involve developing lightweight consensus protocols that are optimized for edge conditions. Another possible direction is to maintain a coherent and synchronized state across distributed etcd clusters, possibly by employing novel data replication techniques or tailored consistency models through eventual consistency with an acceptable convergence time, considering edge constraints.

4.4 Scalability and performance

Kubernetes excels in managing containerized applications with its built-in scalability features. The platform is designed to scale not only in terms of handling more workloads with the addition of more nodes and pods but also in its control plane capabilities which oversee the operation and management of the cluster itself. The API server, scheduler, etcd, and controller manager all play pivotal roles in the control plane's handling of the increasing scale. Scaling out the API server instances and etcd cluster (which stores all Kubernetes cluster data) can help maintain performance as the scale of the system increases.

Kubernetes allows applications to scale through the use of ReplicaSets, Deployments, and StatefulSets. It can efficiently manage the desired number of pod replicas to handle workload demands. However, the actual performance depends on the capacity of the underlying nodes and how they are managed in terms of pod scheduling and network configuration.

Autoscaling is another crucial aspect where Kubernetes uses the Horizontal Pod Autoscaler to adjust pod counts based on specific metrics and the Cluster Autoscaler to add or remove nodes based on the needs of the workload. Networking becomes even more important as the number of services and interactions between components grows. The storage system must also maintain scalability, ensuring data remains consistent and available during scaling operations.

Performances are directly impacted by how well all these scalability features function and by the underlying infrastructure's performance characteristics. This becomes apparent when considering the unique challenges presented by decentralized edge environments, where Kubernetes is expected to operate across distributed nodes that may be geographically dispersed.

In the context of ACES, the goal will be to enable managing large numbers of smaller, more distributed clusters while still maintaining performance and reliability. This can be done by optimizing the control plane components for operation across widespread and potentially unreliable networks.

Another important research direction is related to advanced autoscaling algorithms to enable flexible adaptation to the dynamic conditions at the edge. Based on historical data on resources monitoring and workload profiling, specialized ML-enhanced techniques may anticipate workload changes and predict the need for autoscaling to improve the turnaround time of jobs and the performance of the system.

4.5 Multi-Cluster control and scheduling optimizations

Multi-cluster control involves overseeing multiple Kubernetes clusters as coherent parts of a larger computational resource pool. This requires a unified control plane capable of harmonizing operations, sharing critical configurations, synchronizing deployments, handling failover, and facilitating resource sharing across clusters.

In the ACES context, each EMDC will be composed of a number of Kubernetes clusters. Applications will have the ability to be executed upon the resources of multiple EMDCs, hence the coordinated control of multiple Kubernetes clusters is an important aspect. KubeFed has been traditionally the technique promoted by the Kubernetes community for this purpose, but since its deprecation beginning of 2023, the community has driven its focus on various open-source tools similar to KubeFed that try to solve various issues of multi-clustering. Tools such as nuvla, karmada, and open cluster management for the high-level control and execution of jobs on multiple Kubernetes clusters, or submariner and Istio for multi-cluster networking, are tools that will be explored and enhanced in the context of ACES.

Techniques such as the ones proposed by Larsson et al. [6], enabling decentralized control in a federated multi-cluster setting can be adopted and enhanced with tools such as nuvla.

In more detail, scheduling in a multi-cluster environment is a complex task. The scheduler needs to have a global view to make optimal decisions about where to place workloads. It needs to take into account a range of factors beyond what is typically considered in a single cluster, including geolocation of clusters, network latency, data locality, edge node capacities, cluster-specific policies, and possibly different optimization objectives.

Optimizations for multi-cluster scenarios are essential for improving the overall efficiency and performance of Kubernetes at the edge. Enhancements in this area could involve the creation of more intelligent scheduling algorithms, methods for more efficiently managing a large number of small clusters, and improved paradigms for cluster federation.

Creating advanced scheduling strategies that can effectively allocate workloads across multiple clusters needs to consider cluster load, real-time resource availability, and network topology for optimal workload placement near the data sources, to minimize latency and maximize performance. ACES brings forward a completely decentralized setting across the different Kubernetes clusters, whose goal is to make use of swarm intelligence algorithms to perform the selection of resources and workload scheduling. In ACES, the development of the actual multi-cluster scheduling algorithm will be explored in the context of Orchestration (Chapter 6), but the development of the components to perform the scheduling in the multi-cluster setting will be part of the resource management component.

Finally, another important direction considering multi-clustering are the data management approaches to be used for the control plane of multiple Kubernetes clusters. This control plane needs to be decentralized and has to maintain consistency and availability of data across multiple Kubernetes clusters at the edge. Research must consider data replication, partitioning, and synchronization to provide seamless data access despite the geographical distribution of clusters. Hence, a close collaboration with Data Management (Chapter 5) is needed to effectively address the issues related to decentralized control of multiple Kubernetes clusters.

5 Data management

The data management component is in charge of managing all the internal data and information that needs to be persisted by the ACES platform in order to run its operation smoothly. This component provides a core part of the platform backend, ensuring the knowledge base is kept for every ACES cluster and EMDC.

5.1 Managed data

The data management component will be responsible for managing all the data that will be relevant for the orchestration and AI components to perform their reasoning and provide autopoietic qualities to the overall system. This includes all information relevant to be captured as features, or characteristics algorithmically employed. A complete overview of the information required by these elements, referred to as the ACES knowledge model, is presented in D3.1. In contrast, there will be no application storage being managed. These data elements associated to specific deployed elements will be managed as part of the orchestration component, as described below.

The ACES knowledge model provides information about the supply, demand and runtime aspects of ACES. The supply represents the fundamental concepts that will be used by the agents to reason about the ACES platform elements. This includes the architecture components, physical and logical entities presented in the environment. The demand model captures the applications that need to be presented in the environment, deployed on Kubernetes. The model includes all additional information regarding their requirements to function correctly, as well as the SLOs and other non-functional characteristics that must be satisfied. Finally, the runtime model expands these concepts with temporal information about the events and status of each element of concern. This information directly feeds from the monitoring sources that are managed in that component.

5.2 Design requirements

The data management component is designed based on several requirements that shape its role, structure, and underlying technologies. We briefly state here the main requirements that were specific to this component of the architecture.

5.2.1 Nature of data

As described in D3.1, the model will be represented as knowledge graphs following the property graph specification. There are multiple formats for representing and querying these structures, but the data management component must support both its efficient storage, powerful query retrieval, and flexibility to capture all the required types of data and relationships.

In addition, the runtime information represented as time series data must be captured in a mechanism that is both fitting the monitoring component and enabling cross referencing with the graph information model.

5.2.2 Decentralized storage solution

The ACES platform is designed to be able to operate in geographically distributed environments, with multiple EMDCs and cluster working in an orchestrated way to provide a whole platform. For an effective operation of such a system in a scalable and potentially privacy preserving way, the whole ACES knowledge base will be stored using a decentralized solution. Keeping relevant data to the geographical location where they were initially originated will greatly improve the overall scalability and privacy. This solution will be effectively exploited in turn by the ACES reasoning components that can operate, as well from a decentralized model, such as the swarm agent models presented in the orchestration component.

5.2.3 Data consistency, partition, and synchronization

At every location of the ACES system, the knowledge base must have a consistent view. Conflicts within this view of the world would cause further problems among the different reasoning agents. Therefore, the information required at every location will be consistent with the latest view. It is important to note however that the data management system will not have fully replicated information at every management node. Instead, the minimum goal is to ensure all the required knowledge is present and consistent at each location.

Regarding potential information conflicts, it should also be noted that the information will be read at multiple places, but there will not be multiple competing locations from which the same fact is updated. It can be seen that metrics are originated from a single source, and runtime topology characteristics, or changes derived from agent actions (e.g., moving one component).

Nonetheless, information must be propagated in a timely manner from the original source to the nodes requiring it to take their own decision. For these pieces of information, the knowledge base will be consistent data despite being distributed across different clusters. The choice of consistency model (strong, eventual, or causal) will impact the architecture. Moreover, tools or mechanisms for data replication and synchronization within the decentralized storage system must be chosen to maintain this consistency while allowing for the necessary trade-offs between consistency, availability, and partition tolerance (CAP theorem).

5.2.4 Scalability and performance

The knowledge base will have a data volume that grows in particular with respect to the runtime metrics extracted from the multiple clusters. The storage component will be able to scale horizontally to accommodate these needs. In order to achieve so, storage solutions and database technologies used must be capable of scaling out across clusters without significant degradation in performance.

5.2.5 Availability and fault tolerance

The knowledge base needs to be highly available, with built-in redundancy to handle node or cluster failures. This involves implementing replication strategies across clusters, designing for failover, and having robust health-check and self-healing mechanisms in place. The swarm intelligence layer may adaptively manage these aspects, rerouting traffic and workload when necessary to maintain service continuity.

5.2.6 Data security and governance

Data privacy, security, and compliance with regulatory requirements are critical for managing sensitive knowledge bases. This encompasses encryption for data in transit and at rest, RBAC for access control, etc. Governance mechanisms must be established to enforce these policies across the multi-cluster environment.

5.3 Relationship with other ACES elements

The data storage will interact with multiple other components to perform its required functions. In this subsection, we detail what these interactions are, and their purpose.

Runtime data is produced within the monitoring and telemetry components. Therefore, the data initially captured at the observability and monitoring infrastructure will be eventually processed and stored by the data management component. These values will be cross-checked against the overall graph model to verify they belong to a known entity of the knowledge base and will be stored in the time series storage component, where they will be made available for further postprocessing to derive from them usable features for the reasoning components.

The main consumer of ACES data will be the orchestration component. The different agent and reasoning models need to utilize knowledge both for training their internal models and for performing the required diagnosis and decision-making. It is important to establish that every reasoning agent will need a different set of features (derived from specific elements of the knowledge base). This means that there is an interdependency in multi-EMDC ACES scenarios between the location of the different reasoning agents, including the swarm agent components, and the data-providing storage components. This relationship further restricts the strategy for partitioning and replicating the data, as the locations where that information will be consumed must be taken into account for the internal management of each knowledge element.

5.4 Candidate tools

As we have mentioned, the data management component must support two types of data: knowledge graphs for the base concepts plus time series measurements from the observability component. While in principle it would be feasible to opt for a single platform that stores both types of information, they have very different characteristics and requirements, meriting that instead a heterogeneous solution with two systems is selected.

For graph storage we consider platforms that support the property graph paradigm, as it provides a powerful abstraction that can represent knowledge graphs. It offers a robust data model supporting nodes and relationships, each with its associated properties, aligning well with the requirements of complex data relationships in modern applications. The supply and demand ACES knowledge models are defined using the NGSI-LD, with JSON-LD serialization, as is discussed in D3.1. This representation is fully compatible with the expressivity provided by property graphs. Moreover, property graph storage systems can take advantage of the highly expressive OpenCypher query language to retrieve information.

For the graph storage component there are two main alternative frameworks: Memgraph and Neo4J. Both systems provide a community edition that is open source, expanded by a commercially licensed distribution. Another alternative is to use a general purpose NoSQL store such as MongoDB, but that option would lack the native graph advantages of these alternatives such as the availability of a tailored query language. Memgraph is a high-performance, in-memory graph database tailored for handling large-scale graph data with an emphasis on property graphs. Optimized for high-throughput and low-latency operations, Memgraph is particularly suited for real-time analytics. Regarding its scalability, Memgraph supports replication across multiple replicas, although it lacks sharding capabilities where the graph is partitioned among multiple nodes. Memgraph also integrates with streaming platforms like Kafka to support scenarios requiring more timely updates. In comparison, Neo4j is the most established graph management system, predating Memgraph for over a decade. It employs Cypher as its primary query language, similar to Memgraph but with distinct functions and optimizations. While Neo4j is generally efficient, performance benchmarks show a substantial advantage on this front in favour of Memgraph. However, Neo4j's strong community support and a wide range of plugins and integrations are significant assets. Both options present limitations with respect to the requirements of a fully decentralized distributed solution for storing the whole supply and demand model, but they are suitable taking into account. The decision to partition the model will be defined in coherence with the required information and location for the different reasoning agents.

For the time series database component, InfluxDB is an ideal candidate for the storage. This database is specialized on handling time-stamped data, which includes the metrics and events recorded in the

ACES runtime model. Its core strengths lie in high write and query throughput, essential for real-time analytics on time series data. InfluxDB's efficient data compression algorithms and retention policies ensure high scalability for managing the potentially large amount of data generated by the observability and monitoring component. Additionally, its native support for time-centric functions and queries will ease the access of that information for both the data processing functions developed in WP3, as well as the reasoning components that need to utilize these features.

6 Orchestration

The management of the edge infrastructure, the so-called edge continuum, presents a dynamic computing landscape. Within the edge continuum, intelligence is spread across the edges forming a distributed environment. This will make the edge more autonomous and fine-grained in local decision making within a regional context and make it more independent from a central coordination point. Resource allocation, workload scheduling, and data management are challenges that increase the complexity of the edge orchestration and edge-cloud interaction. Despite the growing interest in edge computing, there remains a notable research gap in developing comprehensive solutions that efficiently manage edge interactions [21].

This section introduces a novel component, the so-called swarm intelligence component that combines agent-based modelling and swarm intelligence as an emergent orchestrating mechanism to address these complexities. Agent-based modelling and swarm intelligence are known for providing advantages in simulating complex systems with autonomous entities including adaptability, scalability and robustness. They utilize collective decision-making processes as observed in nature by swarms of insects, fish or birds [22]. This framework is at the core of the architecture, required to manage a powerful edge infrastructure, a mesh of Edge Micro Data Centers (EMDCs) capable of processing big data and AI at the edge-to-edge environment independent from a distant cloud. The EMDCs operate autonomously in serving local demand for edge-cloud services and creating regional collaborative federations to provide edge services. The advantages of creating an autonomous powerful edge are to minimize data traffic between edge to a centralized cloud, reduce costly data extraction for centralized clouds, enhance edge-to-edge data traffic, reduce round trip latency of inference, improve resilience and reduce the fall-out from security breaches [21].

Central to our approach is the integration of autopoietic characteristics that include the emergent intelligence of self-organization, regeneration, and regulation. These characteristics enable the system to dynamically adapt and optimize in response to changing conditions. AI-driven optimization methods (including swarm intelligence) in cloud infrastructure are successfully being researched (see Deliverable D4.2 for more details). Among recent notable examples of utilization of swarm intelligence to optimize complex systems, is the work of Schranz et al. [23], where authors successfully utilize bottom-up job shop scheduling applying swarm intelligence algorithms for optimizing a large production plant. Thus, we propose the edge continuum with its characteristics and limitations as a novel field of application for swarm intelligence [21].

In the edge-continuum, we work on two levels applying self-organization using swarm intelligence:

1. Multi-cluster orchestration comprises the distribution of the demand on multiple clusters in one EMDC.
2. Inter-cluster orchestration comprises the emergent workload scheduling of one cluster in one EMDC

This numeration also reflects the efforts put first in level 1. and then as an extension in level 2.

6.1 Background

The main idea of swarm intelligent behaviour is that it should be able to produce a complex and scalable way of acting in a group, starting from simple and local rules/computations. In nature, the behaviour of social animals appears to be adaptive, robust, and scalable [21]. These are desirable properties of a system that swarm intelligence design precisely aims to replicate in technical systems.

Adaptability represents the ability of a swarm to adapt to dynamic changing environments and to cope with different tasks. By exploiting **robustness**, a swarm can cope with disturbances and failures, such as the loss or the malfunctioning of individual agents. **Scalability** gives a swarm the ability to perform well with different numbers of swarm members and with differently sized problems. Adding or removing swarm members does not lead to a significant decline in performance, as long as their number does not fall below a critical mass (explained below). A collection of individuals can be considered a swarm if it exhibits a verifiable swarm behaviour, including all characteristics mentioned above.

Two crucial aspects should be noted [21]: Critical mass (briefly mentioned above) and super-linear scaling of the system performance. These system features, which can be seen as mandatory for a true swarm system, have been discussed in current literature only marginally.

Critical mass: The advantages of SI (Swarm Intelligence) algorithms can only be exploited if a critical mass of swarm members is reached. In natural swarm systems, such as honeybees' or cockroaches' collective aggregation decisions [23, 24] as well as in autonomous robotic swarms [43], it was found that the number or density of agents plays an important role in the swarm performance and that swarm systems below a critical number of agents do not function well as a collective.

Such a critical mass threshold can be illustrated by the example of the formation process of a sand dune: given three grains of sand, together they do not form a sand dune, although they must obey the same physical laws as the billions of sand grains that form a massive dune. This is because the order-generating feedback loops do not impart any effect strongly enough to overcome the system's noise that drives it towards the disorder. In general, SI applications do not work well below a critical mass, but increasingly well above this threshold up to a size where other effects reduce swarm performance again. It should be noted that it is not yet clear what this critical threshold (the minimum number of swarm members) should be [22].

Super-linear performance scaling: When scaling up beyond the critical mass, it is expected that in any collective system, a large group of agents will achieve more work in total than a smaller group at the same time. In a true swarm, the interactions between the swarm members should exhibit super-linear characteristics, i.e., the effect of the overall system is required to be more than the sum of the effects of its individual parts. Examples are described for honeybees [23], for robotic swarms [25, 26], and for multi-processor systems [27]. The overall system is a well-designed swarm application only if the synergies of cooperation boost each individual swarm member's performance and the local control algorithms of the swarm members are well-designed. This means that within the bounds of feasible swarm sizes, not only the efficiency of the whole swarm as a group but also the efficiency of each single individual must increase. We refer to this as the swarm effect [21].

Figure 9 illustrates the expected performance scaling properties of three different systems with C1 – C4 as fictive threshold numbers for agents in a swarm:

a. The performance of a hypothetical perfectly scaling disembodied algorithm or a swarm model that does not care for physical constraints at all, which will scale linear with O(n) as the dashed line, where n is the number of swarm members.
b. The performance scaling of an algorithm that shares resources with other algorithm instances or a swarm model that considers space as a shared resource for agents, which will scale O(log n) as the dotted line.
c. The performance of a physically embodied swarm system operating in the real world as the solid hat-shaped curve [21].

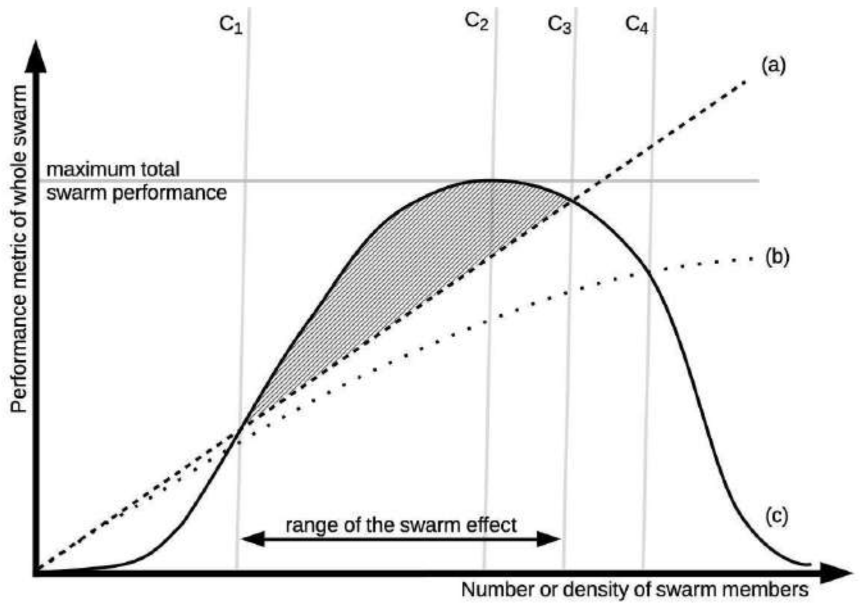

Figure 6.1: Swarm scaling performance

6.2 Multi-Cluster orchestration

In the multi-cluster environment, we consider multiple clusters, thus multiple swarms. On this level the swarm agents represent the demand and the clusters. As in the inter-cluster orchestration, the demand swarm agents represent workload behaviours at the microservice level. The cluster swarm agents, on the other hand, represent the individual clusters of an EMDC that is characterized with the resources available in each cluster. By the implementation of swarm algorithms, these agents collaborate within an Edge Micro Data Center (EMDC) environment, orchestrating processes such as workload placement, storage management, and caching optimization.

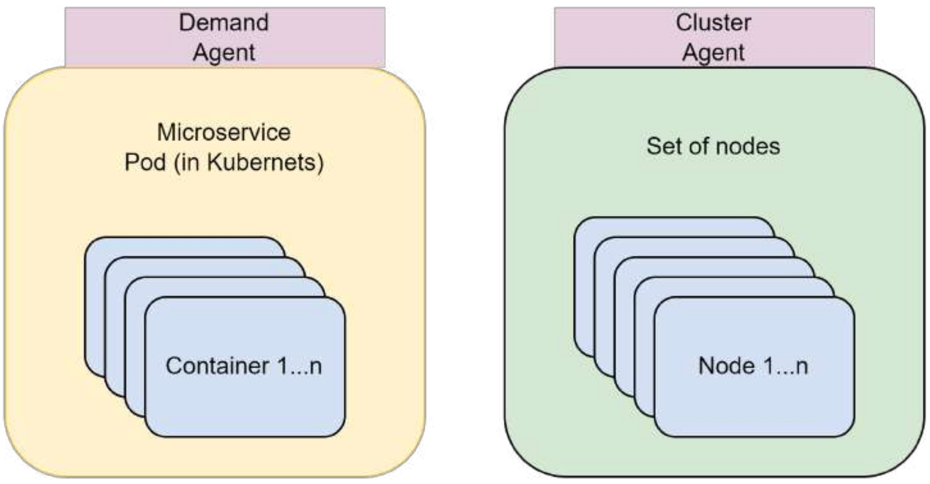

Figure 6.2: Multi-cluster orchestration swarm agents

6.3 Inter-Cluster orchestration

For the inter-cluster orchestration, the key to our approach is the use of swarm agents, representing demand and supply entities. Demand swarm agents represent workload behaviours at the microservice level (or pod level if we directly refer to Kubernetes), ensuring workload scheduling optimization. On the other hand, supply swarm agents represent node dynamics. These agents collaborate within an EMDC cluster, orchestrating processes such as workload placement, storage management, and caching optimization (see Deliverable D3.1 for more details on agent-based modelling). Exemplary swarm algorithms, the hormone and ant algorithms (details in Deliverable D4.2) are utilized to accomplish the desired functionality of the system. For example, demand swarm agents deploy synthetic hormones to communicate their requirements and priorities. Supply swarm agents detect these hormones to make informed allocation decisions. The ant algorithm dynamically optimizes workload-node assignments by simulating the foraging behaviour of ants, depositing pheromones to guide subsequent decisions [28]. The agent types are shown in Figure 6.3.

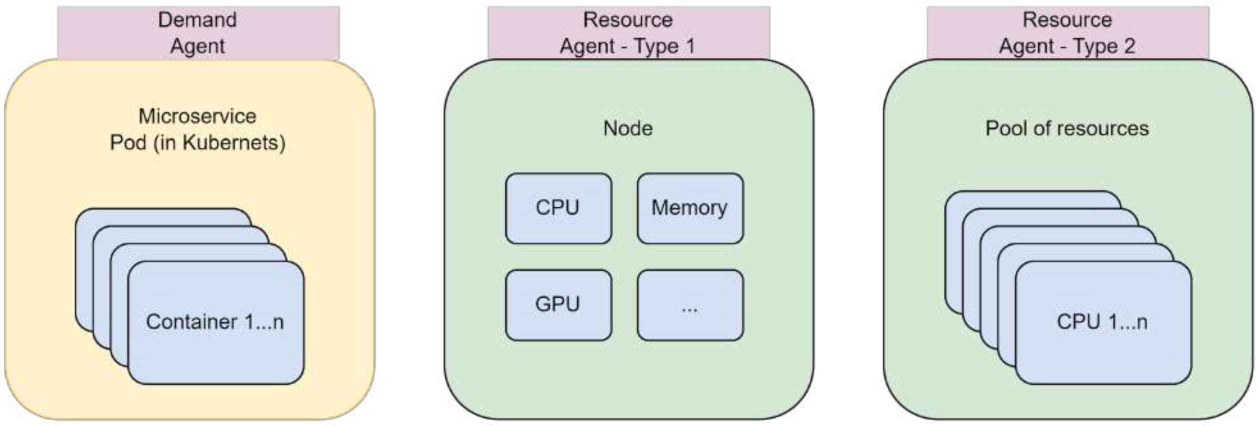

Figure 6.3: Inter-cluster orchestration swarm agents

6.4 Innovations through swarm intelligence

An innovation that the ACES project will introduce is the use of AI/ML to achieve autopoietic behaviour in individual agents, on multiple layers. Specifically, regarding swarm agents, ACES will make the coalition's behaviour adapt autonomously to changes in the operating environment. To do this, AI/ML algorithms will be used to enable the swarming algorithms to calibrate and update their hyperparameters autonomously.

Hyperparameters in swarming algorithms control the overall behaviour of the coalition (i.e., intensity of hormone attraction, quantity of generated hormones, mobility of hormones, etc.) and are usually chosen through random/grid searches, heuristics, or trial and error. In ACES, on the other hand, we will implement AI/ML tools (such as Bayesian learning or Reinforcement Learning) that, by monitoring the performance of KPIs, will be able to select the best value for the hyperparameters.

Furthermore, once an initial configuration of hyperparameters is found, the proposed algorithms will be capable of quickly adapting to real-time changes in the context, using the previous configuration as a starting point for recalibration.

More details on this are given in D3.1 and D4.2. Future deliverables D3.3. and D4.3 will explain the procedure in depth.

7 Networking

This Chapter aims to present a general overview of the ACES network architecture. The ACES network will be *intelligent* through an advanced closed-loop **Software-defined Networking (SDN)**-based network control infrastructure that couples monitoring with machine learning-based analytics. It will provide a range of new functionalities targeting edge-based infrastructures, including in-network authentication and intrusion detection for security, network acceleration for performance, and advanced forms of in-network computation to assist the swarm-based ACES intelligence.

7.1 Design requirements

From the networking point of view, the design of ACES is driven by eight main requirements.

Service connectivity: A critical aspect of the ACES edge-computing platform is the establishment of robust service connectivity. This goal involves configuring seamless communication between service instances while prioritizing Quality of Service (QoS) requirements, namely, ensuring low latency and high throughput to meet the platform's objective of enhancing intelligence at the edge.

Network observability: ACES requires high operational visibility to gain deep insights into the performance and behaviour of the underlying infrastructure. This visibility not only aids in proactive issue detection but can also enable efficient resource allocation, contributing to the overall stability and reliability of the edge-computing system. ACES network approach to this problem entails incorporating in-network telemetry mechanisms.

High throughputs and low latencies: Meeting the demands of edge computing requires high network throughputs and low latencies. This optimization ensures that data traverses the network swiftly and efficiently, enabling the platform to deliver timely and responsive intelligence to edge systems. ACES achieves this by incorporating in-network acceleration mechanisms.

Closed-loop network control: ACES network control should enhance the platform's agility and responsiveness to varying workloads and network conditions. Towards this goal, ACES will integrate network monitoring with machine learning-based analytics to form a closed-loop control. This synergy empowers the platform to dynamically adapt to changing conditions, automatically adjusting configurations and resource allocations based on real-time insights.

Scalability: The scalability of the ACES network platform requires a distributed control architecture. This design allows the platform to seamlessly expand its capabilities to accommodate a growing number of edge devices and service instances. By distributing control functions across the network, ACES ensures scalability without compromising performance, supporting the evolving needs of edge computing deployments.

No single point of failure: Redundancy is a fundamental consideration for ACES to guarantee uninterrupted service. The platform adopts a design philosophy with replicated control, eliminating any single point of failure. This redundancy ensures continuous operation even in the face of hardware or network component failures, enhancing the reliability and resilience of the edge-computing infrastructure.

Network security: Security is paramount in the design of the ACES platform. Our aim is to incorporate comprehensive security features into the networking sub-system to ensure that ACES is secure by design. We will implement robust network security measures employing AI mechanisms to safeguard against potential cyber threats. We leave details on this specific requirement to Chapter 10.

Multi-Cluster Networking: The multi-cluster networking solution must seamlessly integrate Kubernetes clusters and programmable network switches while supporting dynamic scaling and pod mobility. This will be brought through technologies such as Submarriner or Istio service mesh.

7.2 Network architecture overview

The network architecture of the ACES edge-based platform reflects a design that includes SDN principles, control-loop enhancements, and programmable data plane devices. This holistic approach aims to provide a foundation for adaptive, intelligent, and efficient edge networks.

The ACES network architecture embraces the **SDN paradigm**, a departure from traditional networks that rely on dedicated hardware devices like routers and switches for network traffic control. In ACES, the forwarding state in the data plane is managed by a remote control plane, introducing a decoupling between the two planes that enhances adaptability and agility. The SDN approach within ACES further incorporates a **control-loop mechanism**, seamlessly **integrating network telemetry and machine learning**-based analytics. This integration allows ACES to respond dynamically to changing network conditions and harness insights derived from analytics for informed decision-making, creating a symbiotic relationship between network control and intelligent data analysis.

Our SDN-based design enables the **dynamic orchestration of ACES multi-cluster networking**. We resort to overlays to allow for dynamic scaling and mobility of pods across clusters. To enable seamless interfacing of the Kubernetes multi-cluster with our network switch infrastructure, we plan to explore existing and develop new Container Network Interface (CNI) plugins. Furthermore, our design will include isolation and segmentation mechanisms between clusters for security and operational flexibility. Finally, we integrate network switch monitoring into Kubernetes monitoring tools, ensuring comprehensive visibility and control over both the Kubernetes and network switch environments.

ACES takes a multifaceted approach to intelligence within its network by **leveraging programmable data plane devices**, including programmable switches and SmartNICs/DPUs. This strategic integration goes beyond conventional network architectures, enabling ACES to achieve enhanced intelligence through in-network telemetry, accelerated network functions, integrated in-network security, and support for swarm-based orchestration mechanisms. Programmable switches and SmartNICs/DPUs allow the platform to adapt and evolve, allowing for dynamic adjustments in response to varying workloads and operational requirements. This level of programmability not only enhances the overall intelligence of the ACES network but also ensures a scalable and efficient edge infrastructure capable of meeting the diverse demands of modern edge computing applications.

7.3 Control plane

In contrast to traditional networking models where control functions are distributed across individual devices, ACES opts for a logically centralized control model following an **SDN** approach. This choice allows for unified and programmable control over the network, facilitating efficient resource allocation, dynamic configuration changes, and seamless adaptability to the evolving demands of edge computing environments. The SDN-based logical centralization enables the intelligence ingrained in the ACES network control architecture.

ACES prioritizes fault tolerance and scalability in its network control plane architecture. For **fault tolerance**, ACES replicates the control plane. In a replicated control plane, multiple instances of the SDN controller are deployed across different nodes, working in tandem to manage network operations. If one instance fails (due to hardware issues or other unforeseen circumstances), the remaining replicas continue to operate seamlessly. This redundancy minimizes the risk of a single point of failure, ensuring the reliability and continuous operation of the network control even in the face of unexpected events.

For **scalability**, ACES distributes the control plane. As the ACES edge infrastructure grows with additional EMDCs, the controller can efficiently manage the increased load by distributing control functions across multiple instances. This balancing of efforts ensures that the network control remains responsive and can effectively handle the evolving demands of a dynamically expanding edge environment.

The ACES network control plane features a sophisticated **closed-loop mechanism, integrating SDN principles with Machine Learning (ML) analytics**. This fusion creates a dynamic and intelligent control system capable of autonomously adapting to changing network conditions. The closed-loop mechanism continuously monitors the network, gathering performance and traffic pattern data. This data is fed into an ML analytics platform, enhancing the system's ability to predict, analyse, and optimize network behaviour.

In summary, the network control plane of the ACES edge-based platform is characterized by its SDN-based logical centralization, replicated control for fault tolerance, distributed control for scalability, and a closed-loop mechanism augmented with ML-based analytics. This approach establishes the foundation for a resilient, scalable, and intelligent network control system tailored to the dynamic nature of edge computing environments.

7.3 Data plane

The network data plane of the ACES edge-based platform integrates **high-performance programmable data plane devices**, such as programmable switches, Data Processing Units (DPUs), and/or Intelligence Processing Units (IPUs). This integration aims to enhance multiple facets of the edge environment. For instance, leveraging programmable switches' capabilities fortifies security by expediting intrusion detection processes. Network performance may benefit from the direct execution of AI/ML algorithms and cryptographic operations in the data plane, reducing latency and improving overall efficiency. Furthermore, offloading specific application and system functionality onto programmable devices can elevate ACES infrastructure control (e.g., by running swarm mechanisms on network devices) and boost service performance.

ACES includes **fine-grained network telemetry** as a cornerstone of its data plane architecture. Examples include the implementation of in-network sketches and per-packet statistics computations, providing comprehensive insights into network behaviour and performance. In addition to traditional telemetry methods, ACES leverages in-band network telemetry data to refine the monitoring capabilities further, by gathering telemetry metadata for packets traversing the network (e.g., explicit information on packet routing paths, latency experienced by packets, and in-network congestion information). This level of granularity enables real-time monitoring and analysis of network traffic, facilitating the identification of potential bottlenecks, congestion points, and security threats. These advanced telemetry mechanisms help support data-driven decision-making and proactive management of network resources.

The ACES data plane enables the **acceleration of critical network functions**, including load balancers, Network Address Translators (NATs), proxies, firewalls, and more. ACES explores innovative program synthesis techniques to address the inherent software engineering challenges of supporting its diverse data plane platforms. Different techniques will be explored, based on formal methods like symbolic execution or AI/ML approaches, to enable the automatic generation of optimized code for specific target platforms. This approach enhances the efficiency and flexibility of the ACES data plane, ensuring that it can seamlessly adapt to the diverse requirements of edge computing workloads.

In summary, the network data plane of the ACES edge-based platform leverages recent programmable network hardware to deliver high performance, security, and service efficiency. Integrating fine-grained network telemetry and function acceleration directly in data plane devices collectively forms the foundation for a robust, performant, intelligent network architecture.

8 Monitoring and observability

When considering complex distributed architectures—spanning multiple cluster or edge environments—the ability to gain deep insights into the performance, health, and interactions of these clusters, nodes, and constituent workloads becomes paramount. Observability represents an approach to analysing and optimizing systems by providing a real-time perspective on all operational data related to applications and infrastructure. Observability lays the groundwork for the ACES platform to proactively identify and address issues to ensure seamless operations and optimal resource utilization. However, the complexities of multi-cluster or edge environments, such as the ones in ACES, change the way of comprehensively viewing system behaviour, dependencies, and potential bottlenecks and subsequently detecting, diagnosing, and resolving fatal errors.

The monitoring and observability framework is a vertical ACES component, spanning the overall ACES architecture and its constituent components. The component provides monitoring and observability aspects to the different layers of the software stack on various levels (i.e., edge, application, network, and cloud layer). It encompasses monitoring, logging, tracing, metrics collection, alerting, anomaly detection and analysis, visualization, and performance analysis. Due to its inherent distributed nature, the monitoring and observability framework considers hierarchical and distributed monitoring and storage, including across multiple clusters.

In this section, the monitoring and observability state-of-the-art is described, which serves as a foundation for the proposed monitoring and observability requirements and architecture. Next, the requirements with the resulting architecture are detailed and related to other ACES components in light of federated/distributed multi-cluster monitoring. Moreover, the monitoring and telemetry data, as well as their acquisition aspects, are discussed. Finally, considerations on asset runtime, fine-grained network, periodic and event-driven observability and monitoring are presented.

8.1 Monitoring and observability in modern systems

The notion of observability states that a system is observable when the root cause of issues or events can be straightforwardly detected without additional investigation. That is, the system's observability is delineated by the capacity to understand its complex internal state based on sufficient measurable external outputs. On the other hand, monitoring aims to assess the system's state via the collection, measuring, and analysis of the system's outputs. As monitoring generally considers only a predefined set of measurements, its scope is limited to detecting only a specific set of eventualities**.** This implies that with the complexity of modern systems, observability permits the so-called white-box monitoring [31], i.e., monitoring where the system internals are known and comprehensively report their current state is required, as opposed to traditional black-box monitoring, which primarily observes the system's state from outside.

With the upswing of cloud and edge computing, modular distributed systems composed of flexible containerized services are becoming ubiquitous. Such workloads may be extended with a behaviour logic description that characterizes its internal behaviour and improves observability [32]. Moreover, as distributed workloads and systems generally have a complex execution path—with requests crossing multiple services or compute nodes—, a need for capturing resulting traces emerges. In [33], a generic methodology for capturing such requests is proposed. Another crucial aspect of observability is identifying the performance of distributed systems. To this end, a mixture of benchmarking and simulation approaches have been applied [34]. Finally, to provide actionable information to the operator, detecting and explaining system deviations is required. Several approaches to automated detection of anomalies have been proposed, such as in [35], as well as root cause analysis, as in [36].

Whereas the area of monitoring and observability has been thoroughly explored and with many industrial production systems already in place, a number of challenges remain. In [37], a qualitative analysis observed the following contemporary challenges in distributed system monitoring:

- Increasing dynamics and complexity of distributed systems, especially due to the emerging trends of microservice architectures and cloud computing, which are not manually manageable.
- Heterogeneity of distributed systems, which often constitute of legacy components and consider multitenancy.
- Problems in culture and mindset, which may result in monitoring pitfalls, such as isolated monitoring and lack of collaboration/communication.
- Distribution of monitoring aspects without employing a centralized system-wide point of view, lowering the transparency of impacts.
- Extensive amounts of data, which may impact the prioritization and ability to draw conclusions due to the complexities arising from overwhelming information.
- Expert dependency and lack of experience, time and resources, which may hinder the timely and comprehensive diagnosis of issues and imply reactive as opposed to proactive monitoring and observability.
- Insufficiently defined non-functional requirements, such as availability and performance, resulting in inadequately measured and monitored context.
- Applying reactive instead of proactive implementation, often triggered by a failure in production systems, resulting in ad hoc solutions as opposed to systematic solutions.

The major features of a monitoring and observability system should thus cover the aspects of data requirements and related measurements, basic functionalities and key characteristics [8] as outlined in the following. The three main data types (logs, metrics, traces) provide the operators with the so-called golden signals of observability, which serve as a basis for alerting, troubleshooting and tuning/capacity planning. Generally, the following telemetry measurement types are considered: latency, traffic, errors, and saturation. Building upon the measurements, the observability system should provide three key functionalities: correlation, i.e., linking events to other related events, topology, i.e., providing a graph of dependencies, and incident response, i.e., automated handling of remediation. The key characteristics of observability systems are connected context, easier and faster exploration, a single source of truth, capturing arbitrary wide events, and decoupling data sources from sinks [38]. Moreover, [37] denotes the following requirements in employing observability in distributed systems: a holistic approach, management from a business and user experience view, the definition of core metrics from a customer-centric view, governance, use of a unified monitoring platform, detection of normal and abnormal patterns, among others.

Observability and monitoring in practice have been enabled via various solution implementations, covering aspects from monitoring data collection to visualization. In Table 8.1, we provide an overview and comparison of state-of-the-art open-source solutions. The solutions generally target specific monitoring scenarios, such as networks or applications. Most solutions provide some form of visualization capacities and additionally enable integration with standard telemetry/monitoring ecosystems. Moreover, big data and scalability support is generally provided out-of-the-box. Finally, some solutions provide advanced analysis, anomaly detection and alerting functionalities.

Table 8.1: Overview of monitoring and observability solutions

SOLUTION	SCOPE	MAIN FEATURES
Apache SkyWalking[1]	Application performance monitoring	– End-to-end distributed tracing, service topology analysis – Log management pipeline, metrics aggregation – Alerting and telemetry pipelines – Integration with telemetry ecosystems – Big data scalability

[1] https://skywalking.apache.org/

Consul[2]	Application network observability	– Service discovery mechanism – Metrics collection and topology visualization – Integration with telemetry ecosystems
Kibana[3]	Observability data analytics	– Data ingestion and enriching – Search and analysis of data and visualization – Full and multi-stack monitoring – Automated alerting – High scalability and resiliency
Cilium[4]	Network observability	– Service map visualization – Network flow logs – Metrics and tracing collection/export – Advanced network protocol visibility
Prometheus[5]	Monitoring system	– Big data scalability – Standardized query language – Alerting and visualization Integration with telemetry ecosystems
OpenTelemetry[6]	Telemetry collection	– Collection and export of traces, metrics, logs – Support for system instrumentation – Standardised
Monasca[7]	Monitoring system	– High scalability and performance – Multitenancy support – Integration with OpenStack – Metrics processing and querying Streaming alarm and notification engine
Sysdig[8]	System/container observability	– Native inspection of physical/virtual machines and containers at OS-level, considering storage, processing, network, and memory subsystems – Trace collection, filtering – Unified and customizable user interface
Grafana[9]	Observability platform	– Visualization and querying of data – Anomaly detection and alerting – Integration with telemetry ecosystems – High scalability and performance
Jaeger[10]	Distributed tracing platform	– Distributed workflow monitoring – Analysis of service dependencies and visualization – Integration with telemetry ecosystems – High scalability and performance

[2] https://www.consul.io/
[3] https://www.elastic.co/kibana
[4] https://cilium.io/
[5] https://prometheus.io/
[6] https://opentelemetry.io/
[7] https://wiki.openstack.org/wiki/Monasca
[8] https://sysdig.com/
[9] https://grafana.com/
[10] https://www.jaegertracing.io/

Fragmenting of solutions and lack of interoperability present a burden in modern observability and monitoring. In addition to vendor lock-in when applying proprietary solutions, a considerable risk is represented by the compatibility of different monitoring and observability solutions, which hinders the ability of composing comprehensive and connected systems. To this end, various initiatives have emerged aiming to define interoperable and standardized specifications and implementations. Notably, to ensure portability and interoperability in telemetry, a set of telemetry standards and specifications have been defined in the OpenTelemetry specification [39] that provides a set of rules, guidelines, and requirements that the resulting implementations should follow. The specifications define API, SDK, as well as data model specifications. Moreover, an Observability Query Language Standard (QLS) [40] workgroup in the Cloud Native Computing Foundation (CNCF) has been recently established to define a standardized query language for observability data. Additionally, standardization activities took place to establish a standardized Prometheus remote-write protocol [41] for transmitting metrics data. To tackle establishing a common data model for propagating context information enabling distributed tracking scenarios, a W3C recommendation on trace context was defined [42].

8.2 Requirements

The design requirements for the monitoring and observability component outline the desired specifications and functionalities of the related components. This information is crucial for the definition of the architecture and provides guidance throughout the subsequent system development stages. The requirements were gathered by reviewing state-of-the-art technological and non-technological aspects outlined in Section 8.1, as well as considering the ACES requirements, architecture, and use cases defined in this document. Table 1 presents a compilation of these requirements, including both functional and non-functional aspects. Furthermore, these requirements are linked to the architectural elements described in the next section.

Table 8.2: Monitoring and observability module requirements

NAME	GROUP	DESCRIPTION	RELATED SUBMODULE
Metrics, traces, and log collection	Functional	The solution should enable the collection of metrics, traces, and logs of ACES components, infrastructure, and workloads.	All
Data collection mechanisms		The solution supports several data collection mechanisms, such as pull/push or custom protocols.	Retrieval Worker, Push Gateway, Data Forwarder
Anomaly detection and alerting		The solution should implement real-time log/metric analysis and anomaly detection on monitored data and related alerting of implicated entities.	Anomaly Detection, Alert Manager
External anomaly detection		The solution should support integration with external anomaly detection and alerting solutions.	Anomaly Detection, Alert Manager
Automated target discovery		The solution should support automated configuration and the addition of monitored targets.	Service Discovery Mechanism

Data retrieval and aggregation		The solution should implement interfaces for monitoring data retrieval and aggregation.	Server, Data Forwarder
Data analysis and visualization		The solution should support data analysis and visualization.	Data Analysis, Export and Visualization
External distributed storage		The solution should enable the integration of external distributed storage systems.	Internal Database, Storage
Monitoring federation		The solution should enable federation among distributed monitoring instances.	Data Forwarder, Storage
Standardized query languages		The solution should support data retrieval, aggregation, and analysis via standardized query languages.	Server
Large data	Non-functional	The solution should support large data processing.	All
Open standards		The data models, formats, and APIs should be based on established open standards.	All
Real-time processing		The solution should enable low-latency real-time processing and analysis.	All
Modularity and extensibility		The solution should be modular and extensible.	All
Interoperability		The solution should be interoperable with modern as well as legacy monitored targets and systems.	Push Gateway, Data Forwarder
Scalability		The solution should be horizontally and vertically scalable.	All
Constrained environments		The solution should support constrained edge environments.	All

8.3 Monitoring and observability architecture

To effectively analyse and develop complex systems, a structured system architecture is necessary. This architecture outlines the modules and their interactions, helping with experimentation, validation, and reasoning about the system. In Fig. 8.1, we present the ACES monitoring and observability architecture, which is built upon the analysis of state-of-the-art platforms and standards as well as system requirements provided in Sections 8.1 and 8.2, respectively. The proposed generic architecture may be applied to implement custom and specialized metric pipeline architectures, such as the one defined in D3.1 - ACES Data and Knowledge Model.

The ACES Monitoring and observability architecture targets decentralized system observability, observability data management, and alerting. The architecture consists of the main subcomponents Monitoring and Observability Core, Push Gateway, Alert Manager, and Data Forwarder, which provide core functionalities of monitoring and telemetry data collection, storage, forwarding, querying, and alerting to ACES workloads or other ACES components. Additionally, auxiliary subcomponents enable functionalities such as service discovery or data analysis, data export, and visualization. In the following, we provide additional details on the subcomponents.

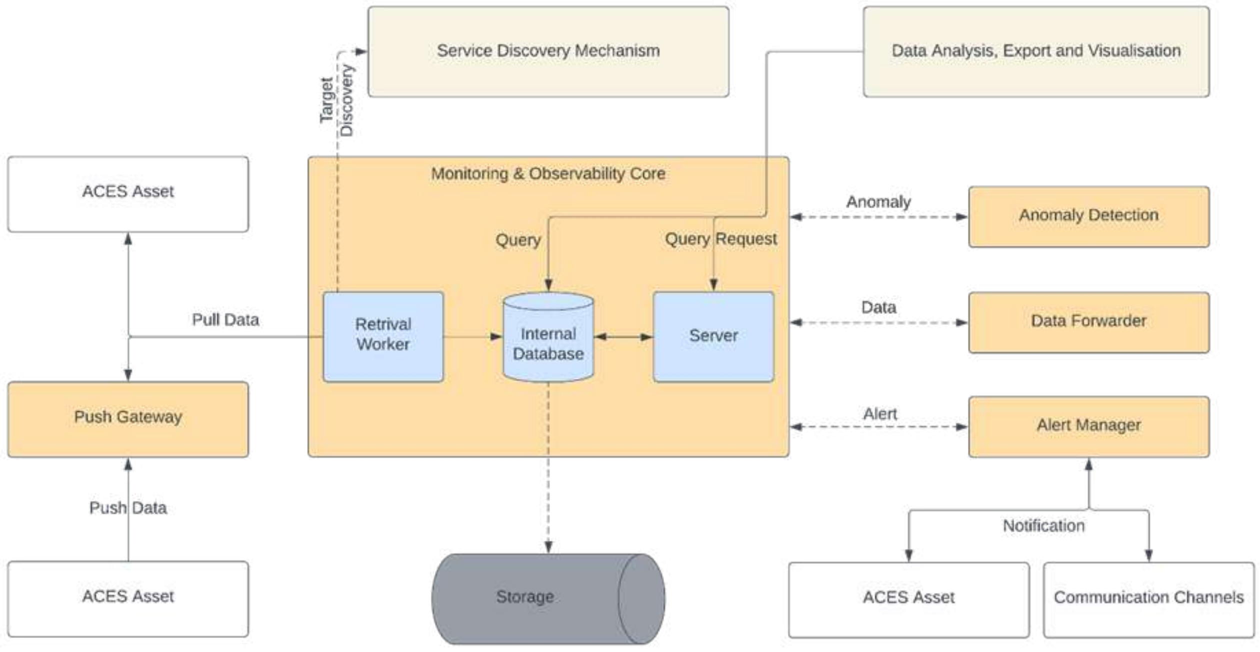

Figure 8.1: Monitoring and observability architecture

Monitoring and Observability Core represents the primary subcomponent of the architecture, as it is the part of the system that conducts the actual monitoring. It consists of the following components:

- *Storage*: A local time series database optimized to store vast volumes of timestamped data. This database is used to store all collected monitoring and telemetry data.
- *Retrieval Worker*: A component that periodically pulls metric and telemetry data from workloads, Push Gateway or other components and saves them to storage.
- *Server*: An API server that accepts queries and retrieves the requested data from the Internal Database. This component is the primary interface for data analysis, export, or visualization workflows.

Push Gateway allows the Monitoring & Observability Core to receive monitoring and telemetry data via an alternative push mechanism. The workloads may push their data to the gateway, which is then periodically queried via the Retrieval Worker. This mechanism is especially relevant for short-lived workloads.

Data Forwarder enables pulling or pushing monitoring/telemetry data from and to other ACES components or tools via custom pluggable protocol and format extensions. For instance, the Data Forwarder subcomponent may be applied to implement additional metric scraper or streaming modules. Moreover, the Data Forwarder subcomponent may be applied to establish federation among the distributed monitoring instances. Additional data aggregation steps may be applied while forwarding data.

Anomaly Detection submodule implements real-time log and metric analysis to enable early detection of system issues and transgressions. In case of an anomaly, the concerned entities are notified. The submodule may be extended with external cognitive components.

The **Alert Manager** component handles the alerts sent by the Anomaly Detection subcomponent or other applications. It enables automatic alert routing and forwarding, as well as more complex operations, such as alert deduplication, grouping-related alerts, or rule-based alert suppression.

Service Discovery Mechanism is a mechanism that allows the Monitoring & Observability Core to discover and monitor targets automatically, i.e., with no manual configuration. This proves especially useful when working in environments where the number of running instances may be rapidly changing. The Service Discovery Mechanism may support one or more discovery methods, such as static discovery files, DNS-based discovery or other orchestration-specific discovery.

Data Analysis, Export and Visualization implements advanced data analysis and retrieval methods via graphical user interface dashboarding toolboxes or data browsing and analysis solutions. The data are retrieved from either the Internal Database or Server component using a standardized query language.

ACES Asset represents any monitorable ACES workload, ACES cluster, ACES component or ACES infrastructure element in the ACES platform.

Communication Channels define external messaging means capable of displaying notifications, e.g., dashboards or communication platforms.

8.3.1 Federation and multi-cluster monitoring

There are several approaches to implement monitoring and observability when considering multiple clusters within ACES. One option is to apply a single cluster with a monitoring instance to scrape the data from other clusters or to scrape the data from remote metrics endpoints. The advantage of this approach is simplicity, as there is only one monitoring instance in a central cluster that collects data from multiple clusters. Another advantage is the fact that monitoring, alerting, and analysis are centralized, thus enabling a unified view of the system. On the other hand, there are several disadvantages, including a single point of failure (due to centralization). Additionally, latency in data ingression may impact the speed at which metrics are collected, potentially leading to delayed or inaccurate monitoring. A somewhat similar approach is based on a push mechanism; the remote clusters may remotely write the data directly to the centralized monitoring instance.

Alternatively, a range of monitoring federation principles may be applied. Fundamentally, we distinguish two types of federation approaches targeting different goals: query and scrape federation. Such systems are more complex as they require a monitoring instance in each monitored cluster. The scraping federation considers a monitoring instance in the central observability cluster that periodically scrapes the monitoring instances in remote clusters. Whereas this again enables a unified view of the system and additionally improves availability due to distributed redundancy, there is considerable data duplication. Moreover, additional latency ensues due to the periodic scraping delay on every level of the hierarchy. To enable federated querying, an implementation that extends the monitoring instance with a query sidecar may be adopted. Using this approach, the data stays in a local cluster while a comprehensive view of all clusters, as in the centralized system, is enabled. Nevertheless, if a remote cluster is unreachable, consequently, the data are unreachable. Queries can additionally experience extensive delays if large amounts of data have to be transferred ad hoc, especially in edge environments.

8.3.2 Relationship with other ACES components

The monitoring and observability component spans the overall ACES platform and components and, in this, relates to all ACES workloads, ACES components, or ACES infrastructure elements in the ACES platform to provide monitoring and observability aspects. Additionally, a specific integration point with the Distributed Storage component is identified for data storage functionalities and with the Security and Privacy component for authentication, authorization, auditing, and secure storage functionalities. Primary consumers of the Monitoring and Observability component are analysis and AI components that require knowledge discovery or implement proactive management based on the operational monitoring insights, such as Resource Management, Cognitive Engine, Networking or Orchestration components. Integration capabilities are provided via the Data Forwarder subcomponent.

8.4 Monitoring and telemetry data

The ACES project will provide monitoring and observability related to the different levels of the software and hardware stack, targeting various layers, such as the edge layer, cloud layer, application layer, and network layer. In addition to workload, component, and node observability, an important aspect is network observability. High operational visibility is required to gain deep insights into the performance and behaviour of the underlying infrastructure, which aids proactive issue detection, optimization, and related efficient resource allocation. To comprehensively characterize the infrastructure and workload execution context, the following types of monitoring and telemetry data (also referred to as signals) will be collected:

- **Traces** are an essential concept in understanding how requests and traffic move through the infrastructure, components, workloads, and services. They help understand the entire path of the process and can be used to identify bottlenecks and errors.
- **Logs** are simple timestamped text records that may or may not be structured. Historically, logs are the most widespread form of system observability. They are almost always used in legacy software and systems due to widespread built-in support.
- **Metrics** are runtime measurements of the state of the system. They are used to monitor the health of the system and are typically used to trigger alerts.

As the monitoring and observability component stores all data as time series streams of timestamped values belonging to the same data type and set of labeled dimensions, every time series is uniquely identified by its data name and optional key-value pairs where the key name indicates the parameter that is being monitored. Additionally, the data may have supplementary attributes which provide further metadata, such as data description. The implementation of the monitoring framework will consider four main metric types:

- *Counter*: A counter is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. The counter is generally used for metrics that measure the number of requests, errors or completed tasks.
- *Gauge:* A gauge represents a single numerical value that can either increase or decrease. Gauge is used for metrics that measure CPU temperature, memory usage or the number of running processes.
- *Histogram:* A histogram samples observations and counts them in configurable buckets. It also provides a sum of all observed values. Histograms are used for metrics that measure request duration or response size.
- *Summary:* Summary is similar to histograms with the extension that the summary can calculate configurable quantiles over a sliding time window.

The definitive set of anticipated metric data types is provided in D3.1 - ACES Data and Knowledge Model and subsequent deliverables.

8.5 Data acquisition methods

Data acquisition in monitoring and observability refers to the processes of monitoring/telemetry data ingression and retrieval. In the following, the most common methods and related considerations are presented.

Pulling is the primary data collection mechanism, supported by the Retrieval Worker. Its flow is as follows: the monitored assets must expose their metrics on a predetermined endpoint, generally named "*/metrics*"; the Retrieval Worker then scraps these endpoints via a request, and the content is stored in the Internal Database. One advantage of this approach is the simple addition of additional monitored assets due to the light configuration change requirements or even no extra configuration in case a discovery mechanism is applied. Additionally, identifying when a monitored asset is unreachable is reliable and asset metric data are accessible to other authorized entities that can access the endpoint, while multiple pull-based monitoring tools can use the same endpoint simultaneously. With this approach, generally, only one centralized configuration of the Retrieval Worker is required. On the other hand, there is a need for a discovery mechanism to identify the assets. This is often accomplished by means of the host name. Finally, the endpoints must be accessible to the Retrieval Worker. This can be simplified by keeping an instance of the Retrieval Worker in the same cluster as the assets.

In order for the Retrieval Worker to be able to parse and store the scraped data, the endpoints must use a standardized format referred to as exposition format. Generally, text-based formats are applied. The format's structure is usually based on lines, meaning each line will be considered a new entry, which are sometimes referred to as samples. Lines are separated by a line feed character. Tokens within the line must be separated by at least one space. Empty lines as well as trailing, leading and redundant whitespaces are ignored. The following values are then extracted:

1. Metric names and labels: used for identification, querying, additional information, and human understanding. Labels are optional, and the combination of a metric name and labels must be unique for every line.
2. Value: a float value representing the numeric value of the metric. NaN, Inf+ and Inf- may be provided for not a number, positive and negative infinity, respectively.
3. Timestamp: an integer representing the time of the event in a standard timestamp format.

The second method of data collection is via a **push** mechanism. As the Retrieval Worker can only pull data, which may not be applicable for short-lived workloads and assets, a Push Gateway component is proposed. The assets push their data to the Push Gateway, which serves as temporary cache storage. The data are then periodically pulled from the Push Gateway by the Retrieval Worker.

Data retrieval for analysis and visualization is supported by directly querying the Internal Database or requesting the data from the internal Server. Querying is generally performed using a standardized query language, such as PromQL[11], which enables filtering and aggregation by the metric name, labels, timestamp of the data, or time ranges.

Data collection and retrieval using custom formats and protocols, such as AMQP, is provided in the form of a Data Forwarder component. Data Forwarder is an extensible component that enables retrieving or ingressing data via arbitrary protocols in a plugin-like fashion. Additional data aggregation steps may be applied while forwarding data.

8.6 Monitoring and observability of asset runtime

Observability in modern service-based systems is characterized by its complex execution and implementation. During the asset runtime—the asset being an operating system, compute node, network switch, workload, or other monitorable entity—the system can be in numerous states and experience many transitions. To properly characterize the system runtime, the system should provide sufficient signals that aptly describe the execution context. In general, the following signals (as defined in Section 8.4) are captured: traces, metrics, and logs. Traces are typically composed of spans, which are individual units of work. Spans are connected in a tree-like structure, where the root span is the entry point of the request, and the leaf spans represent the exit points. The spans are usually represented in a JSON format and are the easiest to describe as well-structured logs. Spans within the same trace all share the same trace identifier and are hierarchically connected via a parent identifier, which contains the span identifier (a unique identifier of a span) of the span's parent. In distributed tracking, context propagation is a core concept that allows for spans to be assembled into a single trace regardless of where they were generated. The context is an object storing information that allows us to correlate related spans and associate them with a trace. Propagation is the process of passing context between services.

Each observable asset must produce and emit some of the above-mentioned signals. This process is called instrumentation. There are three main kinds of instrumentation corresponding to different integration considerations:

[11] https://prometheus.io/docs/prometheus/latest/querying/basics/

- *Native instrumentation:* A system component is instrumented out-of-the-box, i.e., the component was instrumented at development time.
- *Automatic instrumentation:* A system component may be instrumented by applying additional external dependencies that automatically plug in into the component and instrument it.
- *Manual instrumentation:* A system component must be manually adopted by changing the component and reporting the relevant telemetry.

To ensure portability and interoperability, a set of telemetry standards and specifications are normally adopted. The specification provides a set of rules, guidelines and requirements that the resulting implementations should follow. The specifications define API, SDK, as well as data model specifications. An example of such a specification is the OpenTelemetry[12] specification.

8.7 Fine-grained monitoring in network switches

One of the innovations of the ACES monitoring framework is its tight integration with the network sub-components. Specifically, we will leverage the computational capabilities of network data plane devices (including programmable switches and SmartNICS/DPUs) to improve network observability.

The ACES network switches will compute fine-grained, flow-based metrics, *per packet*, directly in the data plane. The rationale for our design is as follows. When deployed in an EMDC edge at Terabit traffic speeds, conventional server-based solutions can only monitor *a small subset of traffic* for its downstream applications, as they are limited to a few Gbps packet processing at best. Network traffic needs thus to be sampled (at very low sampling rates) to meet the capabilities of the monitoring server. By observing and computing in-network statistics *over all network traffic* in the network data plane, the ACES monitor records are richer than the sampling-based records generated by traditional systems, enabling new and improved network monitoring applications.

The ACES network telemetry data is to be considered along three axis, which we describe next (and is further detailed in Deliverable D3.1).

Flow type. The ACES network switch monitor will compute metrics for multiple flow keys. Currently, we are considering 4 types of keys: *[MAC src, IP src], [IP src], [IP src, IP dst], and [5-tuple]*.

Flow atoms. The ACES switch stores telemetry data as "flow atoms". These are specialized counters pertaining to a specific flow key. At the moment, we consider three flow atoms: number of packets, number of bytes, and squared number of bytes. These atoms are maintained in the switches' stateful memory and are used to compute several statistics.

Flow statistics. For generality, the ACES switch will compute a diverse set of statistics of two types: unidirectional (1D), tracking outbound traffic, and bidirectional (2D), considering both inbound and outbound traffic. The 1D flow statistics considered include weight, mean, standard deviation, time intervals, etc. The 2D statistics include magnitude, radius, approximate covariance, and correlation coefficient.

The goal for maintaining telemetry information considering multiple types of keys, storing multiple counters, and computing a multitude of statistics, is generality. Certain applications (e.g., traffic engineering), require coarser-grained information (e.g., aggregated per destination) to decide how to shift traffic to improve network utilization. Others required fine-grained information (e.g., for each containerized application), to understand application dynamics. More complex applications, such as intrusion detectors used for network security, require a diversified set of information (both fine- and coarse-grained) to detect attacks.

[12] https://opentelemetry.io/docs/specs/

8.8 Periodic and event-driven monitoring

The ACES monitoring framework will employ both periodic and event-driven approaches. For instance, the enriched records mentioned in the previous section can be sent to the network control plane or to other nodes of the ACES monitoring infrastructure either periodically or in a push-based manner. For the latter, we will investigate the integration of a traffic change primitive in the network switches.

Traffic changes are commonly associated with events that require special attention from the operator. They may be an indicator of a malicious attack on the network, of a bottleneck caused by a flash crowd, or can be a sign of persistent congestion. The ability to detect traffic changes fast and efficiently is therefore a fundamental requirement of many network operation tasks. A change detector primitive avoids a difficult operational question: what should be the periodic timer interval? If too large, one may miss important events (e.g., an attack); if too small, it will generate unacceptable network overhead. A change detector is thus the enabler for the event-based mechanism we plan to integrate into ACES.

The challenge is implementing this primitive on the very restrictive compute and memory environment that is a network switch. Our approach is the use of sketches [44,45,46]. Sketches are space-efficient and provide probabilistic memory-accuracy guarantees, enabling the design of efficient and scalable monitoring solutions for the network data plane. Several sketch-based systems have been recently proposed, running different network monitoring tasks at line rates in high-speed server platforms or on commodity switches. These modern systems are, however, restricted to heavy-hitter variants, and none has considered the general problem of change detection we will consider in ACES.

9 Cognitive Framework

The cognitive framework within the ACES platform is designed to enhance the edge data center with autopoietic capabilities, allowing it to self-maintain in an autonomous manner. This capacity for autopoiesis ensures that the EMDCs can adaptively manage their own resources for optimal performance, resilience, and efficiency, mirroring living systems' ability to be self-sustaining. In this regard, the cognitive framework facilitates advanced cognition in the systems software stack by taking into account monitoring data, learning from previous events and interactions, and making informed decisions to maintain optimal system conditions.

The cognitive framework's capabilities extend to include knowledge acquisition, whereby the system continuously learns from the environment and user interactions. It processes this information to understand the patterns and feedback, thereby enabling predictive analytics and prescriptive actions to maintain system health and manage the workload dynamically. With an embedded machine learning lifecycle management, the framework supports continuous integration and deployment (CI/CD) of AI models, ensuring that they are always up-to-date and trained on the latest data.

Additionally, the cognitive framework includes a feedback loop mechanism that enables the system to monitor its own performance and take corrective actions autonomously. This loop, driven by monitoring tools and AI-driven analytics, enables a proactive response to potential issues, thereby maintaining system reliability and stability. As part of cognition, the framework ensures data-driven decision-making that aligns with the broader organizational objectives and performance metrics, empowering the EMDC to act and react in an intelligent and informed manner.

Finally, the cognitive framework promotes a symbiotic relationship between different EMDCs and the cloud, leveraging particular high-performance resources (Cloud) for heavy-duty processing and storage while maintaining edge-specific operations for real-time and low-latency tasks. The achieved balance enables a seamless execution and scalability along with a distributed approach to data processing, where the offloading decisions will be made by the cognitive engine based on workload requirements and resources availabilities and conditions.

9.1 Autopoiesis and predictive analytics

The foundation for autopoiesis is the detailed monitoring to continuously collect data on every aspect of an EMDC's operation. This involves the monitoring infrastructure described in Chapter 8 which will be implemented to aggregate and store the high-volume data, feeding it into the cognitive framework for processing and analysis.

With data collected, the next step is for the cognitive framework to analyse and interpret this information, identifying patterns and anomalies. Tools like Apache Spark can be used to handle large-scale data analytics, applying algorithms to recognize normal operating parameters and detect deviations.

Leveraging historical and real-time data, the cognitive framework employs predictive analytics to forecast future conditions and potential issues before they arise. For example, it may predict resource shortages, potential component failures, or periods of high demand using time-series forecasting models or AI-based anomaly detection systems.

Based on these details a feedback loop mechanism will be provided in order to enable a self-regulating EMDC. Driven by the insights gained from monitoring data and predictive analytics, the cognitive framework determines prescriptive actions, adhering to a set of predefined rules, objectives, or learned experiences. These actions are aimed at maintaining system health, such as triggering automated scaling to meet increased demand, redistributing loads across servers, or initiating preventive maintenance. Decision-making is supported by AI models that evaluate multiple scenarios and their potential impact on the system's performance and health.

9.2 ML models for the Edge

Edge-specific ML models must address unique challenges such as limited computational power, constrained memory, sporadic connectivity, and latency-sensitive applications. The nature of edge computing favors algorithms that can operate with high efficiency on small datasets, often in real-time. For example, decision tree-based models or compact neural networks are preferable for such constraints and can be implemented using libraries like Scikit-learn or TensorFlow Lite. Anomaly detection and predictive maintenance are typical uses of ML at the edge, where immediate processing is critical, leveraging models such as Isolation Forests or LSTM neural networks.

To facilitate lightweight ML models for edge environments, we must consider methods that reduce model size and complexity. One approach is model compression, which can be achieved using pruning techniques supported by libraries like TensorFlow's Model Optimization Toolkit, which streamline the network architecture by removing unnecessary weights. Quantization, done via TensorFlow Lite or PyTorch's torch.quantization, can also reduce the precision of the numbers used in the model, thereby speeding up inference times and decreasing memory usage. Another technique to explore is splitting computation through cloud-edge collaboration: computing partially at the edge while offloading more complex processing to the cloud.

9.3 Edge Computing for ML

In the context of Edge Computing for AI-based applications, state-of-the-art collaborative and distributed machine learning (ML) paradigms such as Federated Learning (FL) and Split Learning (SP) can greatly benefit from the ACES platform, owing to its intelligent design and distributed edge cloud computing capabilities.

Federated learning (FL)13 is an innovative collaborative ML approach. In FL, clients train model updates locally based on their data (and a shared global model), then send these updates to a central aggregator. This aggregator combines them into a new global model, which is redistributed to clients for further training iterations. FL is efficient and scalable, distributing training across numerous clients and executing it in parallel. Crucially, by allowing clients to retain their training data locally, thus, FL enhances privacy of clients' data. This aspect is vital for compliance with privacy regulations like the GDPR and is generally beneficial when handling personal and sensitive data. Applications of FL include next-word prediction for mobile keyboards, medical imaging, and intrusion detection systems.

Split learning (SL)14, another emerging collaborative ML paradigm, trains or infers ML models without sharing raw data between clients. In a typical SL setup, each client trains a partial deep network, a designated cut layer. The outputs at this layer are sent to another entity (server or client), which completes the training. This process ensures that raw data privacy is maintained, as only specific gradients from the cut layer are shared. The training continues until the model is fully developed without compromising the privacy of raw data.

The ACES platform facilitates easy deployment, computational efficiency, and security for these emerging collaborative ML paradigms. Its distributed edge infrastructure enables multiple clients in FL and SL to collaboratively train ML models in an efficient and privacy-preserving manner. For instance, each client in an ACES node can adapt flexibly to the dynamic and heterogeneous computing and storage resource demands.

[13] https://federated.withgoogle.com/

[14] https://www.media.mit.edu/projects/distributed-learning-and-collaborative-learning-1/

9.4 Combining Swarm Algorithms with AI/ML

The combination of AI/ML with swarm algorithms can lead to sophisticated models capable of managing distributed resources in edge computing scenarios. Swarm intelligence principles, derived from organisms' natural behaviour like birds flocking or fish schooling, can be applied to solve optimization problems from the bottom up. These techniques can be combined with AI using, e.g., reinforcement learning (RL) algorithms, accelerating the learning process through hyperparameter tuning. This has already shown interesting results in the context of workload placement but most of the related works have focused mainly on theoretical studies [47] in various libraries like Stable Baselines or Reinforcement Learning Toolkit (RLTK) or others15. By using RL in conjunction with swarm intelligence, distributed systems can autonomously determine optimal configurations and resource allocations, which plays an important role in the dynamic environments of edge computing.

9.5 Explainability in AI

Explainable AI (XAI) refers to artificial intelligence systems designed to be transparent and understandable to humans. XAI aims to make AI model outputs more interpretable, fostering user trust and comprehension. The imperative for explainability in AI within the edge ecosystem lies in its ability to provide transparency and build trust amongst users. Within ACES, we are exploring technologies to enhance understanding of how the ACES cognitive engine functions, including the relationships between internal and external data points and machine-based decision-making. For example, Libraries such as SHAP offer capabilities to demystify the opaque decision-making processes of machine learning models, enabling users to understand and trust their automated operations. For neural networks, techniques like gradient-based saliency maps or class activation mappings, which are available in libraries like tf-explain, offer visual interpretations of which parts of the input contribute to the model's predictions. Explainability can be taken into account in the MLOps lifecycle, ensuring that it is rather a consistent feature of the AI systems16 deployed at the edge.

9.6 MLOps and model lifecycle management

The ACES cognitive framework is designed to streamline the entire machine learning (ML) model lifecycle, enabling everything from data preparation and model training to deployment and monitoring. By implementing an MLOps approach, EMDCs can effectively manage the deployment of AI models at the edge, ensuring that they are consistently trained on the latest data and optimized for the unique constraints of edge environments. With the infrastructure required to support ML models continuously evolving, this MLOps framework ensures agility and adaptability through automated workflows and standardized processes. Thus, the edge micro data centre benefits from the MLOps capabilities by minimizing human intervention and providing a systematic way to track model versioning and performance over time.

Specifically, the MLOps framework within EMDCs defines a common set of services and functions that are applicable across various ML models, thereby avoiding redundancy and simplifying the service architecture. Through REST APIs and supporting libraries, it offers interfaces for ML model registration, training, deployment, and serving, thereby supporting a seamless transition from model development to production. This not only accelerates the time-to-market for AI-powered applications at the edge but also ensures these applications are robust and scalable.

The ACES MLOps component will be composed of the following subcomponents:

- **The Model trained Registry interface:** Users can register different ML models after they trained the models.
- **The ML Training interface:** the interface to integrate different ML models' training methods. With the integration, users can use the unified format to call the different ML training models

[15] https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python
[16] https://github.com/mlflow/mlflow/pull/3513

and set up the related training configuration especially for distributed ML models. For example: when to trigger training, the parameters of the training process, etc.

- **The ML Serving interface:** the interface for prediction/inference. With this interface, users can use the unified format to call the different ML prediction methods.
- **The ML validation interface**: the interface to integrate different ML models validation methods. With this interface, users can use the unified format to use the different ML validation function or use some default validation methods in the cognitive framework.
- **The ML model provider interface:** for integration of different ML models. It provides a unified API for the users to choose and use a ML model from those registered in the cognitive framework.
- **The ML monitoring interface:** responsible for the communication and aggregation of distributed training nodes, to track all the ML process data and provide different information for monitoring services.
- **The ML Security interface:** helps setting up security parameters and integrate security specific methods.

9.6.1 Cognitive framework tools and open-source libraries

The Cognitive Framework will be developed using the python language, supporting different python-based libraries like scikit learn and TensorFlow. The Flask API will be used to develop all the APIs.

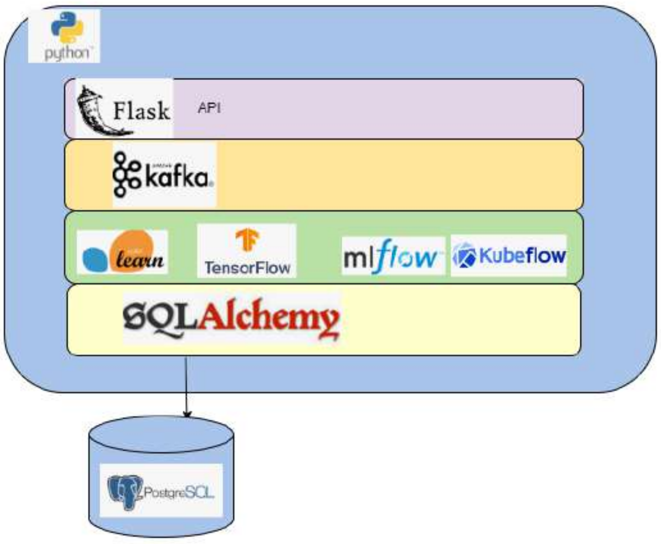

Figure 9.3: Cognitive framework tools

Each component of the cognitive Framework will include metadata collection, using kafka message queue technology. For ML model related training, serving, and deployment components, we will use the MLflow and kubeflow opensource tools. For the object relational mapper (ORM), we plan to use SQLAlchemy for python to communicate with a postgresDB.

9.6.2 MLflow and kubeflow tools

The Cognitive Framework will provide the whole ML model lifecycle management. This includes the model register, model tracking, model training, model deployment, model serving, and model monitoring.

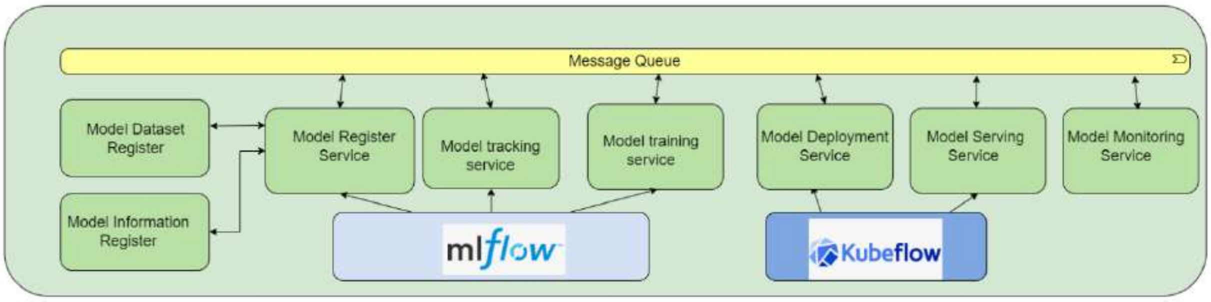

Figure 9.4: MLFlow and Kubeflow integration

To integrate different ML models and support different functions of the ML lifecycle, we plan to use and improve upon open-source tools MLflow and Kubeflow.

MLflow focuses on track model runs, including model parameters, metrics, results, data used, and code. It provides the model register and model tracking components that fit our cognitive framework functions requirement. It uses the jupyter notebook to save the model code and parameters. After the training process, all model related information is recorded.

The Kubeflow tool is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable, and scalable. It provides a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures.

10 Security and Privacy

Cloud-edge services process and store large volumes of (sensitive) data, making them vulnerable to cyber-attacks such as data breaches and unauthorized access, leading to potential misuse of data, data loss or data privacy violation. Moreover, the inherent interconnectedness of cloud-edge networks makes them susceptible to network-based Distributed Denial of Service (DDoS) attacks. Such attacks can overload network resources, severely disrupting service availability. Another critical concern is the risk of Man-in-the-Middle (MitM) attacks. These occur during data transfers between edge devices and cloud servers, potentially compromising the integrity and confidentiality of the data. Furthermore, the complexity of various regional and industry-specific regulations, such as the General Data Protection Regulation (GDPR), poses significant challenges in cloud-edge settings. This complexity is exacerbated when data is processed and stored across multiple locations, complicating compliance efforts.

In ACES, we recognize that protecting systems, services, and data against security and privacy threats is crucial to secure the system, comply with regulations, and meet customer requirements. Therefore, we aim to incorporate comprehensive security features into the system to ensure that ACES is secure by design. In the following sections, we will identify specific security and privacy requirements by carefully considering the use-case requirements (defined in Section 2.3) and the ACES architecture (Section 3.2). We provide a systematic design for ACES security components based on these requirements. This design is aimed at ensuring that ACES is resilient to cyberattacks, thereby safeguarding the availability, integrity, confidentiality, and privacy of ACES systems and data. We aim to prevent unauthorized access, service disruptions, data leakage, and data loss.

10.1 Security requirements

We are guided by three general security and privacy requirements:

- **Authentication:** The system should have robust authentication mechanisms such as access controls, secure communication and data encryption to ensure only authorized parties can access the service and their data.
- **Availability and Integrity:** The system should ensure that the computing services and data are accurate, consistent, and reliable against cyberattacks. The system should implement measures to detect and prevent unauthorized access and modification of data. Further, the system should have effective network intrusion and anomaly detection to monitor network traffic, nodes, and containers, to detect and mitigate suspicious behaviours caused by cyberattacks such as DoS attacks.
- **Privacy:** The solution must comply with data privacy regulations such as GDPR. Further, the solution should be able to prevent data leakage to any unauthorized parties.

10.2 Overview of security and privacy component

To achieve the requirements mentioned above, we provide a comprehensive multi-layer security solution aligned with the overall ACES architecture provided in Section 3. Figure 10.1 provides a high-level overview of the security components. This consists of five sub-components that will be elaborated in the following sections.

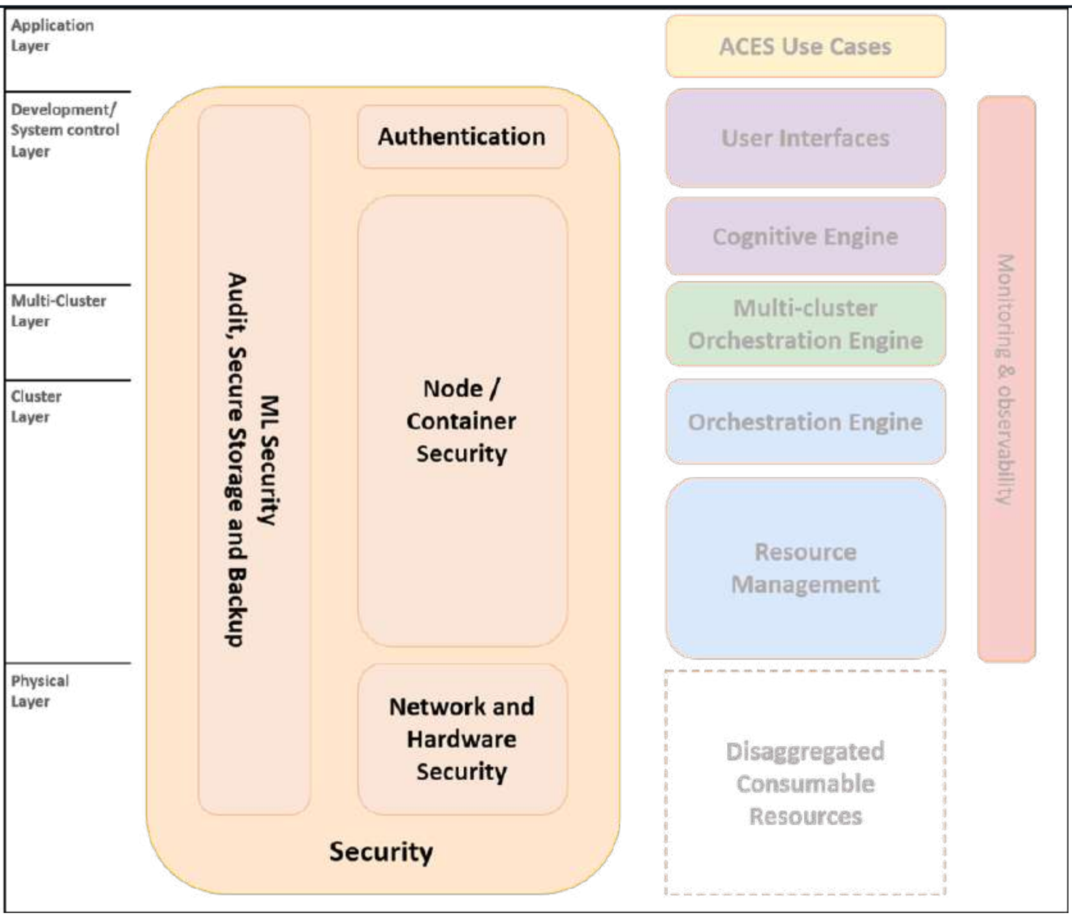
Application Layer
Development/ System control Layer
Multi-Cluster Layer
Cluster Layer
Physical Layer
Authentication
ML Security
Audit, Secure Storage and Backup
Node / Container Security
Network and Hardware Security
Security
ACES Use Cases
User Interfaces
Cognitive Engine
Multi-cluster Orchestration Engine
Orchestration Engine
Resource Management
Disaggregated Consumable Resources
Monitoring & observability

Figure 10.1: Overview of ACES Security Component

10.3 Authentication

Authentication is essential for any service to protect access from unauthorized parties. Given the remote locations of ACES nodes, privacy is also a requirement for ACES authentication. The proximity between ACES nodes and clients during authentication can compromise user privacy, potentially revealing sensitive information such as location data. Among the requirements of ACES are client privacy and compliance with European laws, namely the GDPR. We will develop anonymous authentication schemes leveraging public key encryption that will explore the introduction of pseudonyms for efficiency. The methods to be deployed should also preserve client privacy after possible revocation, introducing a new abstraction of non-revocation proofs. We will follow a fully distributed design, respecting the principles of Verifier Local Revocation (VLR), a crucial aspect for ACES to maintain scalability and avoid reliance on centralized services that can become single points of failure.

10.4 Audit, secure storage and backup

ACES will ensure fault tolerance through the replication of its services and storage. The ACES data and intelligence will be replicated horizontally across edge sites, operating within a zero-trust environment. Given the vulnerability of edge sites to failures, corruptions, or attacks due to their exposed nature, an inherent security risk could jeopardize the fidelity of ACES replication. We will, therefore, introduce auditing tools to assess the correct level of ACES replication at the edge. In response to this challenge, we will develop storage-proof mechanisms designed to audit the location of data in distributed settings, such as the edge environment. The cryptographic proof should pinpoint data locality with millisecond precision despite the variations in network delays at the edge, and it should be able to detect SLA violations. This auditing tool further enables distributed entities to build trust at the edge.

10.5 Network and hardware security

In ACES, we will develop an ML-based attack detector with two key characteristics. First, it is based on the analysis of network meta-data, instead of payload traffic. As such, it can be used to detect attacks *that use encrypted traffic*, in contrast to conventional signature-based malicious traffic detectors that work only on unencrypted traffic (e.g., Snort). Second, it works by tracking deviations from regular traffic patterns to detect attacks, enabling the detection of zero-day attacks. To address the main performance of existing systems –the overhead of the ML pipeline processing – we will develop a cross-platform malicious traffic detector. The key idea is to offload the detection process to the network data plane. Specifically, we aim to run the ML feature computation in a network switch. The ACES switch should process a diverse set of flow statistics as ML features of types, without the need to inspect packet payloads. By computing features in the switch, we avoid the required packet sampling of state-of-the-art detectors to improve detection performance in the ACES Terabit network.

10.6 Node and container security

Container-based applications are increasingly being adopted due to their convenience in development, deployment, and management. In ACES, containers are fundamental elements for deploying ACES agents and services. However, recent studies show that containers are vulnerable to various security attacks e.g., Authentication Bypass, Disclosing Credential Information, and Denial of Service (DoS) attacks. Such attacks allow attackers to open a reverse shell or establish a backdoor within a container by exploiting specific vulnerabilities. The successful exploitation grants the attacker complete control over the container. Further, Disclosing Credential Information Attacks strive to uncover usernames, passwords, or the directory structure of the underlying OS such that the attacker can impersonate a legitimate user, altering, deleting, or stealing valuable data. DoS Attacks target containers and container services, rendering them inoperable by exhausting the containers or host's resources. An illustrative example is a clandestine cryptocurrency miner operating within a container, excessively consuming resources and inhibiting the container's primary functions. To secure the ACES system, we will build a framework that involves a systematic method to effectively analyse and evaluate anomaly detection models. We will perform extensive analyses regarding the different types of attacks and defenses existing for the container ecosystem. This will provide a clear view regarding the state-of-the-art attack techniques and defenses for container systems. More importantly, we will develop novel defense approaches to prevent state-of-the-art attacks effectively. Our approach will leverage advanced techniques, such as vulnerability scanning and dynamic deep-learning-based anomaly detection, to detect not only vulnerabilities but also attacks in real time. To this end, our container security component is designed to monitor and safeguard containers against state-of-the-art threats, such as privilege escalation, credential disclosure, or DoS attacks.

10.7 ML security

Machine Learning (ML) is pivotal in powering ACES intelligence agents and services. However, ML algorithms and frameworks are vulnerable to security and privacy threats. In ACES, we will develop components incorporating multiple security mechanisms to defend against data and model poisoning attacks, as well as inference attacks, in distributed learning systems like federated learning—a key potential service within ACES—and AI/ML-based network control. These components should bolster security against backdoor and inference attacks.

In backdoor attacks, adversaries subtly manipulate the global model, causing specific inputs the attacker chooses to produce incorrect predictions. An adversary might also introduce multiple backdoors simultaneously. Meanwhile, in inference attacks, adversaries attempt to glean information about clients' local training data by analysing their model updates. To counter backdoor attacks, we propose a comprehensive strategy that employs cutting-edge defense techniques, including model clustering, clipping, and parameter noising. Additionally, we integrate several privacy measures to thwart inference attacks, such as secure two-party computation techniques (STPC), trusted computing, and blockchains. These measures restrict access to local model updates, thereby hindering potent inference attacks.

11 System integration

This section summarises the ACES components interfaces and APIs specifications along with their main responsibilities and technologies to be based upon. Furthermore, it describes the integrated development and testing environment upon which the ACES solution platform is built; including the Continuous Integration/Continuous Delivery processes put in place to support all the development, testing, integration, and deployment activities.

11.1 ACES components, interfaces and interactions

In total, a set of 9 components was defined, as follows:

- The ACES frontend component, including the Authentication and Authorization interface;
- The Workflow Management Component for the definition of applications as graphs of microservices;
- The Cognitive Engine and MLOps component;
- The Orchestration component composed of a multi-cluster and a inter-cluster workload placement based on swarm intelligence;
- The Data Management component featuring the decentralized storage functionality;
- The Resource Management and Containerization Runtime component;
- The Networking component;
- The Monitoring and Telemetry component;
- The Security component.

Each of the components is capable of performing a specific set of actions / functionalities and addressing a specific set of requirements. On the other hand, the conceptual representation of ACES architecture aims at integrating all identified components into a logical diagram, facilitating a complete information flow, comprising the preliminary version of the conceptual architecture of the integrated platform.

The architecture is designed in a modular way facilitating easy maintenance, modifiability and extensibility, and can thus be used and easily extended and customised accordingly in order to include *end users*, *data scientists* and *stakeholders* needs not considered until now, including new inputs to reach different needs of interested parties, as well as new ones.

Figure 3.1 (Section 3.1) depicted the high-level ACES architecture diagram including the main architectural components, information flows and interactions among them. The ACES architecture follows a layered approach which aims at ensuring interoperability among all involved components, putting emphasis on the way that pipelining of information is supported, safeguarding smooth interoperation of the supported services.

The ACES architecture is defined in such a way that every component can be independent while bringing a particular functionality. Each component in ACES may consist of different internal software modules or sub-components while having the capability to interact with other components via a particular interface, usually an API. This sub-section will provide a summary of, per component: responsibilities, main technologies used, interactions with other components, and main interfaces. This is shown in the following tables.

Name	ACES frontend interface
Description	This component is responsible for providing the frontal interface of ACES platform featuring the authentication & authorization panel along with the connection to the different system or user level components
Responsible Partner	HIRO
Function	ACES frontend, authentication & authorization panel and centralization of the various ACES interfaces
Subsystems	Authentication & authorization, Centralization of interfaces
Type of Interface	Web, REST
Technologies	Django, React
Interaction with components	Workflow Management, Data Management, Cognitive Engine, Monitoring

Name	Workflow Management
Description	This component is responsible for enabling the creation of applications through workflows/graphs of microservices
Responsible Partner	MAR
Core Partners	HIRO, UL
Function	Workflow design, microservices connection in workflows, microservices packaging
Subsystems	-
Type of Interface	REST
Technologies	Prefect, Ryax, Airflow
Interaction with components	Data Management, Cognitive Engine, Orchestration, Resource Management

Name	Cognitive Engine

Description	This component is responsible for selecting the right ML algorithms for the different aspects needed for the system and performing the MLOps for system and user-level needs
Responsible Partner	HIRO
Function	MLOps platform
Subsystems	Cognitive Engine and MLOps
Type of Interface	REST
Technologies	Kubeflow, MLFlow, Feast
Interaction with components	Data Management, Orchestration, Resource Management

Name	Orchestration
Description	This component is responsible for enabling the orchestration in either the multi-cluster or the inter-cluster scenario
Responsible Partner	LAKE
Core Partners	UPM, HIRO, UL
Function	Container orchestration and scheduling
Subsystems	multi-cluster scheduler, single cluster scheduler
Type of Interface	REST
Technologies	multi-cluster scheduling or Kubernetes scheduler
Interaction with components	Data Management, Orchestration, Resource Management

Name	Data Management
Description	This component is responsible for managing the ACES knowledge base that captures the supply, demand, and current runtime state of the platform.
Responsible Partner	UPM

Core Partners	MAR
Function	-
Subsystems	Knowledge model, property graph storage component, time series data storage component.
Type of Interface	REST
Technologies	MemGraph, InfluxDB
Interaction with components	Workflow Management, Orchestration, Resource Management

Name	Resource Management
Description	This component is responsible for i) application orchestration at the level of EMDCs and Kubernetes clusters and ii) EMDC and/or nodes' management. Applications' orchestration will be driven by the cognitive engine component
Responsible Partner	SIXSQ
Core Partners	HIRO
Function	configure a Container-as-a-Service Kubernetes endpoints, launch applications of the Kubernetes cluster
Subsystems	Kubernetes, telemetry tools
Type of Interface	REST
Technologies	Nuvla/NuvlaEdge. Kubernetes, CRIO
Interaction with components	Workflow Management, Cognitive engine

Name	Monitoring and Observability
Description	This component is responsible for providing monitoring and observability aspects to the different layers of the software stack on various levels (i.e., edge, cloud, application, and network layer)
Responsible Partner	UL

Function	Monitoring data collection, instrumentation, telemetry, anomaly detection, alerting, data analysis and visualization
Subsystems	Monitoring and Observability Core, Push Gateway, Data Forwarder, Anomaly Detection, Alert Manager, Service Discovery Mechanism, Data Analysis, Export and Visualization
Type of Interface	REST, graphical (web), custom pluggable
Technologies	Prometheus, Thanos
Inputs	ACES monitoring and telemetry data, metrics
Interaction with components	All

Name	Networking
Description	This component is responsible for enabling the networking capabilities for the edge micro data centers
Responsible Partner	INESC
Function	-
Subsystems	Single-cluster networking and multi-cluster networking
Type of Interface	REST
Technologies	Cilium, Istio, Submarinner, ONOS. P4, P4RT
Interaction with components	Workflow Management, Orchestration, Resource Management

Name	Security
Description	This component is responsible for protecting ACES systems, services, and data against security and privacy threats
Responsible Partner	TUD
Function	Security and privacy solutions for monitoring and protecting ACES components, services, and data

Subsystems	Secure authentication and storage, network security, node and container security, ML security
Type of Interface	Web, REST
Technologies	Sysdig, NIDS, Anomaly Detection
Inputs	ACES metrics
Interaction with components	Workflow Management, Orchestration, Resource Management

11.2 ACES platform integration environment

The ACES platform ensures consistency and quality through a structured process including integration, testing, release, distribution, and deployment. The process is designed to adhere to high-quality standards, and it includes standard approaches to packaging, distribution, deployment, and documentation. ACES uses Continuous Integration and Continuous Deployment (CI/CD) to automate these processes. The benefits of this system include quicker delivery and feedback, less manual effort, and fewer errors.

Whenever a change is made to a component, an automatic pipeline is initiated consisting of the following steps:

1. Compiling the component into a binary file from its source code.
2. Running unit tests and checks on the component.
3. Packaging the component in Docker Images, and creating Helm charts for Kubernetes deployment.
4. Deploying everything into a test environment.

To enable this, an on-site infrastructure is being assembled, including:

- A GitLab server for managing source code and collaboration.
- A Harbor server for hosting Docker Images and Helm charts.
- A Kubernetes cluster to serve as the testing ground.
- Tools like Kaniko for Docker Image builds.
- Other infrastructure tools such as LDAP, a mail server, and logging and analysis tools.

There are also internal tools developed to streamline the pipeline, such as 'cicd-scripts' for job templates, a 'cicd-helper-image' that includes necessary tools, and a Helm 'common' chart for simpler Helm chart development.

Pipeline runs are routinely executed as parts of components are updated, producing reports and artifacts that are stored on GitLab. However, not all these builds are stable or ready for release; they are often for development and integration checks only. A release occurs once the version is stable and verified—this action is facilitated by GitLab's "Releases" feature, allowing developers to automate the release process by labelling the version, noting the release, and thus triggering the pipeline and storing the final artifacts on the Harbor repository. All ACES component releases are maintained in a centralized public repository, ensuring unified access and management.

The ACES platform is built to work across different settings, including mainly on-premises and edge micro data centres, as well as cloud-based infrastructures. Due to its versatility and the range of components it includes, there's a need to employ simple and reliable deployment methods that minimize the risk of errors. To achieve this, Infrastructure as Code (IaC) tools like Terraform are used. Terraform is the chosen tool because it is highly regarded for setting up infrastructure efficiently and safely. Being an open-source tool, it has a large support community and works with various infrastructures.

The deployment process for ACES is split into two main stages. In the first stage, Terraform scripts are pulled from the central ACES Git repository to set up the infrastructure where the ACES components will be housed. In the following stage, Helm charts are employed to install and configure these components in the Terraform-prepared environment. Anyone deploying the ACES platform will just need the Terraform and Helm clients, as well as the necessary resources for ACES to operate.

12 Conclusion

This deliverable presented the high-level view of the architecture of ACES along with the different research areas and components related to the ACES platform. The architecture is generic enough to remain as is until the end of the project, whereas the refinement of the list of components will take place in the upcoming months. The final set of components to be implemented will be presented in the upcoming deliverable D2.2a ACES kernel components. In case there are updates in the ACES architecture, these will be presented in that intermediate deliverable.

13 References

[1] https://www.sciencedirect.com/science/article/pii/S0160791X23001203
[2] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9744527
[3][https://www.scopus.com/record/display.uri?eid=2-s2.0 84987842183&origin=inward&txGid=f8351620c2647bed6c937753f185652e]
[4] [https://www.sciencedirect.com/science/article/pii/S2666792420300068]
[5] Andrew Jeffery, Heidi Howard, Richard Mortier: Rearchitecting Kubernetes for the Edge. CoRR abs/2104.02423 (2021)
[6]L. Larsson, H. Gustafsson, C. Klein and E. Elmroth, "Decentralized Kubernetes Federation Control Plane," 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), Leicester, UK, 2020, pp. 354-359, doi: 10.1109/UCC48980.2020.00056.
[7] Zhang Wei-guo, Ma Xi-lin, and Zhang Jin-zhong. 2018. Research on Kubernetes' Resource Scheduling Scheme. In Proceedings of the 8th International Conference on Communication and Network Security (ICCNS '18). Association for Computing Machinery, New York, NY, USA, 144–148. https://doi.org/10.1145/3290480.3290507
[8] Malte Schwarzkopf: Cluster Scheduling for Data Centers. ACM Queue 15(5): 70 (2017)
[9] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, John Wilkes: Borg, Omega, and Kubernetes. Commun. ACM 59(5): 50-57 (2016)
[10] B. Hindman et al, "Mesos: A platform for fine-grained resource sharing in the data center," in NSDI'11 Proceedings of the 8th USENIX conference on Networked systems design and implementation, 2011.
[11] https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/
[12] https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/
[13] https://github.com/containernetworking/cni
[14] https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
[15] https://k3s.io/
[16] https://microk8s.io/
[17] https://github.com/kubernetes-sigs/federation-v2
[18]Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, Jörg Ott:Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. USENIX Annual Technical Conference 2023: 215-231
[19]Marcos Dias de Assunção, Alexandre Da Silva Veith, Rajkumar Buyya: Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. J. Network and Computer Applications 103: 1-17 (2018)
[20]Zhang Wei-guo, Ma Xi-lin, and Zhang Jin-zhong. 2018. Research on Kubernetes' Resource Scheduling Scheme. In Proceedings of the 8th International Conference on Communication and Network Security (ICCNS '18). Association for Computing Machinery, New York, NY, USA, 144–148. https://doi.org/10.1145/3290480.3290507
[21] Schranz, M., Di Caro, G. A., Schmickl, T., Elmenreich, W., Arvin, F., Şekercioğlu, A., & Sende, M. (2021). Swarm intelligence and cyber-physical systems: concepts, challenges and future trends. Swarm and Evolutionary Computation, 60, 100762.
[22] Hamann, H. (2018). Swarm robotics: A formal approach (Vol. 221). Cham: Springer.
[23] Szopek, M., Schmickl, T., Thenius, R., Radspieler, G., & Crailsheim, K. (2013). Dynamics of collective decision making of honeybees in complex temperature fields. PloS one, 8(10), e76250.
[24] Sempo, G., Canonge, S., Detrain, C., & Deneubourg, J. L. (2009). Complex dynamics based on a quorum: Decision-making process by cockroaches in a patchy environment. Ethology, 115(12), 1150-1161.
[25] Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of a path by the ant Lasius niger. Journal of theoretical biology, 159(4), 397-415.
[26] Bodi, M., Thenius, R., Szopek, M., Schmickl, T., & Crailsheim, K. (2012). Interaction of robot swarms using the honeybee-inspired control algorithm BEECLUST. Mathematical and Computer Modelling of Dynamical Systems, 18(1), 87-100.
[27] Gunther, N. J. (1993, December). A simple capacity model of massively parallel transaction systems. In Int. CMG Conference (pp. 1-9)

[28] Melanie Schranz, Kseniia harshina, Peter Forgacs, Fred Buining, Agent-based Modeling in the Edge Continuum using Swarm Intelligence, submitted to the ICAART Conference, 2024.
[29] Melanie Schranz, Gianni Di Caro, Thomas Schmickl, Wilfried Elmenreich, Farshad Arvin, Ahmet Şekercioğlu, Micha Sende, Swarm intelligence and cyber-physical systems: concepts, challenges and future trends. Swarm and Evolutionary Computation, 60, 100762, 2021.
[30] Melanie Schranz, Martina Umlauft, Wilfried Elmenreich, Bottom-up Job Shop Scheduling with Swarm Intelligence in Large Production Plants. In International Conference on Simulation and Modeling Methodologies, Technologies and Applications, pp. 327-334, 2021.
[31]R. Kumar, "White Box Monitoring and Black Box Monitoring explained," March 2020. [Online]. Available: https://www.devopsschool.com/blog/white-box-monitoring-and-black-box-monitoring-explained/. [Accessed 5. 12. 2023].
[32]C.-a. Sun, M. Li, J. Jia and J. Han, "Constraint-based model-driven testing of web services for behavior conformance," in Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16, 2018.
[33]Y. Yang, L. Wang, J. Gu and Y. Li, "Transparently capturing execution path of service/job request processing," in Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16, 2018.
[34]H. Johng, D. Kim, T. Hill and L. Chung, "Estimating the performance of cloud-based systems using benchmarking and simulation in a complementary manner," in Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16, 2018.
[35]O. Mart, C. Negru, F. Pop and A. Castiglione, "Observability in kubernetes cluster: Automatic anomalies detection using prometheus," in 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2020.
[36]J. Lin, P. Chen and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16, 2018.
[37]S. Niedermaier, F. Koetter, A. Freymann and S. Wagner, "On observability and monitoring of distributed systems–an industry interview study," in Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17, 2019.
[38]M. Usman, S. Ferlin, A. Brunstrom and J. Taheri, "A survey on observability of distributed edge & container-based microservices," IEEE Access, 2022.
[39]OpenTelemetry, "Specifications | OpenTelemetry," [Online]. Available: https://opentelemetry.io/docs/specs/. [Accessed 5. 12. 2023].
[40]"Query Standardization Working Group," [Online]. Available: https://github.com/cncf/tag-observability/blob/main/working-groups/query-standardization.md. [Accessed 5. 12. 2023].
[41]P. Authors, "Prometheus Remote-Write specification," April 2023. [Online]. Available: https://prometheus.io/docs/concepts/remote_write_spec/. [Accessed 5. 12. 2023].
[42]W3C, "Trace Context," November 2021. [Online]. Available: https://www.w3.org/TR/trace-context/. [Accessed 5. 12. 2023].
[43] Kernbach, S., Thenius, R., Kernbach, O., & Schmickl, T. (2009). Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system. Adaptive Behavior, 17(3), 237-259.
[44] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16). Association for Computing Machinery, New York, NY, USA, 101–114. https://doi.org/10.1145/2934872.2934906
[45] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: adaptive and fast network-wide measurements. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18). Association for Computing Machinery, New York, NY, USA, 561–575. https://doi.org/10.1145/3230543.3230544
[46] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: high-performance sketch-based measurement over arbitrary partial key query. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 207–222. https://doi.org/10.1145/3452296.3472892
[47] Alqarni, Mohamed A., Mohamed H. Mousa, and Mohamed K. Hussein. "Task offloading using GPU-based particle swarm optimization for high-performance vehicular edge computing." Journal of King Saud University-Computer and Information Sciences 34.10 (2022): 10356-10364.