A Cognitive Serverless Framework for the Cloud-Edge Continuum

D2.6 COGNIT Framework - Architecture - f

Version 2.0

27 February 2025

Abstract

COGNIT is an AI-Enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that enables the seamless, transparent, and trustworthy integration of data processing resources from providers and on-premises data centers in the cloud-edge continuum, and their automatic and smart adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This final version of the COGNIT Framework Architecture report provides a comprehensive and self-contained description of the completed COGNIT Framework, its components, capabilities, and implementation details as delivered at the end of the project.

This project is funded by the European Union's Horizon Europe research and innovation programme under Grant Agreement 101092711 – SovereignEdge.Cognit

Deliverable Metadata

Project Title:	A Cognitive Serverless Framework for the Cloud-Edge Continuum
Project Acronym:	SovereignEdge.Cognit
Call:	HORIZON-CL4-2022-DATA-01-02
Grant Agreement:	101092711
WP number and Title:	WP2. Adaptive Cloud-Edge Serverless Framework Architecture
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	1.0
Contractual Date of Delivery:	30/09/2025
Actual Date of Delivery:	27/02/2026
Lead Author:	Marco Mancini (OpenNebula)
Authors:	Mirko Stojiljković (OpenNebula), Christophe Ponsard (CETIC), Monowar Bhuyan (UMU), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), Cristina Cruces (Ikerlan), Martxel Lasa(Ikerlan), Jean Lazarou (CETIC), Fátima Fernández (Ikerlan), Aitor Garciandia (Ikerlan), Torsten Hallmann (SUSE), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Deins Darquennes (CETIC), Daniel Olsson (RISE), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Thomas Ohlson Timoudas (RISE), Paul Townend (UMU), Iván Valdés (Ikerlan).
Status:	Final

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	01/12/2025	Draft	Initial Draft
0.2	22/12/2025	Peer-Reviewed	Reviewed Draft
1.0	31/12/2025	Submitted	Final Version
1.1	26/02/2026	Draft	Initial Draft
1.2	27/02/2026	Peer-Reviewed	Reviewed Draft
2.0	27/02/2026	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.1	29/12/2025	Joel Höglund (RISE)
0.1	29/12/2025	Antonio Álvarez (OpenNebula)
1.2	27/02/2026	Antonio Álvarez (OpenNebula)

Summary of Changes from Previous Versions

Second Version of Deliverable D2.6

[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

Deliverable D2.6, released at the end of the project, is the final self-contained version of the COGNIT Framework Architecture. This report provides a comprehensive description of the completed COGNIT Framework, including all implemented components, their interactions, capabilities, and the underlying technologies used for implementation.

The COGNIT Framework addresses the challenges of executing computationally intensive applications on resource-constrained edge devices by providing an AI-Enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum. The framework enables edge devices to offload function executions to a dynamically managed cloud-edge infrastructure that smartly adapts to application requirements, infrastructure conditions, and environmental sustainability metrics.

The completed COGNIT Framework consists of five main architectural components:

- **Device Client**. A lightweight SDK available in both Python and C that allows edge devices to offload function executions to the cloud-edge continuum using a Serverless paradigm. It enables devices to execute functions remotely and interact with the COGNIT Framework through a simple API.
- **COGNIT Frontend**. The main entry point for the COGNIT service that authenticates device clients, manages application requirements and functions, and coordinates with the AI-Enabled Orchestrator to select the most suitable Edge Cluster for function execution.
- **Edge Cluster**. A semi-autonomous environment with dedicated software-defined resources (compute, network, and storage) that provides an abstraction layer enabling interoperability across the multi-provider cloud-edge infrastructure. Edge Clusters host **Serverless Runtimes** that execute functions offloaded by devices in a hardened and secure environment.
- **Cloud-Edge Manager**. The infrastructure management component that is responsible for managing the highly distributed and heterogeneous resources of the cloud-edge continuum, including the lifecycle management of Edge Clusters and Serverless Runtimes, monitoring, authentication, and execution of deployment plans produced by the AI-Enabled Orchestrator.
- **AI-Enabled Orchestrator**. A smart component that manages and optimises the lifecycle of Edge Clusters and Serverless Runtimes according to application requirements, infrastructure usage, and energy-aware policies. It uses AI/ML models and optimization algorithms to produce deployment plans that are executed by the Cloud-Edge Manager.

The COGNIT Framework has been successfully implemented and validated through four Use Cases spanning Smart Cities, Wildfire Detection, Energy Management, and Cybersecurity domains. The framework provides capabilities for optimal workload placement, dynamic resource scaling, energy-aware optimization, secure authentication and authorization, confidential computing support, and comprehensive monitoring and auditing.

This final document is structured to provide a complete reference for the COGNIT Framework architecture, including global and user requirements, detailed component specifications, software requirements, verification methodology, implementation technologies, and security considerations.

Table of Contents

Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge (From MITRE)
AWS	Amazon Web Services
CC	Confidential Computing
CD	Continuous Delivery (Deployment)
CIA	Confidentiality Integrity Availability
CRA	Cyber Resilience Act
FaaS	Function as a Service
FFD	First-Fit Decreasing
IaC	Infrastructure as Code
IAM	Identity and Access Management system
IP	Internet Protocol
ITSRM	Information Technology Security Risk Management Methodology
JSON	Javascript Object Notation
MAPE-K	Monitor, Analyze, Plan, Execute, Knowledge
ML	Machine Learning
OBS	Open Build Service
REST	Representational State Transfer
SysML	System Modeling Language
SDK	Software Development Kit
SR	Software Requirement
TEE	Trusted Execution Environment
UML	Unified Modeling Language
UR	User Requirement
VM	Virtual Machine
VS	Verification Scenario

Introduction

The COGNIT Framework addresses the challenge of enabling resource-constrained edge devices to leverage cloud-edge continuum resources for computationally intensive data processing. Traditional cloud computing solutions assume centralized infrastructure and are not optimized for the highly distributed, heterogeneous nature of the cloud-edge continuum. Existing serverless platforms focus on cloud-centric event-driven models with short-lived functions, whereas edge applications require sustained compute-intensive processing with low latency and energy efficiency.

COGNIT bridges this gap by providing an AI-Enabled Adaptive Serverless Framework specifically designed for the cloud-edge continuum. The framework enables edge devices to offload function executions to intelligently managed resources across multiple providers and locations, with automatic adaptation to changing application requirements, infrastructure conditions, and environmental sustainability goals.

This final deliverable (D2.6) provides a comprehensive and self-contained description of the completed COGNIT Framework architecture. The document is organized as follows:

- **Part I** (Sections 2-3) defines the global requirements framework, including sovereignty, sustainability, interoperability, and security requirements, as well as user requirements extracted from the four project Use Cases.

- **Part II** (Sections 4-9) provides the complete architecture definition, detailing the COGNIT Framework components, their interactions, capabilities, and software requirements.

- **Part III** (Sections 10-11) describes the verification methodology, verification scenarios, and the concrete technologies used to instantiate the COGNIT architecture.

- **Part IV** (Sections 12-13) presents the overall development and implementation status of the project, including software requirement progress, global KPIs, and detailed implementation status for each component. The document concludes with Section 13, which summarizes the key achievements and contributions of the COGNIT Framework.

Throughout this document, we describe the architecture as implemented and validated in the project. All software requirements have been completed except for Federated Learning.

PART I. Global and User Requirements

2. Sovereignty, Sustainability, Interoperability, and Security Requirements

This section summarises the results of the analysis of the European context in order to ensure that, by meeting a number of relevant, transversal sovereignty, sustainability, interoperability, and security requirements, the architecture of the COGNIT Framework is in line with EU digital policies and strategic priorities related to the Cognitive Cloud. This requirement analysis, which also incorporates sector-specific priorities from some of the Project's Use Cases, guided the design and implementation of the COGNIT framework.

2.1. Sovereignty Requirements

Sovereignty, in the context of the development of a European cloud-edge continuum, involves the consolidation of the leadership and strategic autonomy of the EU in the digital world. The EU's New Industrial Strategy, for instance, links that global objective with the need to build solutions capable of leveraging the deployment of 5G and edge infrastructures, as well as competitive European alternatives for the multi-cloud, following an open source model.

It is also worth noting that the European Commission's Open Source Software Strategy 2020-2023 also states that *"open source impacts the digital autonomy of Europe. Against the hyperscalers in the cloud, it is likely that open source can give Europe a chance to create and maintain its own, independent digital approach and stay in control of its processes, its information and its technology"*. With all those priorities in mind, Table 2.1 below provides the list of sovereignty requirements that has been addressed during the execution:

Id	Description	Source
SOR0.1	The COGNIT Framework shall be able to leverage public, private, and self-hosted cloud and edge infrastructures hosted in the European Union.	All
SOR0.2	The implementation of the COGNIT Architecture shall maximise the use of European open source technologies and frameworks.	All
SOR0.3	The COGNIT Framework shall provide an abstraction layer that ensures workload portability seamlessly across different infrastructure providers.	All
SOR0.4	Data handling by the COGNIT Framework shall be compliant with the GDPR.	All

Table 2.1. Global Sovereignty Requirements.

2.2. Sustainability Requirements

These global requirements aim at reducing the ecological footprint of the COGNIT Framework (reported in Table 2.2) in line with the European Green Deal's objective to create a climate neutral continent and assuming that, in line with the Digital Decade objectives, our solution will have to be able to leverage at some point the *"10,000 climate neutral highly secure edge nodes"* that are expected to be deployed throughout the EU by 2023. Sustainability is the focus of the Project's Research Challenge 6—*"Optimization of energy efficiency and adaptation to variable (local) green energy supply throughout the cloud-edge continuum"* stated in the Grant Agreement; this challenge has been studied in particular as part of the Energy Use Case (UC3), where the COGNIT Framework leverage AI/ML techniques to reduce energy usage in a household, consequently reducing its energy footprint.

Id	Description	Source
SUR0.1	Sustainability performance needs to be measurable (e.g. energy profiles should be queryable and updatable for every feature/component within the framework), including energy sources (e.g. renewable, non-renewable) and energy consumption profiles (e.g. estimated power consumption).	UC3
SUR0.2	Sustainability needs to be maximised to reduce environmental footprint by leveraging edge characteristics (e.g. by increasing the share of renewables, minimising battery use/size, using energy otherwise wasted, or scaling down active Runtimes).	UC3
SUR0.3	The whole energy lifecycle should be taken into account in order to implement a circular economy, including e.g. energy availability and cost and hardware degradation.	All

Table 2.2. Global Sustainability Requirements.

2.3. Interoperability Requirements

The EU's New Industrial Strategy also identifies the need to achieve a fairer and more competitive business environment between business users and cloud providers, enhancing access to fair, competitive, and trustworthy cloud solutions. Those objectives are also in line with the technological priorities being defined as part of the updated EU industry roadmap by the European Alliance for Industrial Data, Edge and Cloud. Interoperability is also the focus of the Project's Research Challenge 4—*"Portable and adaptive execution of serverless workloads across a multi-provider cloud-edge continuum"* stated in the Grant Agreement.

The interoperability requirements, reported in Table 2.3, target the ability of the COGNIT Framework to connect and integrate with existing infrastructures, services, hardware. This means that the framework should be aware of and use existing frameworks, technologies and standards, generic enough to be deployed on common infrastructures, and its usage instructions should be properly documented:

Id	Description	Source
IR0.1	Deployment of the COGNIT Framework and of its components should be as portable as possible across heterogeneous infrastructures or cloud/edge service providers (e.g. by using broadly-adopted virtualisation and container technologies).	All
IR0.2	Preference should be given to expanding existing frameworks, tools, and open standards.	All
IR0.3	The interfaces of the COGNIT Framework shall be documented in order to facilitate discovery of its features by third-parties.	All

Table 2.3. Global Interoperability Requirements.

2.4. Security Requirements

Security requirements (reported in Table 2.4) cover the ability of the COGNIT Framework to protect data and services at rest and in transit, in particular regarding confidentiality, integrity, and availability (CIA). In line with the recommendations defined by the Cybersecurity Research Directions for the EU's Digital Strategic Autonomy, they follow these three dimensions: protect the personal data from Internet giants, ensure resilience, and retain the ability to make informed and independent decisions. The COGNIT Framework should implement a security-by-design approach by adopting a number of good practices, including a risk-based approach based on the NIST CyberSecurity framework that structures security controls (identify, detect, protect, respond, recover), and a secure software lifecycle management based on DevSecOps (see Section 10).

In addition, special care has been taken to adapt these high level security requirements to make sure that they are aligned with the latest EU cybersecurity priorities, such as the NIS2 Directive and the CRA. Security is the focus of the Project's Research Challenge 7—*"Secure and trusted execution of computing environments on multi-provider cloud-edge continuum"* stated in the Grant Agreement; and has been considered through the CyberSecurity Use Case by incorporating advanced anomaly detection mechanisms, privacy respecting techniques, and security remediation orchestration:

Id	Description	Source
SER0.1	Communications inside COGNIT, and between the COGNIT environment and the outside (e.g. IoT devices) should be encrypted and signed using security mechanisms such as SSLv3.	All
SER0.2	The COGNIT Framework should be built following security-by-design and Zero Trust practices.	UC4
SER0.3	The implementation of the COGNIT Framework should be aligned with the latest legislative frameworks, such as the NIS2 Directive, the GDPR, and the future Cyber Resilience Act (CRA).	All
SER0.4	Runtimes should be protected against threats by the enforcement of security controls such as secure defaults, vulnerability scans, intrusion and anomaly detection and continuous security assessment (the specific controls to be implemented will be determined by a risk analysis).	All
SER0.5	Resources should be protected by an Identity and Access Management (IAM) system, implementing role based access control (RBAC), security zones, and support for a multi-tenant security model.	All
SER0.6	Integrity of the offloaded functions needs to be guaranteed, including the function inputs and outputs	All

Table 2.4. Global Security Requirements.

3. Use Case Requirements

This section summarises the user requirements from the Use Cases that has been addressed and partially or completely validated by COGNIT:

Id	Description	Source
UR0.1	Device applications should be able to offload any function written in C or Python languages.	All
UR0.2	Device applications should be able to upload data from the device ensuring data locality with respect to where the offloaded function is executed.	All
UR0.3	Device applications should be able to upload data from external backend storages ensuring data locality with respect to where the offloaded function is executed.	All
UR0.4	Execution of functions such as ML inference engines should be able to load machine learning models stored ensuring data locality with respect to where the function is executed.	All
UR0.5	Function execution can be executed in different tiers of the Cloud-Edge continuum according to network latency requirements.	All
UR0.6	Device application shall have the ability to define maximum execution time of the offloaded function upon offloading.	All
UR0.7	Device application shall have the ability to specify and enforce runtime maximum provisioning time and runtime shall be provisioned within the previously specified time.	All
UR0.8	Device applications must be able to request and obtain authorization prior to establishing any further interaction with COGNIT.	All
UR0.9	IAM system integration for high granularity authentication and user management for device clients and the COGNIT Framework.	All
UR0.10	Push mechanism to inform about status/events from the COGNIT Framework back to the requestor device client.	All

Table 3.1. Common user requirements.

Id	Description	Source
UR1.1	Function execution shall be supported in shared, multi-provider environments (with different access and authorization procedures), and the execution must be isolated from other processes on the host system.	UC1
UR1.2	Device applications shall be able to use scalable resources for offloading function execution to maximise exploitation of resources in shared environments, while avoiding saturation or resource kidnapping.	UC1
UR1.3	Function execution should exploit data locality and prioritise edge nodes where the required data is already stored.	UC1
UR1.4	The whole life cycle of either function execution or code offloading should be auditable and non repudiable.	UC1
UR1.5	Device applications should be able to request execution over GPUs.	UC1

Table 3.2. User requirements for UC1 (Smart Cities).

Id	Description	Source
UR2.1	It shall be possible to obtain both a-priori estimates of expected, and actual measurements of, energy consumption of the execution of functions.	UC2
UR2.2	COGNIT Framework should be able to adapt to events with sudden peaks of function requests, in which the offloaded functions require much heavier computations and more frequent execution than usual.	UC2
UR2.3	Possibility for devices to have access to GPUs, when available, during high-alert mode.	UC2

Table 3.3. User requirements for UC2 (Wildfire Detection).

Id	Description	Source
UR3.1	Device Client and user applications shall share a maximum of 500 kB of available RAM in total.	UC3
UR3.2	It shall be possible for the user application to dynamically use scalable resources for function execution due to changes in the application demand.	UC3
UR3.3	The COGNIT Framework shall have support for the C programming language.	UC3

Table 3.4. User requirements for UC3 (Energy).

Id	Description	Source
UR4.1	The COGNIT Framework should have the ability to dynamically set the permissible edge nodes for executing the function based on policy (e.g. geographic security zones, distance to edge node).	UC4
UR4.2	The COGNIT Framework should have the ability to make data available between different (edge) clusters transparently to the user application.	UC4
UR4.3	The Device should be able to request the execution of a function as close as possible to the Device's location, satisfying latency requirements.	UC4

Table 3.5. User requirements for UC4 (Cybersecurity).

PART II. Architecture Definition

4. COGNIT Framework

The COGNIT Framework provides an AI-Enabled Adaptive Serverless model specifically designed for the cloud-edge continuum. Unlike traditional serverless/FaaS platforms that target centralized cloud datacenters with event-driven, short-lived functions, COGNIT addresses the needs of edge devices that require computationally intensive data processing with sustained execution, low latency, and energy efficiency across highly distributed and heterogeneous infrastructure.

4.1. COGNIT Application Profile

COGNIT addresses emerging mobile and IoT edge device applications that need to augment their capabilities for computationally intensive data processing. These applications require access to fast computational units and low-latency connections. Code offloading enables computationally intensive processing to be executed outside the resource-constrained edge devices, translating to more efficient power management and better application performance.

Traditional code offloading faces challenges pertaining to practical usage, including complexity of integration with cloud management environments, dynamic system configuration, scalability. COGNIT addresses these limitations by developing a distributed Serverless model that, integrated with a Cognitive cloud-edge management platform, facilitates the development of elastic and scalable edge-based applications.

The COGNIT Framework enables application developers to easily integrate cloud-edge processing in their coding logic, making it feasible to offload data processing code fragments and tasks from end-user systems, devices, sensors, or actuators to the cognitive continuum to speed up computation, save energy, save bandwidth, or provide low latency.

Table 4.1 summarises the main differences between the COGNIT model and the traditional Serverless/FaaS model implemented by existing cloud offerings and technologies:

	Serverless/FaaS Cloud Model	**COGNIT Serverless Cloud-Edge Model**
PROGRAMMING		
Programming model	Interconnected functions (code) defined at a Cloud provider	Single program source code on device
Where the function runs	On Cloud provider	On cloud-edge location
When the function runs	On event, cloud event-driven	On demand, application logic-driven
Application profile	Low footprint	Compute-intensive data processing
Program state	Stateless	Stateless

Maximum runtime	Short (e.g. <900 seconds)	None
Maximum capacity	Limited (e.g. < 3GiB memory)	None
Deployment requirements	Basic capacity (e.g. memory) & quotas	Performance, Latency, Hardware Features (TEE)
OPERATION		
Scaling	Cloud provider responsible	Cloud provider responsible
Deployment	Cloud provider responsible	Cloud provider responsible
Fault Tolerance	Cloud provider responsible	Cloud provider responsible
INFRASTRUCTURE		
Infrastructure	Single centralised cloud	Dynamic distributed cloud/edge
Location	Single centralised cloud	Developer selects (cloud-edge continuum)
Special-purpose devices	None	Hardware (TEE)

Table 4.1. COGNIT Serverless Cloud-Edge Model vs traditional Serverless/FaaS Cloud Model.

Key differentiators of the COGNIT model include:

- **Application-logic driven execution** rather than event-driven, allowing devices to maintain control over function execution.
- **No artificial limits** on runtime duration or resource capacity, supporting compute-intensive workloads.
- **Multi-provider distributed infrastructure** spanning cloud and edge locations rather than single centralized cloud.
- **Location selection** based on application requirements (performance, latency, energy consumption).
- **Special-purpose resource** support including Trusted Execution Environments.
- **Data as a Service** providing storage for applications to manage their own data.

4.2. COGNIT Execution Model

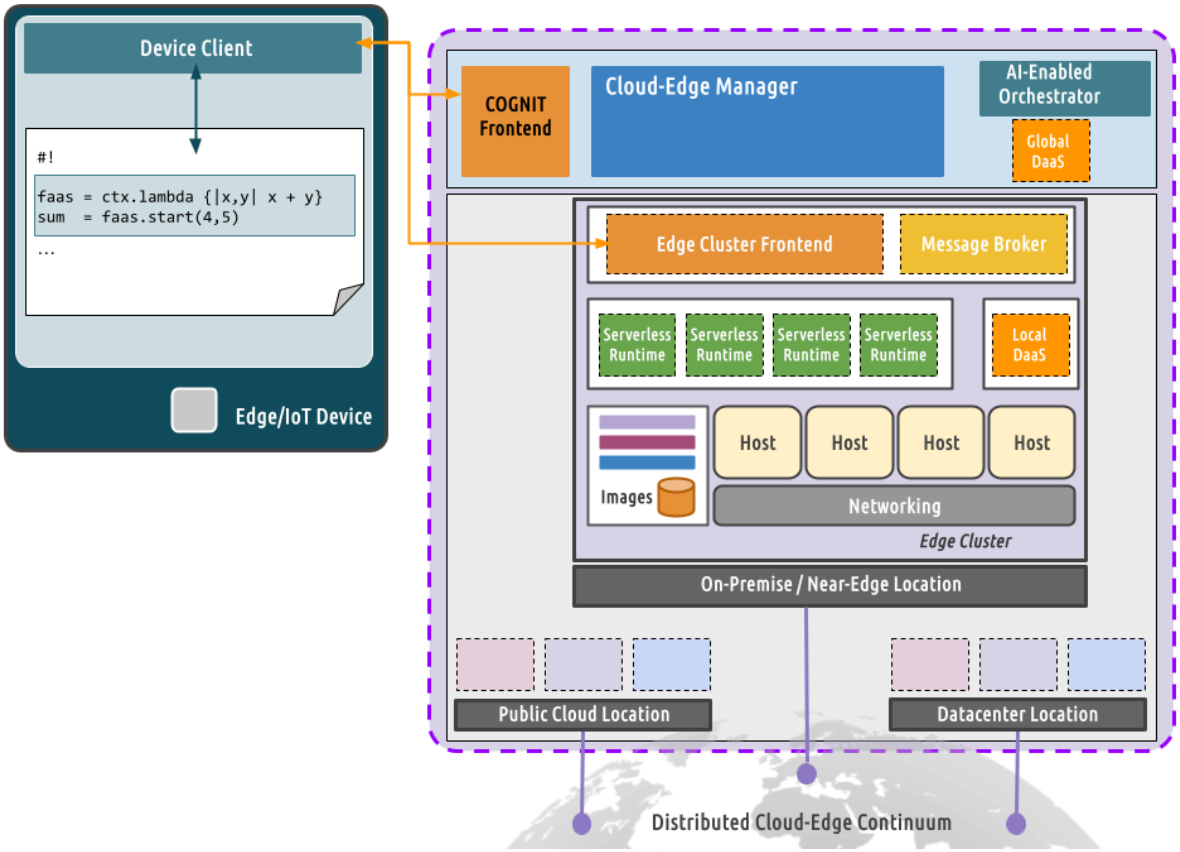

Figure 4.1. General view of the COGNIT Architecture.

The first group of components (described in detail in Section 5) provides the distributed serverless model for edge application development:

- **Device Client.** A lightweight SDK that allows devices to offload function executions to the cloud-edge continuum. It handles authentication, application requirements management, function and data upload, and function execution requests.
- **COGNIT Frontend**. The main entry point for the COGNIT service that authenticates device clients, manages application requirements and functions, and provides Edge Cluster assignments to devices by reading from a shared assignment database that the AI-Enabled Orchestrator updates asynchronously.
- **Edge Cluster.** A semi-autonomous environment with dedicated software-defined resources (compute, network, storage) that hosts Serverless Runtimes for executing offloaded functions. Edge Clusters provide an abstraction layer enabling interoperability across multi-provider infrastructure.

The second group of components (described in detail in Section 6) provides smart and adaptive management of cloud-edge resources:

- **Cloud-Edge Manager.** The infrastructure management component responsible for managing distributed resources, Edge Cluster lifecycle, Serverless Runtime lifecycle, monitoring and metrics collection, and authentication/authorization.
- **AI-Enabled Orchestrator.** A smart component that uses AI/ML models and optimization algorithms to manage and optimize Edge Clusters and Serverless Runtimes according to application requirements, infrastructure usage, and energy policies.

The COGNIT Framework provides **Data as a Service (DaaS)** to support application data management. The framework does not provide stateful function execution or automatic state management—applications are responsible for managing their own data and state. The framework provides two DaaS options:

- **Global DaaS**: A global DaaS service accessible from all Edge Clusters, deployed with the COGNIT Frontend. Devices can upload data to this Global DaaS, making it available to functions regardless of which Edge Cluster executes them. This Global DaaS ensures data persistence even when devices are reassigned to different Edge Clusters.
- **Local DaaS**: Each Edge Cluster includes a Local DaaS instance for cluster-specific data storage. Devices can upload data directly to an Edge Cluster's Local DaaS when data locality is important for performance. Data stored in Local DaaS is only accessible within that specific Edge Cluster. If a device is reassigned to a different Edge Cluster, data stored in the previous Edge Cluster's Local DaaS will not be accessible. Applications using Local DaaS should either upload data just before function execution or use Global DaaS for data that needs to persist across Edge Cluster reassignments.

However, the framework does not provide automatic state management for functions—applications must explicitly manage data upload, storage, and retrieval themselves.

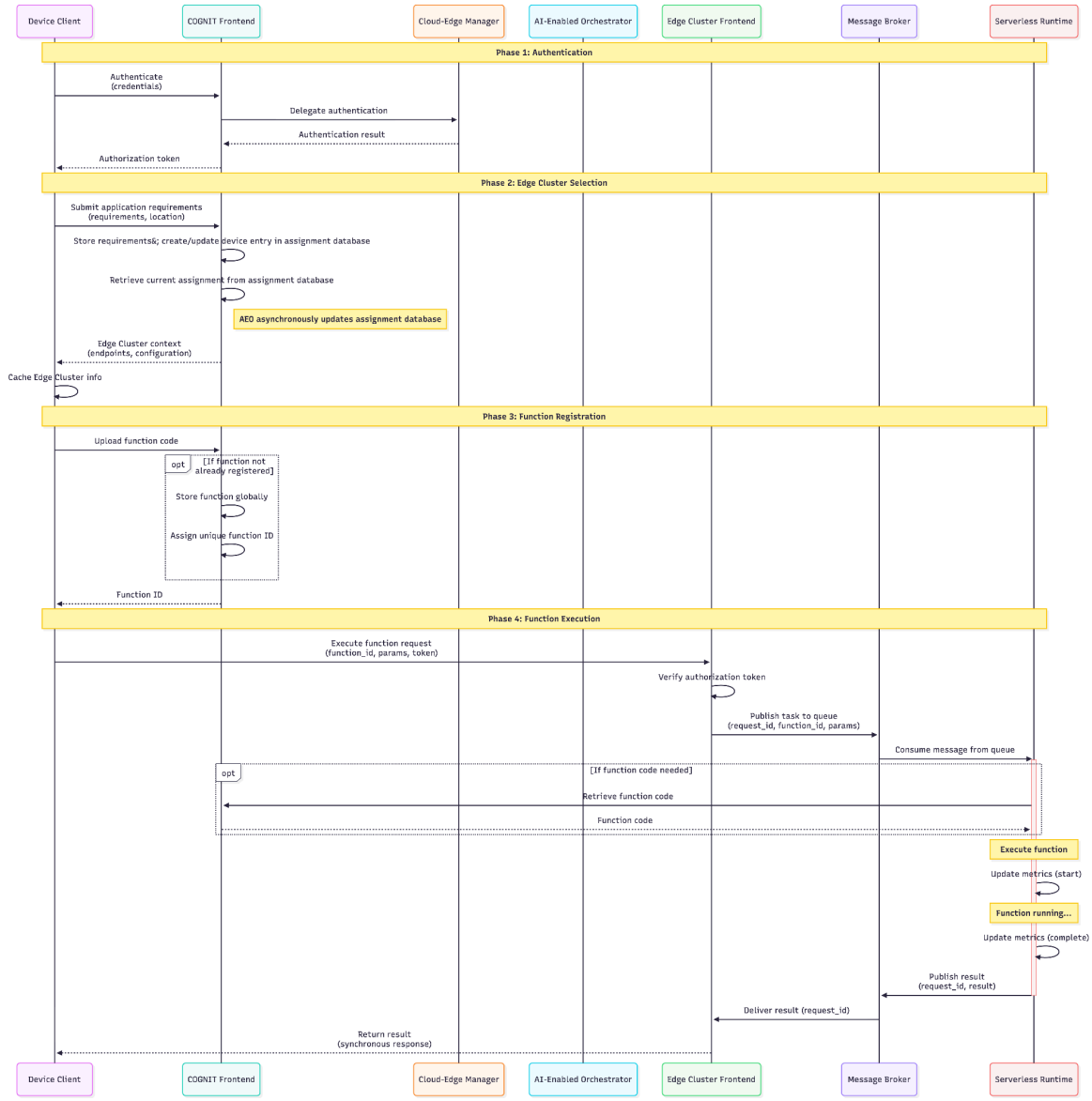

Figure 4.2. COGNIT Execution Model sequence diagram.

The execution flow involves the following phases:

1. **Authentication**
    - Device authenticates with COGNIT Frontend using username/password over a secure channel.
    - COGNIT Frontend delegates authentication to Cloud-Edge Manager.
    - Upon success, COGNIT Frontend returns an authorization token to the device.
2. **Edge Cluster Selection**
    - Device sends application requirements to COGNIT Frontend.

    - COGNIT Frontend stores requirements and creates or updates the device entry in the shared assignment database (accessible to the AI-Enabled Orchestrator)
    - AI-Enabled Orchestrator runs asynchronously, monitors the assignment database and infrastructure state, and updates device-to-cluster assignments in the database
    - When the device requests its Edge Cluster assignment, COGNIT Frontend retrieves the current assignment from the database and returns Edge Cluster context to the device
    - Device caches Edge Cluster information for subsequent function executions.
3. **Function Registration**
    - Device uploads function code to COGNIT Frontend.
    - Function is stored globally and assigned an unique identifier.
    - Function becomes available for execution in any Edge Cluster.
4. **Function Execution**
    - Device sends synchronous function execution request to Edge Cluster Frontend with function ID and input parameters.
    - Edge Cluster Frontend internally uses message queues to dispatch requests to available Serverless Runtime.
    - Serverless Runtime executes function and returns result via internal response queue.
    - Edge Cluster Frontend returns result to device, completing the synchronous request-response cycle.

This execution model provides several key benefits:

- **Separation of concern**s: Authentication, orchestration, and execution are handled by specialized components.
- **Smart placemen**t: AI-Enabled Orchestrator ensures functions execution on optimal resources.
- **Scalability**: Edge Clusters and Serverless Runtimes can scale independently with queue-based decoupling.
- **Flexibility**: Devices can be dynamically reassigned to different Edge Clusters.
- **Efficiency:** Functions are registered once and can execute in any Edge Cluster.
- **Security**: Token-based authentication with fine-grained authorization control.
- **Reliability**: Queue-based architecture provides fault tolerance and automatic retry capabilities.
- **Load distribution:** Serverless Runtimes pull work at their own pace, preventing overload.

The architecture supports both reactive adaptations (responding to changes in application requirements or infrastructure conditions) and proactive optimizations (anticipating needs through AI/ML predictions) as detailed in Section 6.

5. Distributed Serverless Model for Edge Application Development

In this Section we detail the model for an application running on the Cloud-Edge continuum using a distributed Serverless model. The main components are: the Device Client, the COGNIT Frontend and Edge Cluster Frontend.

5.1. Device Client

The Device Client is a lightweight Software Development Kit (SDK) that enables edge devices and applications to interact with the COGNIT Framework for offloading function executions to the cloud-edge continuum.

5.1.1 Device Client Architecture

The Device Client serves as the interface between edge applications and the COGNIT Framework, abstracting the complexities of distributed serverless execution. Its architecture is organized into distinct layers that separate concerns and provide clean abstractions, enabling a simple and intuitive programming interface that allows application developers to offload computationally intensive functions without managing the underlying cloud-edge infrastructure.

The transport layer handles all low-level communication details, including secure HTTPS connections with TLS encryption, JSON encoding and decoding of payloads, and management of authorization tokens. The application layer provides the high-level programming interface that application developers interact with directly, exposing intuitive operations for authentication, function upload, requirements specification, data management, and function execution. The application layer also manages caching and state, storing information such as Edge Cluster assignments and function identifiers to optimize performance and reduce redundant operations.

The Device Client manages the complete lifecycle of application requirements within the COGNIT Framework. When an application specifies its operational constraints in terms of high-level requirements such as maximum acceptable latency, required response time, need for confidential computing environments, or restrictions on which infrastructure providers may be used, the Device Client communicates these requirements to the COGNIT Frontend. This enables the AI-Enabled Orchestrator to make smart placement decisions that satisfy the application's needs while optimizing for system-wide objectives such as energy efficiency and resource utilization.

Function management enables applications to upload their function code to the COGNIT Framework, where it becomes globally available across all Edge Clusters. The Device Client handles function registration, maintains function identifiers, and ensures that function code is properly uploaded.

Data placement and locality management provides mechanisms for uploading data to either Global DaaS (COGNIT DaaS, accessible from any Edge Cluster) or Local DaaS (Edge

Cluster DaaS, specific to a particular Edge Cluster), allowing applications to optimize data placement based on their access patterns.

Function execution coordination operates through multiple steps, as shown in Figure 4.2. When an application needs to execute a function, the Device Client first requests an Edge Cluster assignment from the COGNIT Frontend based on the application's requirements. The COGNIT Frontend retrieves the device's current Edge Cluster assignment from the shared assignment database (which the AI-Enabled Orchestrator updates asynchronously) and returns it to the Device Client. The Device Client caches this Edge Cluster context to avoid repeated assignment requests for subsequent executions. It then constructs and sends the function execution request to the assigned Edge Cluster, including the function identifier and any input parameters. The Device Client supports synchronous execution patterns, where it waits for the result before proceeding.

The Device Client incorporates an asynchronous background thread that enables dynamic Edge Cluster reassignment in response to changing infrastructure conditions. This background thread periodically communicates with the COGNIT Frontend to request updated Edge Cluster assignments, allowing the COGNIT Framework to optimize workload distribution as conditions evolve. When infrastructure characteristics change, such as shifts in resource availability, network latency variations, energy consumption patterns, the AI-Enabled Orchestrator may determine that a different Edge Cluster would better serve the device's requirements. Through the asynchronous reassignment mechanism, devices transparently migrate to more suitable Edge Clusters without application-level intervention. The frequency of these reassignment checks is user-configurable, allowing applications to balance the benefits of optimal placement against the overhead of periodic reassignment queries. This capability is essential for maintaining workload optimization across the dynamic cloud-edge continuum, ensuring that devices continuously benefit from the most appropriate infrastructure resources as the system evolves.

5.2. COGNIT Frontend

The COGNIT Frontend serves as the main entry point and coordination hub for the COGNIT service, providing a unified interface through which devices interact with the distributed cloud-edge infrastructure. As the central gateway, it orchestrates authentication, manages application requirements and functions, coordinates with the AI-Enabled Orchestrator for smart resource selection, and maintains the global state necessary for seamless operation across the heterogeneous cloud-edge continuum.

5.2.1 COGNIT Frontend Architecture

The COGNIT Frontend provides external interfaces through which devices interact with the COGNIT Framework, exposing well-defined endpoints for authentication, requirements management, function registration, data upload, and Edge Cluster assignment queries. It serves as the main entry point and coordination hub, orchestrating authentication, managing application requirements and functions, coordinating with the AI-Enabled Orchestrator for smart resource selection, and maintaining the global state necessary for seamless operation across the heterogeneous cloud-edge continuum.

5.2.1.1 Authentication & Authorization

The COGNIT Frontend authenticates devices and establishes secure communication channels for all subsequent interactions. When a device first connects to the COGNIT Framework, it validates credentials through integration with the Cloud-Edge Manager's identity and access management system. Upon successful authentication, it generates and issues authorization tokens that encapsulate the device's identity, authorized operations, and validity constraints. These tokens employ a capability-based security model where specific permissions are cryptographically bound to the token itself, enabling distributed authorization decisions at Edge Clusters without requiring constant communication with the COGNIT Frontend. The COGNIT Frontend manages the complete token lifecycle, tracking issued tokens and their expiration states, coordinating token refresh operations, and handling revocation when security requirements demand immediate access termination.

5.2.1.2 Application Requirements Management

When devices upload their application requirements—specifying constraints such as maximum latency, maximum response time, confidential computing needs, and allowed infrastructure providers—the COGNIT Frontend validates these requirements, ensures they are well-formed and consistent, and stores them in persistent storage where they remain accessible to the AI-Enabled Orchestrator for placement optimization. The COGNIT Frontend maintains these requirements throughout the application lifecycle, allowing devices to update them as operational conditions evolve. Beyond the explicit requirements, it tracks device metadata including geographic location, capabilities, latency measurements, and historical execution and workload patterns, enriching the context available for smart placement decisions.

5.2.1.3 Function Registry

The COGNIT Frontend maintains a global function registry that enables the distributed serverless execution model. When devices upload function code, it assigns unique identifiers, performs validation checks to ensure code integrity and dependency compatibility, and stores both the code and associated metadata in object storage accessible across all Edge Clusters. This global function registry ensures that once a function is uploaded, it can be executed at any Edge Cluster without requiring per-cluster function deployment. This centralized function management simplifies the developer experience while enabling the distributed execution model that characterizes the COGNIT Framework.

5.2.1.4 Global DaaS

A global DaaS service accessible from all Edge Clusters, deployed with the COGNIT Frontend. Devices can upload data to this Global DaaS, making it available to functions regardless of which Edge Cluster executes them. This Global DaaS ensures data persistence even when devices are reassigned to different Edge Clusters. The COGNIT

Frontend provides mechanisms for devices to upload data to Global DaaS and supports data transfer from external storage sources.

5.2.1.5 Edge Cluster Assignment Coordination

Edge Cluster assignment coordination operates through a database-mediated architecture that decouples device assignment queries from the AI-Enabled Orchestrator's optimization process. When a device uploads its application requirements to the COGNIT Frontend, it checks whether the device identifier already exists in the assignment database. If the device is new to the system, it creates a new database entry that captures the device's identifier, requirements, location, and other relevant metadata. This database serves as the authoritative source for device-to-Edge-Cluster assignments, enabling the COGNIT Frontend to respond immediately to assignment requests by consulting the database rather than synchronously querying the AI-Enabled Orchestrator. When a device requests its Edge Cluster assignment, the COGNIT Frontend retrieves the current assignment from the database and returns complete Edge Cluster context including connection endpoints and configuration parameters, enabling immediate function execution without waiting for optimization computations.

5.3. Edge Cluster

An Edge Cluster represents a semi-autonomous execution environment with dedicated software-defined resources that provides the infrastructure for running offloaded functions from devices. Edge Clusters can be instantiated anywhere across the cloud-edge continuum, from centralized public cloud data centers to private cloud installations, on-premises facilities, or edge locations positioned close to data sources and end users. This deployment flexibility is essential for the COGNIT Framework’s ability to adapt to diverse latency, privacy, and sovereignty requirements.

5.3.1. Edge Cluster Architecture

An Edge Cluster is composed of several integrated components that work together to provide a complete execution environment for offloaded functions. The architecture consists of software-defined infrastructure resources (compute, network, and storage), the Edge Cluster Frontend that interfaces with devices, the Message Broker that manages asynchronous communication, the Local DaaS that provides cluster-specific data storage, and Serverless Runtimes that execute functions. These components are designed to work together seamlessly, providing scalability, fault tolerance, and efficient resource utilization while maintaining strong isolation and security properties.

5.3.1.1 Edge Cluster Resources

Each Edge Cluster is constructed from three fundamental layers of software-defined infrastructure: compute, network, and storage. The software-defined compute layer provides the execution substrate for Serverless Runtimes using virtual machine isolation technology. This approach provides strong isolation, comprehensive security guarantees, and support for specialized hardware requirements including GPUs and trusted execution

environments. The compute layer manages resource allocation and isolation at the host level, ensuring that co-located Serverless Runtimes do not interfere with each other's execution. The compute layer can expose specialized hardware to Serverless Runtimes, including GPUs for acceleration of compute-intensive workloads and trusted execution environments for confidential computing that provide hardware-based cryptographic protection of data and code during execution.

The software-defined network layer creates the communication fabric within an Edge Cluster through two types of networks. Private networks enable internal communication among Serverless Runtimes, the Edge Cluster Frontend, and the local object storage service. A public network provides external connectivity, allowing devices to send function execution requests to the Edge Cluster Frontend.

Software-defined storage within an Edge Cluster serves three primary purposes. First, it caches Serverless Runtime images locally that serve as backing files for virtual machine instantiation. When a new Serverless Runtime is deployed, the storage system uses copy-on-write cloning from these cached images, storing only the differential changes specific to each VM instance while sharing the common base image. This approach significantly improves instantiation times by avoiding remote image retrieval and minimizes storage consumption through image sharing. Second, it stores the VM instances themselves as they execute, maintaining only the copy-on-write differences from the backing files. Third, it provides data blocks for the Edge Cluster's local object storage service, enabling devices to upload data directly to the cluster for improved data locality and performance.

5.3.1.2 Edge Cluster Frontend

The Edge Cluster Frontend serves as the interface between devices and the Serverless Runtimes executing within an Edge Cluster. It implements a queue-based architecture that decouples the submission of function execution requests from their actual execution, providing scalability, reliability, and efficient load distribution properties essential for managing dynamic workloads across heterogeneous resources.

When the Edge Cluster Frontend receives a function execution request from a device, it first validates the authorization token included with the request, ensuring that the device is authorized to execute the specified function and that the token remains within its validity period. After successful authorization, the Edge Cluster Frontend enqueues the execution request in a message queue managed by the Message Broker and maintains the connection with the device. From the device's perspective, this is a synchronous request-response interaction where it sends a function execution request and waits for the result. Internally, however, the Edge Cluster Frontend uses a queue-based architecture to manage the distribution of work to Serverless Runtimes, providing scalability and load distribution while presenting a simple synchronous interface to devices.

When a Serverless Runtime completes function execution, it publishes the result to a response queue along with an identifier that allows the Edge Cluster Frontend to match results with their originating requests. The Edge Cluster Frontend retrieves the result from

the response queue and immediately returns it to the waiting device, completing the synchronous request-response cycle from the device's perspective.

This queue-based architecture provides several important properties while maintaining a simple synchronous interface for devices. Fault tolerance emerges naturally, as execution requests remain in queues until successfully processed; if a Serverless Runtime fails while processing a request, the message returns to the queue for consumption by another Runtime, all transparently to the waiting device. Scalability is enhanced because Serverless Runtimes can be dynamically added or removed without coordination, i.e. new Serverless Runtimes automatically begin consuming from queues, while removed Serverless Runtimes simply stop consuming. The queue depth provides a natural back-pressure signal indicating system load, enabling adaptive capacity management. Failed executions can be automatically requeued improving reliability without requiring explicit retry logic in device code.

Latency measurement capabilities allow devices to continuously monitor network conditions to each Edge Cluster. The Edge Cluster Frontend provides a lightweight endpoint that devices can ping to measure round-trip latency, collecting and aggregating these measurements for export to the monitoring system. This latency data feeds into the AI-Enabled Orchestrator's placement decisions, ensuring that devices are assigned to Edge Clusters that can meet their latency requirements.

The Message Broker provides the asynchronous communication infrastructure that enables the queue-based architecture within an Edge Cluster. It manages execution request queues and response queues, decoupling the Edge Cluster Frontend from Serverless Runtimes and enabling scalable, fault-tolerant function execution.

5.3.1.3 Message Broker

The Message Broker implements a pull-based model where Serverless Runtimes actively consume execution requests from queues at their own pace. This design provides automatic load distribution, as each Serverless Runtime pulls work only when it has capacity to process it. The broker ensures message persistence, guaranteeing that execution requests are not lost even if Serverless Runtimes or the Edge Cluster Frontend experience temporary failures.

The Message Broker maintains separate queues for execution requests and responses, enabling the asynchronous processing model while presenting a synchronous interface to devices through the Edge Cluster Frontend. Execution request queues hold pending function execution requests until they are consumed by available Serverless Runtimes. Response queues store execution results until they are retrieved by the Edge Cluster Frontend and returned to waiting devices.

Queue management capabilities include message acknowledgment to ensure reliable processing, message expiration to prevent stale requests from accumulating, and dead-letter queues for messages that cannot be processed after multiple attempts. The broker provides visibility into queue depth and message flow, enabling monitoring and capacity planning based on workload patterns.

5.3.1.4 Local DaaS

Local DaaS (Edge Cluster DaaS) provides cluster-specific data storage within each Edge Cluster, enabling devices to upload data directly to an Edge Cluster when data locality is important for performance. The Local DaaS is deployed as part of the Edge Cluster infrastructure and is accessible only within that specific Edge Cluster's boundaries.

Devices can upload data directly to an Edge Cluster's Local DaaS when they know execution will occur there and want to optimize data transfer and latency. The Edge Cluster Frontend provides access to Local DaaS, allowing devices to upload data before function execution. Serverless Runtimes executing within the Edge Cluster can retrieve data from Local DaaS during function execution, providing low-latency data access for improved performance.

Data stored in Local DaaS remains visible only within the Edge Cluster's boundaries and is not accessible if the device is reassigned to a different Edge Cluster.

5.3.1.5 Serverless Runtime

The Serverless Runtime represents the fundamental execution unit within an Edge Cluster where offloaded functions actually run. Each Serverless Runtime is deployed as a virtual machine, providing strong isolation, comprehensive security properties, and support for specialized hardware including trusted execution environments for confidential computing.

The core responsibility of a Serverless Runtime is function execution. When the Runtime consumes an execution request from the message queue, it loads the specified function code from the function registry, processes the input parameters, executes the function within a controlled environment, and serializes the result for publication to the response queue. The function executor provides the language runtime environment (i.e. Python) along with common libraries and frameworks that functions may depend upon.

Data access capabilities enable functions executing within a Serverless Runtime to retrieve necessary input data and store output data. The Serverless Runtime provides access to both Global DaaS (COGNIT DaaS, accessible from all Edge Clusters) and Local DaaS (accessible only within the specific Edge Cluster). The COGNIT Framework does not provide stateful function execution or automatic state management. Applications must explicitly manage data upload, storage, and retrieval. Functions are stateless by design—if an application requires stateful behavior, it must manage state through the provided DaaS services. Data stored in Local DaaS will not be accessible if the device is reassigned to a different Edge Cluster; only Global DaaS provides persistence across Edge Cluster reassignments.

Security isolation is paramount in the Serverless Runtime's design, as it executes potentially untrusted function code from multiple devices within shared infrastructure. The Serverless Runtime employs multiple isolation mechanisms including process-level isolation, enforcement of resource limits on CPU, memory, and I/O consumption, network isolation to prevent unauthorized communication. For applications requiring the strongest security guarantees, Serverless Runtimes can optionally be deployed with confidential

computing support using hardware-based trusted execution environments, providing cryptographic protection of data and code during execution.

Integration with the Edge Cluster's queue infrastructure is managed through a message queue consumer component that connects to the Message Broker, subscribes to execution request queues, and pulls requests when the Serverless Runtime has capacity to process them. After successfully executing a function, the Serverless Runtime acknowledges the message to remove it from the queue and publishes the execution result to a response queue. This pull-based model ensures that Serverless Runtimes are never overloaded, as they only consume work they have capacity to handle.

Comprehensive metrics collection within each Serverless Runtime enables monitoring, performance analysis, and smart resource management. The Serverless Runtime tracks framework-level metrics including function execution times, function request rates, resource usage including CPU, memory, and I/O consumption, and measures energy consumption through integration with power measurement tools. These metrics are exported to the centralized monitoring infrastructure, where they inform real-time operational decisions.

6. Cognitive Cloud-Edge Module

The Cognitive Cloud-Edge Module addresses the Project's Research Challenges 4 and 5:

- **Research Challenge 4:** "*Portable and adaptive execution of serverless workloads across a multi-provider cloud-edge continuum*".
- **Research Challenge 5**: "*Automatic and smart adaptation of the cloud-edge continuum to the changing demands of the applications*".

The Cognitive Cloud-Edge Module provides smart and adaptive management of cloud-edge resources. It consists of two main components: the **Cloud-Edge Manager** and the **AI-Enabled Orchestrator**. Together, these components enable the COGNIT Framework to automatically and smartly adapt to changing application requirements, infrastructure conditions, and energy policies.

The Cognitive Cloud-Edge Module follows a **MAPE-K** (Monitor, Analyze, Plan, Execute, Knowledge) autonomic computing architecture, which enables COGNIT to autonomously manage and optimize resources across the cloud-edge continuum. The MAPE-K architecture is foundational to the COGNIT design, enabling a proactive and automated approach to infrastructure management. COGNIT implements a MAPE-K loop where:

- **Monitor**: Monitoring is performed by the Cloud-Edge Manager, which collects metrics from hosts, VMs, and applications. These metrics include infrastructure health, resource utilization, energy consumption, and application performance metrics.
- **Analyze**: The AI-Enabled Orchestrator performs Analysis via ML models that predict workloads and identify optimization opportunities. Historical metrics are processed through ML models to generate predictions for Serverless Runtime scaling requirements and workload forecasting.
- **Plan**: The AI-Enabled Orchestrator performs Planning via optimization algorithms that generate optimal resource allocations. The Multi-Cluster Optimizer optimizes device-to-cluster assignments across the entire cloud-edge continuum and can request dynamic scaling of Edge Cluster hosts when the optimal assignment still exhibits contention or when load increases; the Cluster Optimizer optimizes Serverless Runtime placement within individual Edge Clusters.
- **Execute**: The Cloud-Edge Manager performs Execution by translating optimization plans into infrastructure actions to deploy, migrate, or power off Serverless Runtime VMs. Plans are executed sequentially, with dependency handling and error recovery.
- **Knowledge**: The Knowledge base is distributed across multiple storage systems that store time-series metrics collected by monitoring probes, infrastructure state and monitoring data, device-to-cluster assignments, application requirements, and estimated loads. This knowledge is continuously updated as metrics are collected, plans are executed, and optimization results are fed back to refine ML models and improve future decisions.

The module supports both reactive and proactive adaptability

- **Reactive Adaptability**:
  - Responds to changes in application requirements (updated by devices)
  - Reacts to infrastructure events (host failures, resource exhaustion)
  - Handles sudden workload changes (spikes in function requests)
- **Proactive Adaptability**:
  - Anticipates future resource needs using AI/ML predictions
  - Optimizes energy consumption based on historical patterns
  - Pre-provisions resources to minimize cold-start delays

The following table describes in more detail how the COGNIT Framework has dealt with the adaptability required for the automatic placement and mobility of the Serverless Runtimes and for leveraging in a smart way the elasticity capabilities that the infrastructure across the cloud-edge continuum provides:

Feature	Description	Activation
Serverless Runtime Horizontal Elasticity	Scale up/down the number of Serverless Runtimes to execute device workloads.	✔ **Reactive**: due to event-driven elasticity rules or changes in application requirements ✔ **Proactive**: due to AI-driven forecasting techniques.
Serverless Runtime Mobility	Live migrate the Serverless Runtime within the same Edge Cluster.	✔ **Reactive**: due to event-driven mobility rules or changes in edge application requirements ✔ **Proactive**: due to AI-driven forecasting techniques.
Optimal Global Workload Placement	Optimise the placement of existing Serverless Runtimes.	✔ **Proactive**: due to AI-driven optimisation techniques (e.g. reduce overall energy consumption and/or carbon emission footprint).
Infrastructure Cluster Elasticity	Scale up/down existing Edge Clusters by activating/deactivating hosts.	✔ **Reactive**: due to event-driven elasticity rules or changes in edge application requirements ✔ **Proactive**: due to AI-driven forecasting techniques.

Table 6.1. COGNIT reactive and proactive adaptability features.

In order to respond to those adaptability requirements, the Cognitive Cloud-Edge Module will be based on two main components, which will be presented in detail in the following sections, the **Cloud-Edge Manager** and the **AI-Enabled Orchestrator**.

6.1. Cloud-Edge Manager

The Cloud-Edge Manager represents the infrastructure management foundation of the COGNIT Framework, responsible for orchestrating the highly distributed and heterogeneous resources that span the cloud-edge continuum. It provides a unified management abstraction over infrastructure from multiple public cloud providers, private clouds, on-premises data centers, and edge locations, enabling the COGNIT Framework to treat this heterogeneous infrastructure as a coherent resource pool.

6.1.1. Cloud–Edge Manager Architecture

The Cloud-Edge Manager, as shown in Figure 6.2, provides comprehensive infrastructure management capabilities organized around several key components:

- The **Provider Catalogue** maintains information about all available infrastructure providers and their characteristics, enabling creation of Edge Clusters across the cloud-edge continuum.
- **Edge Cluster Provisioning** automates the deployment of complete Edge Cluster environments across heterogeneous providers.
- **Serverless Runtime Provisioning** manages the complete lifecycle of Serverless Runtimes within Edge Clusters, from deployment through scaling and migration.
- **Metrics, Monitoring, and Auditing** provides comprehensive observability across the distributed infrastructure.
- **Authentication and Authorization** ensures secure device access through a delegated authentication model.
- The **Plan Executor** translates optimization plans from the AI-Enabled Orchestrator into concrete infrastructure actions.

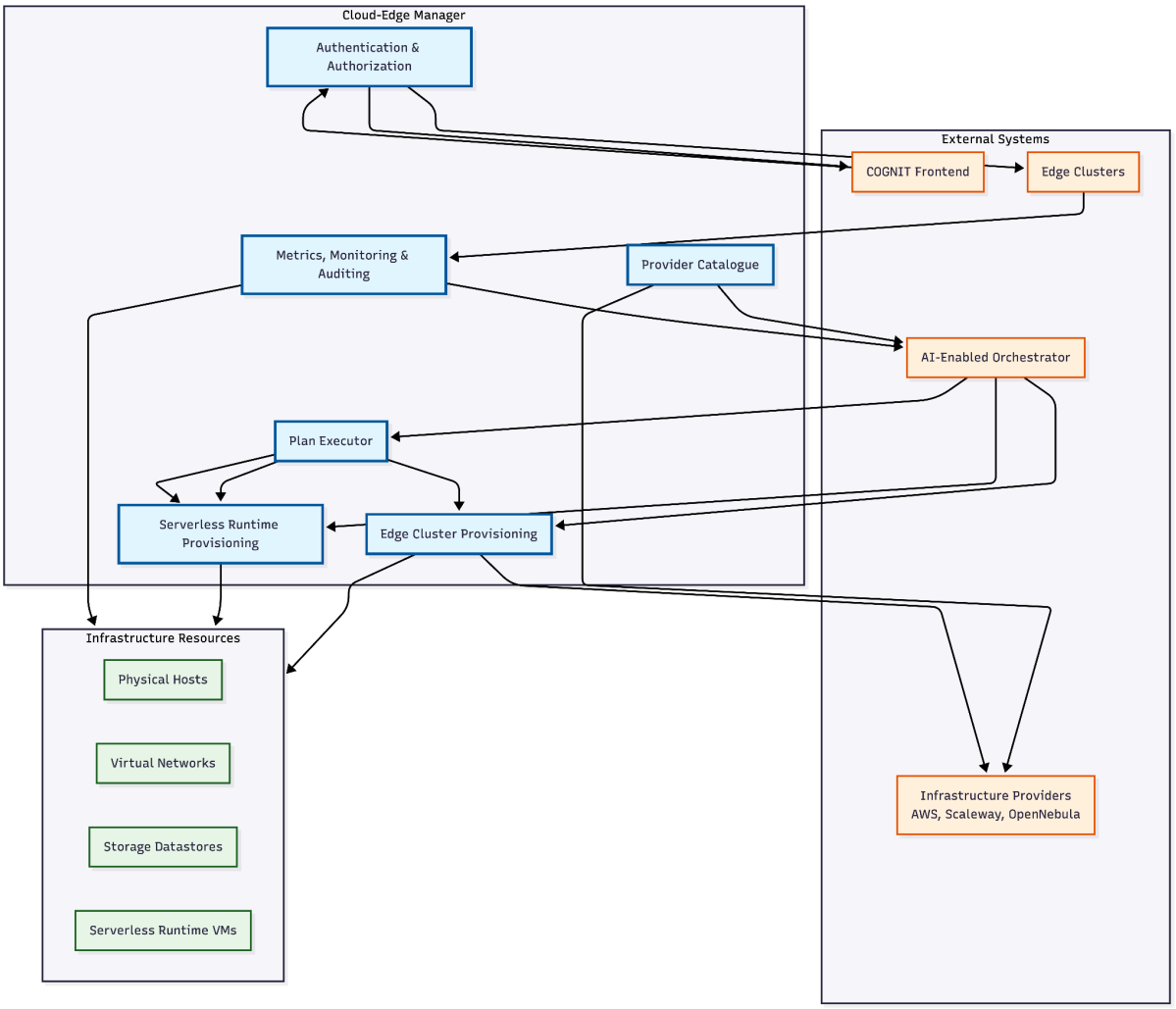

Figure 6.2. Cloud-Edge Manager Architecture.

Together, these components enable the Cloud-Edge Manager to maintain a unified view of heterogeneous infrastructure, execute lifecycle operations for Edge Clusters and Serverless Runtimes, and provide the monitoring and security foundation necessary for secure, efficient operation across the cloud-edge continuum.

6.1.1.1 Provider Catalogue

The Provider Catalogue maintains comprehensive information about all cloud and edge infrastructure providers available to the COGNIT Framework, serving as the knowledge base that enables provisioning of Edge Clusters across the heterogeneous cloud-edge continuum.

For each infrastructure provider integrated with COGNIT, the catalogue maintains a unique identifier, descriptive name, provider type classification (public cloud, or on-premises), and the credentials and endpoints necessary for programmatic access. This provider-level information enables the COGNIT Framework to maintain relationships with multiple

infrastructure sources simultaneously, supporting both multi-cloud strategies and hybrid deployments that span public and private infrastructure.

Provider drivers within the catalogue abstract the heterogeneity of provider-specific APIs behind a unified interface. Each provider driver implements the operations necessary for resource provisioning and scaling using that provider's native API, while exposing a consistent interface to the Cloud-Edge Manager. This abstraction allows the COGNIT Framework to treat all providers uniformly at the management layer while accommodating their implementation differences at the driver layer.

6.1.1.2. Edge Cluster Provisioning

Edge Cluster Provisioning capabilities automate the complete deployment of Edge Cluster environments across heterogeneous infrastructure providers, transforming high-level provisioning requests into fully operational Edge Clusters ready to execute functions. This automated provisioning process eliminates the manual complexity of coordinating resource allocation, network configuration, service deployment, and integration across diverse provider APIs.

The provisioning process begins when an administrator submits a provisioning request that specifies the target provider required compute capacity in terms of host count and types, network configuration including public and private network requirements, storage configuration. This request captures the desired Edge Cluster characteristics at a logical level, abstracting provider-specific details.

A provisioning template defines the complete topology of the Edge Cluster to be created, specifying the compute hosts with their instance types and quantities, network configuration including public networks for device connectivity and private networks for internal communication, datastores with their capacities and performance characteristics, and the services to be deployed including the Edge Cluster Frontend, message broker, local object storage, and monitoring infrastructure.

The Edge Cluster provisioning invokes the appropriate provider driver to actualize the Edge Cluster within the target infrastructure. The driver authenticates with the provider’s API and orchestrates the creation of all necessary infrastructure resources: bare-metal hosts, virtual networks and subnets with appropriate isolation and connectivity, security groups and firewall rules enforcing network security policies, storage volumes and filesystems for persistent data, and r public IP addresses for external connectivity. Beyond just creating resources, the driver configures them appropriately, installing and configuring virtualization infrastructure, establishing virtual networking overlays, mounting storage volumes, and installing monitoring agents that will report metrics back to the central monitoring system.

Service deployment completes the Edge Cluster provisioning by instantiating the software components that make the Edge Cluster operational. The Edge Cluster Frontend is installed and configured to accept requests from devices, the message broker is deployed to provide the queue infrastructure for function execution, local object storage is

established for Edge Cluster specific data, and the monitoring infrastructure is configured to collect and export metrics.

Once provisioning completes, the new Edge Cluster is registered with the Cloud-Edge Manager, capturing its unique identifier, provider and location information for placement decisions, total capacity across compute, memory, storage, and specialized resources such as TEE, network endpoints including the Edge Cluster Frontend URLs. This registration makes the Edge Cluster visible to the AI-Enabled Orchestrator for placement decisions and enables the Cloud-Edge Manager to monitor and manage the cluster throughout its lifecycle.

Beyond initial provisioning, Edge Clusters support dynamic scaling operations that adapt their capacity to changing workload demands. Horizontal scaling adds new hosts to an Edge Cluster when additional capacity is needed or removes underutilized hosts to improve resource efficiency. The scaling operations can be triggered by various mechanisms: resource utilization thresholds that respond to immediate capacity constraints, recommendations from the AI-Enabled Orchestrator based on predictive workload models, or manual administrator actions.

6.1.1.3. Serverless Runtime Provisioning

The Cloud-Edge Manager manages the complete lifecycle of Serverless Runtimes within Edge Clusters, orchestrating their deployment, scaling, migration, and termination in response to workload demands and optimization objectives. Serverless Runtime deployment transforms a logical placement decision from the AI-Enabled Orchestrator into a running execution environment ready to consume and process function execution requests.

Host selection within the target Edge Cluster determines which physical host will execute the Serverless Runtime. The AI-Enabled Orchestrator performs this placement decision, evaluating hosts based on available capacity considering CPU, memory, storage, and specialized resources, current utilization to avoid overloading hosts, affinity and anti-affinity rules that control Serverless Runtime placement relative to other Runtimes, and power consumption to support sustainability objectives. The AI-Enabled Orchestrator generates a deployment plan specifying the selected host, which the Cloud-Edge Manager then executes to actualize the Serverless Runtime deployment.

Beyond initial deployment, Serverless Runtimes support various lifecycle operations that enable dynamic adaptation to changing conditions. Scaling operations implement horizontal elasticity, where additional Serverless Runtimes can be deployed in response to an increase in function execution requests, or existing Runtimes can be terminated when the workload decreases. This horizontal scaling approach allows the system to adapt capacity to demand by adjusting the number of active Serverless Runtimes rather than resizing individual instances.

Migration operations provide Serverless Runtime mobility within an Edge Cluster, moving a Serverless Runtime between hosts within the same cluster. These migrations use live migration techniques that preserve network connections and Serverless Runtime state while minimizing downtime. Migration serves multiple purposes: enabling host

maintenance without service disruption, balancing load across hosts to prevent hotspots, optimizing energy consumption by consolidating workloads onto fewer hosts and powering down idle hosts, and recovering from host failures by relocating affected Runtimes to healthy hosts. Workload redistribution across Edge Clusters is achieved not through Serverless Runtime migration but through device reassignment, where devices are assigned to different Edge Clusters, causing load to shift between clusters and triggering subsequent scaling and rebalancing operations within each cluster.

6.1.1.4. Metrics, Monitoring and Auditing

Comprehensive observability is essential for the Cloud-Edge Manager to maintain operational awareness across the distributed infrastructure and provide the data foundation for smart orchestration.

Infrastructure monitoring tracks the fundamental resources that underpin the COGNIT Framework's execution capabilities. Each physical host across all Edge Clusters reports computational resource utilization including CPU, memory, storage, and network consumption. These metrics flow continuously from distributed monitoring agents deployed throughout the infrastructure to the Cloud-Edge Manager centralized database that provides both real-time query capabilities and long-term historical retention.

Serverless Runtime monitoring provides visibility into the virtual execution environments where application functions actually run. The Cloud-Edge Manager tracks resource consumption for each Serverless Runtime, capturing how much of their allocated capacity they are actively using and how this usage varies over time. Function execution metrics capture the operational characteristics of the workloads themselves, including execution frequency, duration, success rates. This execution telemetry informs capacity planning, identifies performance bottlenecks, and allows the AI-Enabled Orchestrator to understand workload behavior patterns that drive smart placement decisions.

Energy consumption monitoring represents a distinctive capability that enables sustainability-aware orchestration. Each host measures its power consumption, and these measurements are attributed down to individual Serverless Runtimes based on their resource utilization patterns. This energy consumption visibility enables the COGNIT Framework to make orchestration decisions that consider not just performance but also carbon footprint, supporting applications that prioritize environmental sustainability. Geographic location data for both infrastructure resources and devices enriches this energy context, enabling location-aware optimizations that account for regional variations in energy sources and carbon intensity.

Network latency measurements between devices and Edge Clusters enable placement decisions that meet application latency requirements. Application workload characteristics including function invocation rates inform capacity provisioning. By correlating these demand-side metrics with infrastructure supply-side metrics, the Cloud-Edge Manager enables optimization that balances resource availability against actual usage patterns.

6.1.1.5 Authentication & Authorization

Device authentication and authorization within the COGNIT Framework ensure that only legitimate devices can execute functions and access resources while maintaining security across the distributed cloud-edge infrastructure. The framework employs a delegated authentication model combined with cryptographic token-based authorization to enable secure, scalable device access.

Device authentication follows a delegated model where the COGNIT Frontend authenticates devices against the Cloud-Edge Manager's identity system. Upon successful authentication, the COGNIT Frontend issues cryptographically signed authorization tokens that devices use for subsequent interactions with Edge Clusters. This delegation separates identity verification from authorization enforcement, enabling devices to interact with distributed Edge Clusters without requiring each cluster to perform centralized authentication checks on every request.

The authorization tokens employ public key cryptography and carry embedded authorization information including the device's identity, permitted operations, and constraints such as expiration times. Edge Cluster Frontends independently verify token authenticity and extract authorization information using the public key, eliminating the need to consult a centralized authority for every device request. This decentralized verification model enables scalability and eliminates single points of failure, as Edge Clusters can authorize device requests locally.

6.1.1.6. Plan Executor

The Plan Executor bridges the AI-Enabled Orchestrator's smart decision-making with the Cloud-Edge Manager's infrastructure operations. The AI-Enabled Orchestrator produces high-level deployment plans expressing desired infrastructure changes, and the Plan Executor translates these into concrete sequences of operations.

Deployment plans comprise structured sequences of actions, each specifying an operation type, target resources, and execution parameters. Plans may include Serverless Runtime operations such as creation, scaling, migration, or termination, as well as scaling existing Edge Clusters . This structured format enables the AI-Enabled Orchestrator to express complex multi-step optimizations while abstracting from operational implementation details.

6.2. AI-Enabled Orchestrator

The AI-Enabled Orchestrator represents the smart layer of the COGNIT Framework, responsible for making optimal resource management decisions across the distributed cloud-edge infrastructure. It transforms the COGNIT Framework from a reactive resource management system into a proactive, self-optimizing platform that continuously learns from operational experience and anticipates future needs.

The AI-Enabled Orchestrator integrates with a modular resource scheduler framework that provides resource selection and workload optimization capabilities. The framework's Scheduler Manager coordinates the scheduling process by receiving placement and

optimization requests, invoking schedulers to generate plans, validating plans, and submitting validated plans to the Plan Executor for execution.

In the COGNIT Project, scheduler drivers extend the scheduler framework with energy-aware optimization capabilities. These drivers implement optimization algorithms that minimize power consumption and reduce resource contention when placing and migrating Serverless Runtime VMs. The drivers communicate with the Scheduler Manager via the scheduler interface, submitting optimization plans for execution.

The AI-Enabled Orchestrator, as shown in Figure 6.3, consists of several integrated components that work together to provide comprehensive resource management:

- **Predictive Analytics**: Component for workload prediction and scaling recommendations based on historical metrics and workload patterns.
- **Device Load Forecasting**: Component for estimating device workload based on system-wide metrics.
- **Multi-Cluster Optimizer**: Optimizes device-to-cluster assignments across the entire cloud-edge continuum to minimize carbon emissions.
- **Cluster Optimizer**: Optimizes Serverless Runtime initial placement and workload optimization within clusters.
    - **Initial Placement**: Computes optimal host assignments for new Serverless Runtimes.
    - **Workload Optimization**: Analyzes current Serverless Runtime distributions and generates migration plans to reduce contention and energy usage.

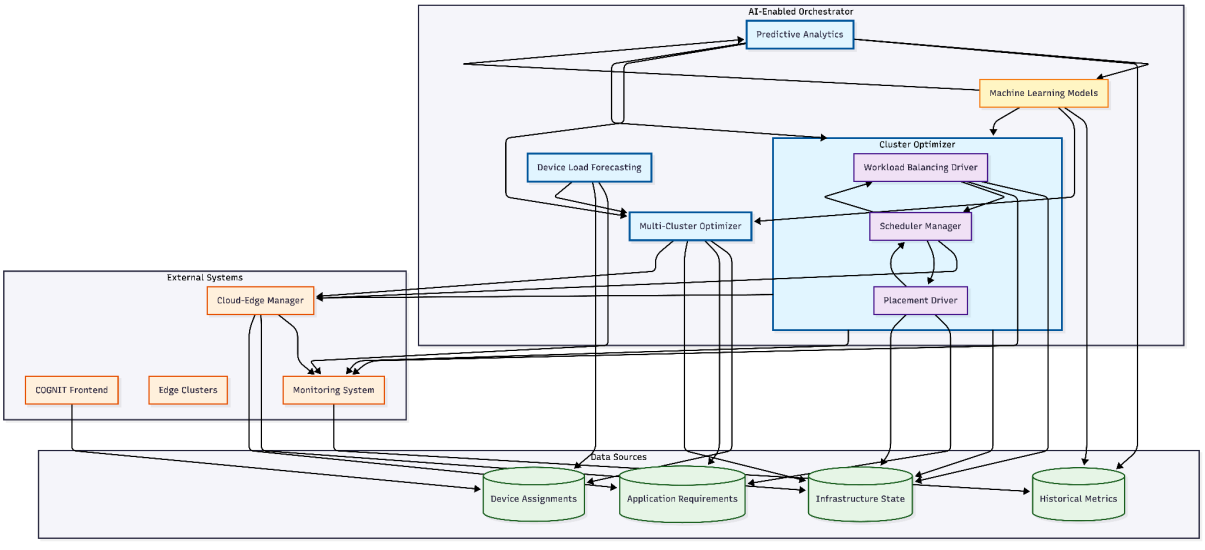

Figure 6.3. AI-Enabled Orchestrator Architecture.

The AI-Enabled Orchestrator operates asynchronously, continuously monitoring the device-to-cluster assignment database, infrastructure state, and evolving conditions across the cloud-edge continuum. When it identifies opportunities for optimization, whether due to changes in infrastructure availability, shifts in workload patterns, updates to device

requirements, or provisioning of new Edge Clusters, the Orchestrator computes improved assignments and updates the database entries accordingly. This asynchronous optimization model ensures that the COGNIT Frontend can always provide low-latency responses to device requests by reading current assignments from the database, while the AI-Enabled Orchestrator continuously refines these assignments in the background to maintain optimal workload distribution.

Smart resource management capabilities enable the AI Orchestrator to make placement decisions that optimize across multiple competing objectives simultaneously. When selecting an Edge Cluster for a device, the orchestrator evaluates candidates based on latency, response time, current load distribution, and energy consumption characteristics, balancing performance requirements against sustainability goals. For Serverless Runtime placement within Edge Clusters, the orchestrator analyzes workload characteristics to predict potential performance interference when co-locating functions, enabling dense resource utilization while maintaining performance isolation. These placement decisions consider not only current infrastructure state but also predicted future conditions, positioning resources proactively rather than reactively responding to saturation.

By learning from historical metrics collected across the infrastructure, the AI-Enabled Orchestrator builds models that forecast future resource requirements, function execution patterns, and infrastructure behavior. These predictions enable proactive scaling operations that provision additional Serverless Runtime capacity before workload increases manifest as performance degradation, and proactive device reassignment that anticipates infrastructure changes before they impact service quality. The orchestrator continuously adapts its models as workload patterns and infrastructure characteristics evolve, ensuring that predictions remain accurate over time and that the system becomes progressively more effective with operational experience.

Multi-objective optimization capabilities allow the AI-Enabled Orchestrator to balance competing goals that cannot be simultaneously maximized. Performance objectives such as minimizing latency and maximizing throughput often conflict with sustainability objectives like minimizing energy usage and carbon footprint. The orchestrator formulates these trade-offs explicitly as multi-objective optimization problems, identifying solutions that represent the best achievable balance between objectives.

Energy-aware orchestration represents a distinctive capability that extends beyond traditional performance optimization. The COGNIT Framework monitors carbon intensity across Edge Cluster locations, identifying opportunities to shift workloads toward locations with more favorable energy profiles. When renewable energy availability increases at a particular location, the orchestrator can proactively reassign devices to Edge Clusters in that region or migrate Serverless Runtimes between hosts to consolidate workloads and power down underutilized hosts. These energy-aware decisions respect performance constraints, ensuring that sustainability optimizations do not violate application requirements for latency or response time.

6.2.1 Machine Learning Models

Machine learning models form the intelligence foundation of the AI-Enabled Orchestrator, enabling it to learn from historical patterns, predict future behavior, and make optimal decisions across diverse optimization objectives. These models transform the streams of monitoring data collected from the distributed infrastructure into actionable insights that drive proactive orchestration and continuous optimization.

The learning models capability enables the orchestrator to predict Serverless Runtime scaling requirements based on historical metrics and workload patterns. The models support workload forecasting using time-series analysis and estimate response times and resource utilization based on current and predicted workload patterns.

The Device Load Forecasting component provides workload estimation for individual devices by aggregating system-wide metrics. The component periodically collects metrics from services deployed across Edge Clusters, including CPU usage and queue depth metrics. It calculates estimated load for each registered device based on total CPU usage and backlog, and updates device assignments with current estimated load values. This enables the Multi-Cluster Optimizer to make informed decisions about device-to-cluster assignments.

Workload characterization models capture and understand the behavioral patterns of application functions executing within the framework. These models analyze historical execution metrics to predict future function invocation rates, execution durations, and resource consumption patterns. By understanding workload characteristics, the AI-Enabled Orchestrator can provision Serverless Runtimes proactively before demand materializes, avoiding reactive scaling delays that could impact performance.

6.2.2 Smart Management of Cloud-Edge Resources

Smart resource management within the AI-Enabled Orchestrator operates across multiple orchestration tasks that span reactive decision-making in response to immediate requests and proactive optimization based on predicted future conditions. These orchestration capabilities leverage the machine learning models and comprehensive infrastructure visibility to continuously optimize resource allocation while respecting application requirements and policy constraints.

Smart resource management operates through two complementary optimization components that operate at different levels of the infrastructure hierarchy.

The **Multi-Cluster Optimizer** addresses resource management at the global level, optimizing device-to-cluster assignments and the number of Serverless Runtimes across the entire cloud-edge continuum. This component:

- **Reads device assignments**, including device IDs, current cluster assignments, application requirements, and estimated load values calculated by the Device Load Forecasting component.

- **Fetches application requirements** from the Cloud-Edge Manager, including constraints such as flavour, confidentiality requirements, provider preferences, and geolocation constraints.
- **Filters feasible clusters** for each device based on application requirements, ensuring that only clusters that can satisfy device constraints are considered.
- **Runs optimization algorithms** to find optimal cluster assignments that minimize energy consumption and carbon footprint across all clusters, while avoiding resource contention if feasible.
- **Triggers cluster scaling** operations based on optimization results to adjust Serverless Runtime cardinality.
- **Issues dynamic scaling** of Edge Cluster hosts (infrastructure horizontal scaling) when the chosen assignment would still lead to contention or when aggregate load requires additional capacity.
- **Updates device assignments** as clusters finish scaling, ensuring consistency between optimization decisions and actual resource allocation.

The optimizer minimizes greenhouse gas emission intensity by considering carbon intensity and power consumption across clusters. It adjusts device-to-cluster assignments and the number of used VMs according to available resources and predicted requirements. First, it tries to perform allocation without any resource contention if feasible, keeping clusters running with high performance and additional available capacities. The optimizer periodically runs optimization cycles, considering multiple objectives simultaneously: minimizing energy consumption, reducing carbon footprint, and satisfying application requirements.

Contention can still arise after optimization for two main reasons. First, the optimizer balances carbon emission and energy objectives against contention; when feasible clusters are limited (e.g. by latency or confidential computing requirements), the best emission-minimizing solution may concentrate devices on fewer clusters and thus leave residual contention. Second, an increase in the number of requests or devices raises the average load on clusters over time, so even a previously contention-free assignment may require more capacity. In both cases, the Multi-Cluster Optimizer can trigger dynamic scaling of Edge Cluster hosts: when it finds a solution where contention remains or when it detects that load growth demands more infrastructure capacity, it issues scaling requests so that the Cloud-Edge Manager adds (or removes) hosts within the affected Edge Clusters, aligning infrastructure capacity with the optimized device-to-cluster assignment and current or forecasted demand.

Cluster Optimizer. Addresses resource management at the cluster level, optimizing initial placement and workload distribution within individual Edge Clusters. This component performs both initial Serverless Runtime placement and cluster-wide workload optimization.

The Cluster Optimizer implements energy-aware optimization algorithms that simultaneously minimize power consumption and resource contention when placing Serverless Runtimes on hosts within a cluster. The optimizer:

- Uses power consumption metrics to make energy-efficient placement decisions.

- Evaluates resource contention by analyzing co-location of Serverless Runtimes on the same host, considering interference effects that could degrade performance.
- Considers host capacity (CPU, memory) and current utilization when making placement decisions, ensuring efficient resource utilization while minimizing energy consumption
- Implements placement and optimization operations:
    - Initial placement: When new Serverless Runtimes are instantiated, the optimizer selects the optimal host based on capacity and requested resources.
    - Workload optimization: Periodically generates energy-aware optimization plans to redistribute Serverless Runtimes across hosts within a cluster, minimizing resource contention and power consumption.

The Cluster Optimizer uses a two-step approach that enables consideration of both resource contention and power consumption: in the first step, it finds the minimal possible resource contention; in the second step, it minimizes power consumption by constraining resource contention to its previously established minimum. This approach minimizes resource contention (allowing good runtime performance when possible) while preserving power consumption as low as possible for the target contention level.

The optimization plans generated by the Cluster Optimizer are executed by the Plan Executor, which handles Serverless Runtime deployment and migration operations.

This multi-level optimization approach enables COGNIT to efficiently manage resources across the entire cloud-edge continuum, from global device-to-cluster assignments down to individual Serverless Runtime placement within clusters, while satisfying application requirements and minimizing energy consumption and carbon footprint.

7. Secure and Trusted Execution of Computing Environments on the Multi-Provider Cloud-Edge Continuum

The COGNIT Framework implements secure and trusted execution capabilities to protect sensitive workloads and data in the distributed cloud-edge continuum. This section describes two key security mechanisms: policy-based authorization through cryptographic tokens, and confidential computing support for hardware-level protection of data-in-use.

Policy-based authorization and confidential computing work together to provide comprehensive security:

- **Layered Protection**: Authorization tokens control access at the application layer, while confidential computing provides hardware-level protection.
- **Defense in Depth**: Multiple security mechanisms ensure that compromise of one layer does not compromise the entire system.
- **Zero Trust**: No component is trusted by default—authorization is verified for every request, and execution occurs in isolated Trusted Execution Environments (TEEs) when required.

This integrated approach ensures that sensitive workloads in the COGNIT Framework are protected both during access (through authorization tokens) and during execution (through confidential computing), providing end-to-end security for the cloud-edge continuum.

7.1. Policy-Based Authorization with Cryptographic Tokens

The COGNIT Framework employs authorization tokens for decentralized, fine-grained access control across the distributed infrastructure. These tokens enable devices to interact with Edge Clusters without requiring centralized authorization checks on every request, which is critical for edge environments.

The framework uses authorization tokens that employ public key cryptography to carry embedded authorization information. Each token encapsulates:

- **Device Identity**: Unique identifier for the authenticated device.
- **Authorized Operations**: Permissions for specific actions (function execution, data upload, etc.).
- **Constraints**: Time-based validity periods.
- **Cryptographic Signature**: Ensures token authenticity and integrity.

When a device authenticates with the COGNIT Frontend, it receives an authorization token that it uses for all subsequent interactions with Edge Clusters. Edge Cluster Frontends independently verify token authenticity and extract authorization information using the public key, eliminating the need to consult a centralized authority for every device request.

The COGNIT Frontend manages the complete token lifecycle:

- **Token Generation**: Upon successful device authentication, the COGNIT Frontend generates a cryptographically signed token with appropriate permissions.
- **Token Validation**: Edge Cluster Frontends verify token signatures and extract authorization information.

The COGNIT Framework has experimented with Biscuit tokens, which provide advanced policy-based authorization capabilities. Biscuit tokens support:

- **Facts**: Embedded statements about the token holder (e.g., device identity, security clearance, geographic location).
- **Policies**: Rules that define what operations are permitted based on facts and context.
- **Caveats**: Additional constraints that can be added to tokens (e.g., time-based restrictions, resource limits).
- **Attenuation**: Ability to create restricted tokens from existing tokens for delegation scenarios.

7.2. Confidential Computing

Confidential Computing provides hardware-level protection for sensitive workloads by ensuring that data and code remain protected even from privileged system software,

including hypervisors and operating systems. This is particularly important for edge computing scenarios where infrastructure may be located in semi-trusted or untrusted environments.

The COGNIT Framework has fully integrated confidential computing capabilities, enabling devices to specify confidential computing as an application requirement. When a device uploads its application requirements to the COGNIT Frontend, it can include a confidential computing requirement that ensures function execution occurs in a Trusted Execution Environment (TEE).

Devices can specify confidential computing as an application requirement when uploading their requirements to the COGNIT Frontend. The requirement is stored and made available to the AI-Enabled Orchestrator, which uses it as a constraint in Edge Cluster selection:

1. **Requirement Specification**: Device includes `confidential_computing: true` in application requirements.
2. **Orchestrator Filtering**: AI-Enabled Orchestrator filters Edge Clusters to only those with CC-capable hosts.
3. **Placement Decision**: Orchestrator selects Edge Cluster with CC-capable infrastructure.
4. **Serverless Runtime Deployment**: Cloud-Edge Manager deploys Serverless Runtime as a confidential VM using the appropriate TEE technology.

When a confidential computing requirement is specified, the Cloud-Edge Manager deploys Serverless Runtimes as confidential virtual machines. The deployment process is transparent to the application, that is functions execute normally within the confidential VM, but with hardware-level protection ensuring that:

- Data-in-use is protected even if the hypervisor is compromised.
- Function code and execution state cannot be inspected by privileged software.
- Memory contents remain encrypted throughout execution.

8. Software Requirements

This section provides an overview of the Software Requirements.

8.1. Device Client

SR1.1 Interface with COGNIT Frontend

Description: Implementation of the communication of the Device Client with the COGNIT Frontend.

- The Device Client shall be able to ask for the most suitable (or a set of) Edge Cluster to which it should offload functions.
- The Device Client shall be able to upload application requirements to the COGNIT Frontend and update them.
- The Device Client shall be able to upload application functions to the COGNIT Frontend.
- The Device Client shall be able to upload data from the device to the COGNIT Frontend.
- The Device Client shall be able to request to transfer data from external resources to the COGNIT Frontend.

SR1.2 Interface with Edge Cluster

Description: Implementation of the communication between a Device Client and an Edge Cluster.

- The Device Client shall be able to invoke an execution of a function at the assigned Edge Cluster.
- The Device Client shall be able to upload data to the assigned Edge Cluster.

SR1.3 Programming languages

Description: Support for different programming languages.

- The Device Client shall support the C programming language.
- The Device Client shall support the Python programming language.

SR1.4 Low memory footprint for constrained devices

Description: Low memory footprint for constrained devices.

- The Device Client supporting the C programming language shall have a memory footprint lower than 500 kB.
- In order to be able to use the Device Client, the Device Application shall be able to implement an HTTP client with TLS capabilities.

SR1.5 Security

Description: The Device Client must be secured.

- All the communications between the Device Runtime and the COGNIT Frontend shall use cryptographically signed tokens for authentication and authorization.
- All the communications between the Device Runtime and the Edge Cluster shall use cryptographically signed tokens for authentication and authorization.
- Device Runtime shall take into account the latest legislative frameworks, such as the NIS2 directive, the GDPR, and the CRA.

SR1.6 Collecting Latency Measurements

Description: Latency measurements against Edge Clusters should be acquired by the Device Client.

- The Device Client shall implement a mechanism for latency measurement towards the assigned Edge Cluster.

8.2. COGNIT Frontend

SR2.1 COGNIT Frontend

Description: Provides an entry point for devices to communicate with the COGNIT Framework for offloading the execution of functions and uploading global data. The COGNIT Frontend:

- Must be able to authenticate the Device Client and return a cryptographically signed token once the device is authenticated.
- Must interface with the Cloud-Edge Manager to delegate the authentication of devices.
- Must provide Edge Cluster assignments to devices by reading from the shared assignment database that the AI-Enabled Orchestrator updates asynchronously.
- Must be able to provide the Device Client with information about Edge Clusters that can be used to offload functions.
- Must provide an endpoint for storing application requirements and other metadata (for instance, Device Client current location) from the Device Client, in a way that is accessible by the AI-Enabled Orchestrator.
- Must provide an endpoint for uploading files related to the execution of functions and global data.

8.3. Edge Cluster

SR3.1 Edge Cluster Frontend

Description: The Edge Cluster must provide an interface (Edge Cluster Frontend) for the Device Client to offload the execution of functions and to upload local data that is needed to execute the function. The Edge Cluster Frontend:

- Must provide an endpoint for the execution of functions previously uploaded through the COGNIT Frontend.
- Must provide an endpoint for uploading data from the Device Client to the Edge Cluster, to be stored locally at the Edge Cluster.
- Must interface with Serverless Runtimes deployed within the Edge Cluster in order to execute functions as requested by the Device Client.
- Must provide an endpoint for measuring latency from the Device Client.

SR3.2 Secure and Trusted Serverless Runtimes

Description: The Serverless Runtime is the minimal execution unit for the execution of functions offloaded by Device Clients.

- The Serverless Runtime must expose a REST API for all execution variants for the Python and C Executor used by the Edge Cluster Frontend.
- The Cloud-Edge Manager must provide a base Serverless Runtime image that contains the following minimal capabilities: a REST API interface for executing functions, a secure channel for the communication within the Edge Cluster and a mechanism to push function-related metrics to the monitoring service.
- The COGNIT Framework must provide an automated procedure for building secure and trusted images (vulnerability scans, security assessment) related to different flavours[2] of Serverless Runtimes, according to the need of the different Use Cases.
- The Serverless Runtime must provide a mechanism to report function execution metrics in a way that can be consumed by the AI-Enabled Orchestrator.

8.4. Cloud-Edge Manager

SR4.1 Provider Catalogue

Description: Implement a backend to persist information about the available providers that can be used to extend the capacity of the COGNIT infrastructure.

- Provider Catalogue data model must be persistent.
- Provider Catalogue must implement an API to manage providers entries.
- Provider Catalogue must be able to filter providers according to latency, costs, energy consumption and/or specific characteristics.

[2] With “flavours” we are referring to different VM templates available for creating specific Serverless Runtimes.

SR4.2 Edge Cluster Provisioning

Description: The Cloud-Edge Manager must be able to provision Edge Clusters as a set of software-defined compute, network, storage on any cloud/edge location available in the Provider Catalogue.

- Cloud-Edge Manager must provide an API to manage the lifecycle (provision/update/scale/deprovision) of Edge Clusters.

SR4.3 Serverless Runtime Deployment

Description: The Cloud-Edge Manager must be able to deploy Serverless Runtimes as Virtualized Workloads within an Edge Cluster.

- Cloud-Edge Manager shall provide an API to manage the life cycle (deploy/update/scale/migrate) of Serverless Runtimes.

SR4.4 Metrics, Monitoring, Auditing

Description: Edge-Clusters monitoring, Serverless Runtimes metrics collection and continuous security assessment.

- A distributed system shall be used for monitoring Edge Cluster entities (hypervisor, virtual/overlay networks, datastores and power consumption).
- A distributed system shall be used for collecting function metrics from Serverless Runtimes deployed across the cloud-edge infrastructure.

SR4.5 Authentication & Authorization

Description: Authentication and authorization mechanisms for accessing cloud-edge infrastructure resources by the devices for offloading workloads.

- Cloud-Edge Manager must provide mechanisms allowing the COGNIT Frontend to delegate authentication and authorization on behalf of the Device Client.
- Devices will use cryptographically signed tokens for the authorization to use resources provided by the Cloud-Edge Manager.

SR4.6 Plan Executor

Description: The Plan Executor is responsible for converting plans provided by the AI-Enabled Orchestrator in Cloud-Edge Manager actions for the life cycle management of Edge Clusters and Serverless Runtimes. It provides the following capabilities:

- A driver for executing Cloud-Edge Manager actions related to the life cycle management of Serverless Runtimes (create, update, migration, delete).

- A driver for executing Cloud-Edge Manager actions for the provisioning, deprovisioning and horizontal elasticity of Edge Clusters.

8.5. AI-Enabled Orchestrator

SR5.1 Building Learning Models

Description: Provide AI/ML models trained with input from collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters and Serverless Runtimes deployed across the distributed cloud-edge continuum. Main features provided are:

- Workflows for training, testing and validation of AI/ML models for diverse downstream tasks: workload characterization, metrics prediction (such as CPU usage, function arrival time, workload, resource requirements), energy-aware scheduling.
- AI/ML Model repository for tracking and versioning ML model artefacts.
- Provide mechanisms for retraining and model updates according to the workload and infrastructure dynamics.
- Integration with the Cloud-Edge Manager for ML model inference.

SR5.2 Smart Management of Cloud-Edge Resources

Description: The AI-Enabled Orchestrator is responsible for the automated management of cloud-edge continuum resources in order to optimize the performance of the applications that are offloading functions to the COGNIT Framework. The AI-Enabled Orchestrator includes the following capabilities.

- Leverages AI/ML models and energy-aware optimization algorithms for the smart (re)allocation of Serverless Runtimes across the Edge Clusters.
- Dynamic workload and resource optimization in response to system-changing conditions of cloud-edge infrastructure using horizontal elasticity (i.e., creation, scaling and deletion of Edge Clusters).
- Interfaces with the Plan Executor for the submission of deployment plans.

8.6. Secure and Trusted Execution of Computing Environments

SR6.1 Advanced Access Control

Description: Implement policy-based access control to support security policies on geographic zones that define a security level for specific areas.

- Cloud-Edge Manager shall install and configure a policy-based access control service to enforce and manage security policies.

- Cloud-Edge Manager shall define appropriate security policies, based on various attributes (location, device type, etc.).

SR6.2 Confidential Computing

Description: Enable privacy protection for the application workloads at the hardware level using Confidential Computing (CC) techniques.

- The resource definition model shall be expanded for CC-capable hosts to be tagged as such.
- CC-capable constraint shall be included in the orchestrator to deploy workloads requiring CC to appropriate devices and hosts.

SR6.3 Federated Learning

Description: Enhance privacy of AI workloads that have confidentiality requirements preventing the exchange of information for training. Federated Learning techniques enable confidential or private data processing under the control of the data owner or controller, with only learned models shared.

- Federated Learning Frameworks to implement such approach of AI model training while preserving data privacy need to be surveyed
- Integration of such FL frameworks with the COGNIT framework to be investigated taking into account COGNIT Framework architectural restrictions and requirements.

9. User to Software Requirements Matching

This section provides the results of the requirement engineering process, which derives system requirements (functional and non-functional) from functional gaps in order to implement a system that fulfils both user requirements and sovereignty, sustainability, interoperability and security requirements. The following tables (Tables 9.1 to 9.6) summarise the resulting system requirements:

Id	Description	Source
SR1.1	Interface with COGNIT Frontend	UR0.1 UR0.2 UR0.3 UR0.4 UR0.8 UR0.9 UR1.2 UR1.4 UR1.5 UR2.3 UR3.1 UR3.2 UR3.3 UR4.1 UR4.2 UR4.3 IR0.3 SER0.1 SER0.3
SR1.2	Interface with Edge Cluster	UR0.2 UR0.3 UR0.4 UR1.3 IR0.3
SR1.3	Programming Languages	UR0.1 UR3.3 IR0.3
SR1.4	Low memory footprint for constrained devices	UR3.1 SER0.1 SER0.5 SER0.6

SR1.5	Security	SER0.1 SER0.2
SR1.6	Collecting Latency Measurements	SER0.1 SER0.2

Table 9.1. System requirements for the Device Client

Id	Description	Source
SR2.1	COGNIT Frontend	UR0.2 UR0.3 UR0.4 UR0.6 UR0.7 UR0.8 UR0.9 UR2.1 UR4.1 UR4.2 UR4.3 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2 SER0.3 SER0.5

Table 9.2. System requirements for the COGNIT Frontend.

Id	Description	Source
SR3.1	Edge Cluster Frontend	UR0.1 UR0.2 UR0.3 UR0.4 UR0.10 UR1.1 UR1.2 UR1.3

		UR1.4 UR2.1 UR2.3 SUR0.1 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2 SER0.6 SOR0.4
SR3.2	Secure and Trusted Serverless Runtimes	UR0.1 UR1.2 UR2.3 IR0.1 IR0.2 SER0.4 SER0.6

Table 9.3. System requirements for the Edge Cluster

Id	Description	Source
SR4.1	Provider Catalogue	UR0.5 UR4.1 SOR0.1 SUR0.1 IR0.1 IR0.2 IR0.3
SR4.2	Edge Cluster Provisioning	UR0.5 UR1.2 SOR0.3 SUR0.1 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2
SR4.3	Serverless Runtime Deployment	UR1.1

		UR2.3 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2 SER0.4
SR4.4	Metrics, Monitoring, Auditing	UR1.4 UR2.1 SUR0.1 IR0.1 IR0.2 IR0.3
SR4.5	Authentication & Authorization	UR0.8 UR1.1 IR0.1 IR0.2 IR0.3
SR4.6	Plan Executor	UR4.1 UR4.2 UR4.3 IR0.1 IR0.2 IR0.3

Table 9.4. System requirements for the Cloud-Edge Manager.

Id	Description	Source
SR5.1	Building Learning Models	IR0.2 SUR0.2 SUR0.3 UR1.2 UR2.1 UR2.2
SR5.2	Smart Management of Cloud-Edge Resources	UR0.6 UR0.7 UR1.2 UR1.3 UR2.2

	UR3.2 UR4.1 UR4.2 UR4.3

Table 9.5. System requirements for the AI-Enabled Orchestrator.

Id	Description	Source
SR6.1	Advanced Access Control	UR0.8 UR1.1 UR4.1 SER0.2 SER0.5
SR6.2	Confidential Computing	UR1.1 SER0.1 SER0.2 SER0.4
SR6.3	Federated Learning	UR1.1 IR0.2 SER0.2 SER0.4

Table 9.6. System requirements for Secure & Trusted Execution of Computing Environments.

PART III. Verification and Implementation Plan

10. Software Build and Verification

The COGNIT software development model has ensured that components are delivered securely, timely, and with a high-quality level. Figure 10.1 shows a DevSecOps approach where sample security controls have been associated with each phase of the development. The phases have covered the entire lifecycle of the platform, from the planning phase where requirements have been gathered and specifications drafted to the operations and monitoring phases where the platform has been made available to end-users. This method has relied heavily on automation to orchestrate the platform life cycle, using techniques such as continuous integration/deployment (CI/CD), infrastructure as code (IaaC), and observability.

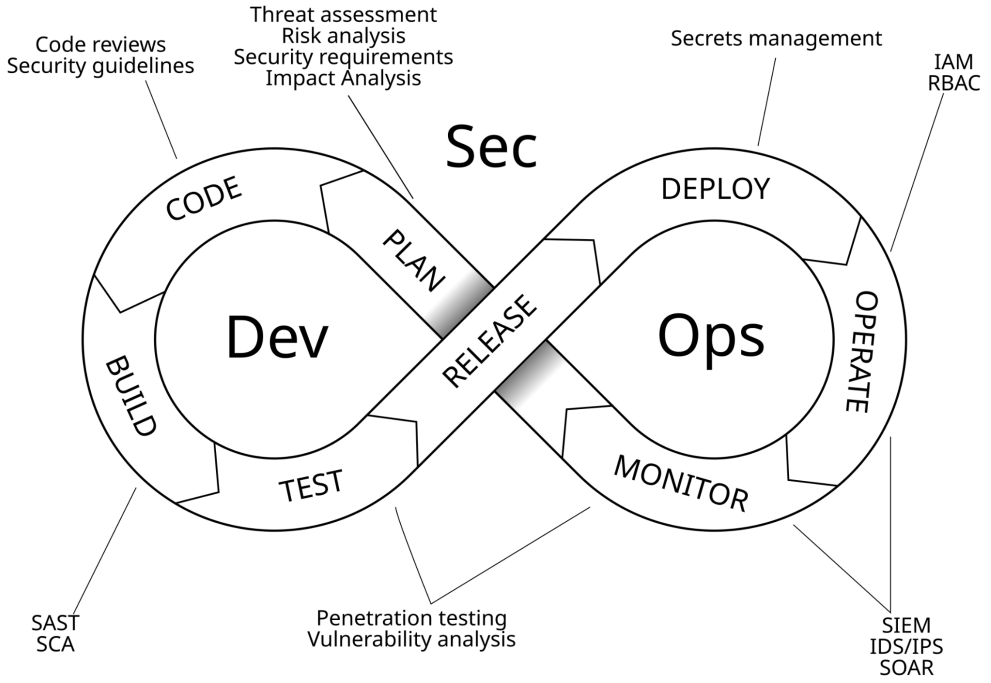

Figure 10.1. DevSecOps integrates development, operations, and security activities.

10.1. Verification Methodology

The goal of the verification process is to assess that the functional components of the software platform conform to the Software Requirements identified in Section 8. This, in turn, has been used to validate that the COGNIT Framework is feature-complete and able to achieve the objectives of the Projects' Use Cases.

In order to support the agile development adopted in this project, the verification process is integrated with the software development procedure mentioned before. The methodology is structured as follows:

- **Verification scenario**. Describes a simple user story that captures one or more functional requirements of a software requirement of a component (see Section 9).
- **Verification test**. An automated testing program that exercises the functional aspects of the scenario. Each verification test is then integrated into the certification platform to certificate and test software releases.

10.2. Verification Scenarios

This section presents a list of verification scenarios for verifying the initial set of Software Requirements defined in Section 8. Each COGNIT Framework component is presented in a separate table that includes a brief description of each scenario.

SW Req.	Verification Scenario
SR1.1	VS1.1.1 The Device Client is able to get authorization from COGNIT and is able to send App valid requirements to the COGNIT Frontend. VS1.1.2 The Device Client is able to receive a valid Edge Cluster (effectively a valid Edge Cluster Frontend IP address). VS1.1.3 The Device Client is able to update the App requirements at any moment. VS1.1.4 The Device Client is able to receive a changed Edge Cluster seamlessly from a COGNIT's proactive decision making action. VS1.1.5 The Device Client is able to handle (upload/read) data on the COGNIT global layer.
SR1.2	VS 1.2.1 The Device Client is able to execute functions (either preloaded or by uploading it at the moment of execution) on the assigned Edge Cluster. VS1.2.2 The Device Client is able to handle (upload/read) data privately in the assigned Edge Cluster.
SR1.3	VS1.3.1 Test previously described validation scenarios implemented in C language. VS1.3.2 Test previously described validation scenarios implemented in Python language.

SR1.4	VS1.4.1 Test validation scenarios described above on a device with less than 500kB of RAM.
SR1.5	VS1.5.1 The Device Client is able to perform secure communications against the COGNIT Frontend and the assigned Edge Cluster Frontend with the acceptance of the authorization mechanism. VS1.5.2 The Device Client is not permitted any unauthorised action towards the COGNIT Frontend or the assigned Edge Cluster.
SR1.6	VS1.6.1 The Device Client is able to measure latencies to different Edge Clusters concurrently with other activities of the Client.

Table 10.1. Verification scenarios for Device Client.

SW Req.	Verification Scenario
SR2.1	VS2.1.1 Authenticate a Device against the COGNIT Frontend and verify that an authorization token is returned. VS2.1.2 Upload application requirements to the COGNIT Frontend and verify that a unique ID is returned for the application requirements. VS2.1.3 Upload application requirements and query the COGNIT Frontend for an Edge Cluster and verify that it meets the application requirements. VS2.1.4 Upload a function to the COGNIT Frontend and verify that a unique ID is returned for that function. VS2.1.5 Test uploading and downloading data by the device to and from the COGNIT Frontend.

Table 10.2. Verification scenarios for the COGNIT Frontend.

SW Req.	Verification Scenario
SR3.1	VS3.1.1 Instantiate a Serverless Runtime and verify that a device can request the execution of a function to the Edge Cluster Frontend and assert the result of the function. VS3.1.2 Test uploading and downloading data by the device to and from the Edge Cluster using a secure communication channel.

SR3.2	VS3.2.1 Build a Serverless Runtime image, customized for each Use Case, in an automated way.

Table 10.3. Verification scenarios for the Edge Cluster.

SW Req.	Verification Scenario
SR4.1	VS4.1.1 Listing the providers belonging to the Provider Catalogue. VS4.1.2 Filtering the providers according to a desired latency threshold on a geographic area. VS4.1.3 Filtering the providers according to energy consumption per hour threshold. VS4.1.4 Filtering the providers according to some specific hardware characteristics (e.g. Trusted Execution Environments).
SR4.2	VS4.2.1 A YAML file containing the information about the provision is provided to the Cloud-Edge Manager that creates a new Edge Cluster. VS4.2.2 Query the Cloud-Edge Manager to return the status of an Edge Cluster identified by its ID. VS4.2.3 Query the Cloud-Edge Manager to scale up/down the number of hosts of an Edge Cluster identified by its ID. VS4.2.4 Query the Cloud-Edge Manager to delete an Edge Cluster identified by its ID.
SR4.3	VS4.3.1 A YAML file containing the information about the deployment is provided to the Cloud-Edge Manager that creates a new Serverless Runtime. VS4.3.2 Query the Cloud-Edge Manager to return the status of a Serverless Runtime identified by its ID. VS4.3.3 Query the Cloud-Edge Manager to scale up/down the resources (CPU, memory and disks) of a Serverless Runtime identified by its ID. VS4.3.4 Query the Cloud-Edge Manager to update the deployment of the Serverless Runtime identified by its ID.

	VS4.3.5 Query the Cloud-Edge Manager to delete a Serverless Runtime identified by its ID.
SR4.4	VS4.4.1 Create an Edge Cluster and deploy a Serverless Runtime and check the metrics collected for a certain period of time.
SR4.5	VS4.5.1 Test the creation of new users and groups. VS4.5.2 Assign ACLs to designated users and test the creation of new Edge Clusters and Serverless Runtimes. VS4.5.3 Communicate with the COGNIT Frontend and the Edge Cluster Frontend using tokens.
SR4.6	VS4.6.1 Submit a plan to the Plan Executor for creating new Serverless Runtimes and verify the deployment of the Serverless Runtimes. VS4.6.2 Submit a plan to the Plan Executor for migrating a Serverless Runtime.

Table 10.4. Verification scenarios for the Cloud-Edge Manager.

SW Req.	Verification Scenario
SR5.1	VS5.1.1 List instances from Devices to Applications to System for metrics to be collected. VS5.1.2 Correlate and represent features that are ready to take as input to the Model. VS5.1.3 Feedback-aware performance check when training the model on represented features. VS5.1.4 Assess the ability in terms of AUROC score for each task (e.g., scheduling).
SR5.2	VS5.2.1 Assess the ability of workload and resource optimization in terms of energy and performance trade-off.

Table 10.5. Verification scenarios for the AI-Enabled Orchestrator.

SW Req.	Verification Scenario

SR6.1	VS6.1.1 Define a security policy that is based on a geographic zone attribute. VS6.1.2 Check enforcement of new security policy when edge device moves closer from one edge node to another.
SR6.2	VS6.2.1 Deploy a function on a host that provides confidential computing capability. VS 6.2.2 Check that the function is executed inside the host trusted execution environment (TEE).
SR6.3	VS6.3.1 Perform training of the ML algorithm without exchanging local data. VS6.3.2 Check that the redistributed models for inference do not contain private data.

Table 10.6. Verification scenarios for the Secure & Trusted Execution of Computing Environments.

11. Instantiation of the COGNIT Architecture

This section identifies the final set of technologies that have been used, enhanced, integrated and/or implemented in COGNIT to instantiate the Architecture through the implementation of the Software Requirements described in Section 8. Based on the analysis in Section 2 with regards to sovereignty, sustainability, interoperability, and security requirements, the choice of technologies has prioritised the use of European open source alternatives.

SR	Description	New/Enh/Int	Implementation	Status
SR1.1	Interface with COGNIT Frontend	New component	Python SDK and C library with HTTPS client, certificate management, and Biscuit token support.	Completed
SR1.2	Interface with Edge Cluster	New component	HTTPS client for Edge Cluster Frontend communication with TLS support (integrated in the Device Client SDK).	Completed
SR1.3	Programming Languages	New component	Python 3.10+ SDK and C library.	Completed
SR1.4	Low memory footprint for constrained devices	New component	C library implementation with minimal dependencies and optimized memory footprint for IoT/embedded devices.	Completed
SR1.5	Security	New component	TLS support, Biscuit token handling integrated in the Device Client SDK.	Completed
SR1.6	Latency Measurement	New component	Built-in latency measurement and reporting capabilities integrated in the Device Client SDK	Completed
SR2.1	COGNIT Frontend	New component	FastAPI application with SQLite database and Biscuit token generation, deployed as service on the OpenNebula Frontend.	Completed
SR3.1	Edge Cluster Frontend	New component	FastAPI application with RabbitMQ integration (Pika), Biscuit token	Completed

			verification, Prometheus metrics export. Deployed as OneFlow Service Role in each Edge Cluster.	
SR3.2	Secure and Trusted Serverless Runtimes	New component	OpenNebula Virtual Appliance incorporating specific libraries according to the Use Case needs.	Completed
SR4.1	Provider Catalogue	Integration	Integration of the OpenNebula OneForm Provider Catalogue in the Cloud-Edge Manager	Partially Completed
SR4.2	Edge Cluster Provisioning	Integration	Integration of the OpenNebula OneForm component in the Cloud-Edge Manager (Development in progress).	Completed
SR4.3	Serverless Runtime Deployment	Integration	Serverless Runtimes defined as OpenNebula OneFlow Services.	Complete
SR4.4	Metrics, Monitoring, Auditing	Enhancement	Enhancement of OpenNebula Monitoring System.	Complete
SR4.5	Authentication & Authorization	Enhancement	Extended OpenNebula Authentication with Biscuit token support for devices, ACL support for fine-grained access control.	Completed
SR4.6	Plan Executor	Integration	Integration of OpenNebula Scheduler Plan Manager component.	Partially Completed
SR5.1	Building Learning Models	New component + Enhancement	Enhancement of OpenNebula AIOps SDK with new AI/ML models developed in COGNIT.	Completed
SR5.2	Smart Operations/Management of Cloud-Edge Resources	Enhancement	Enhancement of the OpenNebula Distributed Resource Scheduler through the integration of algorithms for Cloud-Edge resource optimization.	Partially Completed

SR6.1	Advanced Access Control	Enhancement	Enhancement of the OpenNebula support to ACLs and integration of Biscuit tokens.	Completed
SR6.2	Confidential Computing	Integration	Integration of OpenNebula drivers that expose CC hardware capabilities.	Completed
SR6.3	Federated Learning			Not Completed

Table 11.1. Final technologies used, enhanced, and/or implemented per Software Requirement.

PART IV. Development Report & Conclusions

12. Overall Development Status

12.1. Software Requirement Progress

The table below shows the status of each Software Requirement towards completion, following a simple colour code: – for activities that have not been started, ↻ for activities partially completed and ✓ for completed activities:

Software Requirements	Cycle 1 (M4-M9)	Cycle 2 (M10-M15)	Cycle 3 (M16-M21)	Cycle 4 (M22-M27)	Cycle 5 (M28-M33)
Device Client					
SR1.1 Interface with COGNIT Frontend			↻	✓	
SR1.2 Interface with Edge Cluster			↻	✓	
SR1.3 Programming languages	↻	↻	↻	✓	
SR1.4 Low memory footprint for constrained devices	–	–	–	✓	
SR1.5 Security	–	–	↻	✓	
SR1.6 Collecting Latency Measurements			–	↻	✓
COGNIT Frontend					
SR2.1 COGNIT Frontend			↻	✓	
Edge Cluster					
SR3.1 Edge Cluster Frontend			↻	✓	

SR3.2 Secure and Trusted Serverless Runtimes	↻	↻	✓		
Cloud-Edge Manager					
SR4.1 Provider Catalogue	–	–	–	–	↻
SR4.2 Edge Cluster Provisioning	–	–	–	–	✓
SR4.3 Serverless Runtime Deployment	↻	↻	✓		
SR4.4 Metrics, Monitoring, Auditing	↻	↻	↻	↻	✓
SR4.5 Authentication & Authorization	↻	–	↻	✓	
SR4.6 Plan Executor			–	↻	↻
AI-Enabled Orchestrator					
SR5.1 Building Learning Models	↻	↻	↻	✓	
SR5.2 Smart Management of Cloud-Edge Resources	↻	↻	↻	↻	↻
Secure and Trusted Execution of Computing Environments					
SR6.1 Advanced Access Control	↻	↻	↻	↻	✓
SR6.2 Confidential Computing	↻	↻	↻	↻	✓
SR6.3 Federated Learning	–	–	–	–	–

Table 12.1. Status of each Software Requirement towards completion, per research & innovation cycle.

12.2. Global KPIs Progress

The table below shows in a graphical way a summary of the current final status of each of the Project's Software Requirements in relation to its global KPIs: in green, activities already completed; in yellow the ones partially completed; in red, those that have not been started:

MEASURABLE & VERIFIABLE RESULT	KPI	Device Client						COGNIT Frontend	Edge Cluster		Cloud-Edge Manager						AI-Enabled Orchestrator		Secure & Trusted Execution of Computing Environments		
		SR1.1	SR1.2	SR1.3	SR1.4	SR1.5	SR1.6	SR2.1	SR3.1	SR3.2	SR4.1	SR4.2	SR4.3	SR4.4	SR4.5	SR4.6	SR5.1	SR5.2	SR6.1	SR6.2	SR6.3
Deployment of large-scale, highly distributed data processing environments.	**[KPI1.1]** Framework tested to scale up to tens of thousands of distributed nodes.																				
Adaptation by offering a scalable monitoring system that enables the use of AI/ML techniques and methods for the smart operation of cloud-edge environments.	**[KPI1.2]** New AI-enabled orchestration demonstrated through Validation Use Cases.																				
Adaptation by enabling the elasticity of the infrastructure with automated provisioning of edge PoPs.	**[KPI1.3]** Full provision of edge PoPs in "1-click" and in less than 10 minutes.																				
Deployment and operation of edge cloud environments incorporating infrastructure resources across the whole cloud-edge continuum.	**[KPI1.4]** Deployment of a geo-distributed Testbed with resources from on-premises datacenters, cloud provider, edge provider, 5G provider, and on-premises far edge.																				
Ability to operate both the infrastructure layer and the virtualized resources using a single pane of glass.	**[KPI1.5]** Unified GUI, CLI, and API able to manage all resource layers across the continuum.																				
Seamless access for end users.	**[KPI2.1]** Meet experience level agreements with dynamic needs without intervention of the user.																				
Application developers can define the execution environment for the FaaS Runtimes.	**[KPI2.2]** Definition of Execution Context, Base Image, Communication Patterns, and Deployment Requirements for performance, cost, security and energy requirements.																				
Developers can change the deployment requirements of the FaaS Runtime according to execution flow.	**[KPI2.3]** Migration across edge PoPs to meet application needs.																				
Early and continuous security assurance of FaaS Runtimes.	**[KPI2.4]** Security processes and controls automated and integrated into the runtime lifecycle following a DevSecOps approach.																				
Efficient orchestration of serverless workloads across the Continuum.	**[KPI3.1]** Backward compatibility with virtualization tech (e.g. KVM, Firecracker) and containers (e.g. K8s).																				
Able to migrate workloads in geo-distributed edge clouds, combining edge/cloud resources.	**[KPI3.2]** Migrate workloads across the Continuum with minimal downtime.																				
Intelligent self-adaptation of FaaS Runtime to changes in application behavior and data variability.	**[KPI3.3]** Automatic scale up/down of the microVM running the FaaS Runtime.																				
Use of confidential and trusted computing for secure computation of private data at the edge/cloud.	**[KPI3.4]** Use case demonstrating isolation of sensitive data as it's being processed.																				
Dynamic load balancing to adapt to changes in application needs and cloud-edge infrastructure.	**[KPI4.1]** Scalable, multi-variate AI-enabled modeling and optimization techniques.																				
Intelligent adaptation of cloud-edge continuum management.	**[KPI4.2]** New methods to create and retrain ML-based predictive behavioral forecasts for all major management considerations faced by cloud-edge continuum environments.																				
Hierarchical multi-constraint resource management layer to minimize environmental impact.	**[KPI4.3]** Reduction of at least 10% energy consumption compared with manual edge deployment.																				
Orchestration mechanisms to enforce coherent global security and governance policies.	**[KPI4.4]** Able to automate a European security policy on a multi-provider edge deployment.																				

Table 12.2. Final contribution of each Software Requirement towards meeting the Project's global KPIs, by Milestone.

12.3. Device Client

SR1.1 Interface with COGNIT Frontend
Status: COMPLETED
The Device Client provides a REST API client for all COGNIT Frontend endpoints. It supports authentication, application requirements management, function upload, and data upload to Global DaaS (COGNIT DaaS). The Device Client requests Edge Cluster assignments from the COGNIT Frontend based on application requirements. The implementation is available in both Python and C variants.

SR1.2 Interface with Edge Cluster
Status: COMPLETED
The Device Client provides a REST API client for Edge Cluster Frontend endpoints. It supports synchronous function execution and local data upload to Local DaaS (Edge Cluster DaaS). The Device Client handles Edge Cluster context caching for efficient subsequent executions.

SR1.3 Programming languages
Status: COMPLETED
The Device Client supports both Python and C programming languages. Two separate implementations are provided: a Python Device Client as a pip-installable package and a C Device Client as a library. Both implementations provide equivalent functionality with the same API semantics.

SR1.4 Low memory footprint for constrained devices
Status: COMPLETED
The C Device Client is optimized for minimal memory usage with a static footprint under 500kB including all dependencies.

SR1.5 Security
Status: COMPLETED

The Device Client authenticates with the COGNIT Frontend and uses Biscuit tokens returned by the COGNIT Frontend for subsequent requests to both the COGNIT Frontend and the Edge Cluster Frontend. All communications use TLS encryption.

SR1.6 Collecting Latency Measurements

Status: COMPLETED

The Device Client receives a set of Edge Clusters from the COGNIT Frontend and measures latency against each Edge Cluster Frontend using a ping mechanism. Measurements are collected in a background thread. The Device Client selects the Edge Cluster with the minimum latency from the provided set.

12.4. COGNIT Frontend

SR2.1 COGNIT Frontend

Status: COMPLETED

The COGNIT Frontend is implemented as a REST API application. It authenticates devices and delegates authentication to the Cloud-Edge Manager IAM system. Upon successful authentication, it generates and issues Biscuit tokens. The COGNIT Frontend interfaces with the AI-Enabled Orchestrator for Edge Cluster selection and provides devices with Edge Cluster information. Application requirements are stored in the Cloud-Edge Manager document pool, and device-to-cluster assignments are cached in a SQLite database. The COGNIT Frontend maintains a global function registry and provides Global DaaS for data storage.

12.5. Edge Cluster

SR3.1 Edge Cluster Frontend

Status: COMPLETED

The Edge Cluster Frontend is implemented as a REST API application deployed in each Edge Cluster with an integrated message broker. It provides endpoints for function execution request submission, execution status tracking, local data upload, and latency measurement. The Edge Cluster Frontend implements a queue-based architecture where Serverless Runtimes consume requests at their own pace . It integrates with Local DaaS for data storage. The Edge Cluster Frontend verifies Biscuit tokens for authorization.

SR3.2 Secure and Trusted Serverless Runtimes

Status: COMPLETED

The Serverless Runtime exposes a REST API interface for executing Python functions uploaded to the COGNIT Frontend by the Device Client. Serverless Runtime images are built using an automated pipeline. A base image provides the Python runtime and REST API interface. Specialized images with specific libraries are built according to the needs of different Use Cases. The Serverless Runtime exports metrics to Cloud-Edge Manager Monitoring System using prometheus exporters. Images are stored in the Cloud-Edge Manager image repository. The Serverless Runtime supports confidential computing through AMD SEV-SNP trusted execution environments, enabling hardware-level protection for sensitive workloads.

12.6. Cloud-Edge Manager

SR4.1 Provider Catalogue

Status: PARTIALLY COMPLETED

The Provider Catalogue provides REST API endpoints for creating providers with support for public clouds and on-premises infrastructures.

Tasks not completed
The Provider Catalogue provides a REST API for querying and filtering providers based on latency, energy consumption, and specific characteristics.

SR4.2 Edge Cluster Provisioning

Status: COMPLETED

Edge Cluster provisioning is implemented using OpenNebula OneForm. OneForm templates define Edge Cluster topology (hosts, networks, storage, services). Provider Drivers support public clouds and on-premises infrastructure. The system provides automated provisioning of complete Edge Cluster environments including compute hosts, virtual networks, storage datastores, Edge Cluster Frontend, Local DaaS (Edge Cluster DaaS, implemented as MinIO), and monitoring agents. A REST API is provided for Edge Cluster lifecycle management (create, scale, configure, delete).

SR4.3 Serverless Runtime Deployment

Status: COMPLETED

An OpenNebula OneFlow Service template is defined for the deployment of Serverless Runtimes. The Cloud-Edge Manager API for managing the lifecycle of Serverless Runtimes is based on OpenNebula OneFlow. The system supports deployment, scaling, migration, and termination of Serverless Runtimes within Edge Clusters.

SR4.4 Metrics, Monitoring, Auditing

Status: COMPLETED

The OpenNebula monitoring system is enhanced with a component for collecting energy metrics from hosts and Serverless Runtimes based on the Scaphandre tool. Metrics are pushed to the OpenNebula Monitoring System. The system collects infrastructure-level metrics (CPU, memory, network, storage utilization), Serverless Runtime metrics (resource consumption, function execution metrics), energy consumption metrics. Host and device locations are collected and integrated. Metrics are aggregated from distributed Edge Clusters and stored for historical analysis and made available to the AI-Enabled Orchestrator.

SR4.5 Authentication & Authorization

Status: COMPLETED

The COGNIT Frontend implements delegation mechanisms for authentication and authorization. The COGNIT Frontend delegates credentials provided by the Device Client to the Cloud-Edge Manager IAM system. An IAM mechanism based on Biscuit tokens provides fine-grained access control for FaaS functions. The system implements secure communication between the Device Client and the COGNIT Frontend, and between the Device Client and the Edge Cluster Frontend, using TLS encryption. Biscuit tokens enable decentralized authorization decisions at Edge Clusters without requiring constant communication with the COGNIT Frontend.

SR4.6 Plan Executor

Status: PARTIALLY COMPLETED

The Plan Executor translates optimization plans from the AI-Enabled Orchestrator into concrete infrastructure actions executed on the Cloud-Edge Manager. It receives

optimization plans from the Scheduler Manager and processes them sequentially, executing actions (deploy, migrate, poweroff) in order while handling dependencies and error recovery.

Tasks not completed:
Driver for executing Cloud-Edge Manager actions for the provisioning and deprovisioning of Edge Clusters.

12.7. AI-Enabled Orchestrator

SR5.1 Building Learning Models

Status: COMPLETED

The AI-Enabled Orchestrator extends the OpenNebula AIOps SDK for OneFlow Service predictions. An ML training pipeline collects data from the OpenNebula monitoring system, performs feature engineering, and trains models (LSTM, Transformer, Random Forest, Gradient Boosting, ARIMA, Fourier analysis) with hyperparameter tuning and model evaluation. Models are trained for workload prediction, resource utilization prediction, energy consumption prediction. The Device Load Forecasting component estimates device workload based on OneFlow service metrics. Integration with the Cloud-Edge Manager monitoring infrastructure provides training data and inference inputs.

SR5.2 Smart Management of Cloud-Edge Resources

Status: PARTIALLY COMPLETED

The AI-Enabled Orchestrator integrates with the OpenNebula Resource Scheduler Framework. The Multi-Cluster Optimizer daemon uses MILP optimization for device-to-cluster assignments. The Cluster Optimizer uses ILP optimization for intra-cluster placement, including initial placement and workload redistribution. The system generates deployment plans in XML format and submits them to the Plan Executor. Optimization objectives include latency, energy, performance, sustainability, and carbon footprint. The system provides smart scheduling, scaling, and migration of Serverless Runtimes, and scaling of Edge Clusters to meet workload requirements.

Tasks not completed:
Automated provisioning and deletion of Edge Clusters.

12.8. Secure and Trusted Execution of Computing Environments

SR6.1 Advanced Access Control

Status: COMPLETED

Completed Tasks:
The COGNIT Framework implements Biscuit token-based authorization with fine-grained permissions. Policy enforcement occurs at the Edge Cluster Frontend.

SR6.2 Confidential Computing

Status: COMPLETED

Completed Tasks:
The COGNIT Framework supports AMD SEV-SNP confidential computing. Application requirements can specify confidential computing requirements. The AI-Enabled Orchestrator filters Edge Clusters to CC-capable hosts when the requirement is specified. The Cloud-Edge Manager deploys Serverless Runtimes as SEV-SNP protected VMs.

SR6.3 Federated Learning

Status: NOT COMPLETED

13. Conclusions

This final deliverable (D2.6) provides a comprehensive description of the completed COGNIT Framework architecture. The COGNIT Framework delivers an AI-Enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that addresses the challenges of enabling resource-constrained edge devices to leverage heterogeneous infrastructure across the cloud-edge continuum for computationally intensive processing. The framework provides a solid foundation for edge computing applications requiring smart, adaptive, and sustainable resource management across multi-provider infrastructure.

The COGNIT Framework consists of five main components working together to provide smart, adaptive serverless computing:

- **Device Client** (Python and C) for lightweight offloading from devices.
- **COGNIT Frontend** as the main entry point for authentication and coordination.
- **Edge Clusters** providing semi-autonomous execution environments.
- **Cloud-Edge Manager** for infrastructure management.
- **AI-Enabled Orchestrator** for smart resource optimization.

The framework satisfies all User Requirements from the four Use Cases (Smart Cities, Wildfire Detection, Energy Management, Cybersecurity) and meets all global requirements for sovereignty, sustainability, interoperability, and security. All Software Requirements have been implemented and validated, with the exception of SR6.3 (Federated Learning).

The COGNIT Framework is built on European open source technologies, primarily OpenNebula for cloud-edge management, fulfilling the sovereignty requirement for strategic autonomy. The framework leverages and extends established open source frameworks (e.g. OpenNebula, OpenSUSE, MinIO, RabbitMQ), following the interoperability requirement. Edge Clusters provide an abstraction layer enabling workload portability without vendor lock-in, supporting deployment across heterogeneous infrastructure spanning public cloud providers, private clouds, and on-premises infrastructure.

The AI-Enabled Orchestrator implements machine learning models and multi-objective optimization algorithms that predict workload patterns and resource requirements, optimize placement for latency, energy, and performance, enable proactive scaling and resource provisioning, and continuously learn and adapt to changing conditions. This addresses Research Challenges 4 and 5 regarding portable workload execution and automatic adaptation.

The framework integrates energy monitoring, carbon intensity awareness, and energy-aware optimization algorithms. The AI-Enabled Orchestrator optimizes placements to maximize renewable energy usage and minimize carbon footprint, addressing sustainability requirements and Research Challenge 6.

The security architecture implements defense-in-depth with multiple layers of protection: decentralized authentication with Biscuit tokens, fine-grained authorization with

policy-based access control, confidential computing support, comprehensive monitoring and audit logging, and alignment with GDPR, NIS2, and CRA.

The COGNIT architecture is flexible and extensible, allowing future enhancements and adaptations to evolving requirements and technologies. The choice of European open source technologies ensures strategic autonomy and enables the European research and industrial ecosystem to maintain and extend the framework beyond the project lifecycle. The COGNIT Framework has been validated through four diverse Use Cases and is ready for adoption by organizations seeking to leverage cloud-edge resources for computationally intensive edge applications.