A Cognitive Serverless Framework for the Cloud-Edge Continuum

D2.1 COGNIT Framework - Architecture - a

Version 1.0

28 April 2023

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 intelligent 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 document defines the main components of the COGNIT Framework, identifies the main software requirements derived from the global and Use Cases requirements, describes the methodology and specific scenarios that are being employed for the verification of the innovative COGNIT functionalities, and provides an initial plan for both the instantiation of the COGNIT Architecture and the prioritisation of Software Requirements during the next research and innovation cycles 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:	31/03/2023
Actual Date of Delivery:	28/04/2023
Lead Author:	Marco Mancini (OpenNebula)
Authors:	Michael Abdou (OpenNebula), Monowar Bhuyan (UMU), Dominik Bocheński (Atende), Aritz Brosa (Ikerlan), Idoia de la Iglesia (Ikerlan), Sébastien Dupont (CETIC), Grzegorz Gil (Atende), Torsten Hallmann (SUSE), Joan Iglesias (ACISA), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Martxel Lasa (Ikerlan), Ignacio M. Llorente (OpenNebula), Alberto P. Martí (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Behnam Ojaghi (ACISA), Daniel Olsson (RISE), Holger Pfister (SUSE), Tomasz Piasecki (Atende), Francesco Renzi (Nature4.0), Kaja Swat (Phoenix), Paul Townend (UMU), Iván Valdés (Ikerlan), Thomas Ohlson Timoudas (RISE), Riccardo Valentini (Nature4.0), Constantino Vázquez (OpenNebula), Shuai Zhu (RISE).
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	14/04/2023	Draft	Initial Draft
0.2	21/04/2023	Peer-Reviewed	Reviewed Draft
1.0	28/04/2023	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.1	21/04/2023	Prof. Rubén S. Montero (OpenNebula/UCM)
0.1	21/04/2023	Mr Philippe Massonet (CETIC)

Summary of Changes from Previous Versions

First Version of the “COGNIT Framework - Architecture” Deliverable

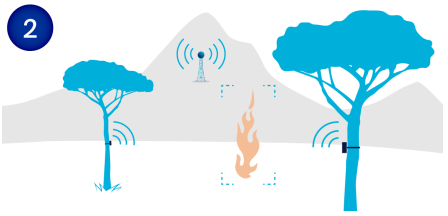
[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

This is the first version of Deliverable D2.1, the COGNIT Framework Architecture report in WP2 “Adaptive Cloud-Edge Serverless Framework Architecture”. It includes a summary of Use Cases requirements (T2.1), an analysis of sovereignty, sustainability, interoperability, and security requirements (T2.2), the design and architecture specifications of the COGNIT Framework (T2.3, T2.4) and the methodology for verification (T2.5).

The project’s Use Cases, described in full detail in Deliverable D5.1, will drive the development of the COGNIT Framework. The scientific and technological aspects of this project will always be driven by the evolving requirements of its users. This report describes the main components that will be considered to build an AI-Enabled Serverless Cloud-Edge Computing Platform, identifying the main software requirements derived from the analysis of the following Use Cases and their respective user requirements:

Smart Cities



Wildfire Detection

Energy

Cybersecurity

The main objectives of the COGNIT Framework are:

- Support a new innovative Serverless paradigm for edge application management, based on code offloading.
- Enable the on-demand deployment of large-scale, highly distributed and self-adaptive serverless environments using existing data processing resources from cloud/edge infrastructure providers, including local data centres, cloud providers, and 5G/telecom operators
- Optimise where data is processed according to changes in application demands and behaviour, and energy efficiency heuristics.

These are main components of the COGNIT Architecture:

- The **Device Client** allows the device to offload functions to any tier in the cloud-edge continuum, according to some requirements provided by the device itself; it will communicate with the Provisioning Engine in order to create Serverless Runtime for the execution of device application tasks.
- The **Serverless Runtime** is in charge of executing the functions offloaded by the devices and storing data uploaded by the devices or from external storage backends.
- The **Provisioning Engine** is responsible for managing the lifecycle of the Serverless Runtimes.
- The **Cloud-Edge Manager** is responsible for scheduling the Serverless Runtime according to the deployment plan provided by the AI-Enabled Orchestrator
- The **AI-Enabled Orchestrator** is the component that, according to the device requirements and infrastructure availability, will optimally schedule the Serverless Runtime on the cloud-edge continuum resources.

Finally, this document also includes the description of the methodology and the specific verification scenarios to be used for the validation of the project's new features and their applicability in each Use Case.

This analysis represents the starting point for Work Packages WP3 and WP4, which will specify, design, and develop the components of the COGNIT Framework in order to satisfy the list of global and user requirements defined in this document.

This deliverable has been released at the end of the start phase (M3), and will be updated with incremental releases at the end of each research and innovation cycle (i.e. M9, M15, M21, M27, M33).

Table of Contents

Abbreviations and Acronyms

ACL	Access Control List
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CC	Confidential Computing
CCA	Confidential Computing Architecture
CD	Continuous Delivery (Deployment)
CI	Continuous Integration
CIA	Confidentiality, Integrity and Availability
CRA	Cyber Resilient Act
DaaS	Data as a Service
DB	Database
FaaS	Function as a Service
FLOPS	FLoating point Operations Per Second
GPU	Graphics Processing Unit
GDPR	General Data Protection Regulation
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management system
IOPS	I/O Operations Per Second
IP	Internet Protocol
IoT	Internet of Things
JSON	Javascript Object Notation
LDAP	Lightweight Directory Access Protocol
ML	Machine Learning
NIS	Network and Information Security
NIST	National Institute of Standards and Technology
OIDC	OpenID Connect
OS	Operating System
QoS	Quality of Service
REST	Representational State Transfer
RBAC	Role-Based Access Control
S3	Simple Storage Service
SDK	Software Development Kit
SEV	Secure Encrypted Virtualization

SGX	Software Guard eXtension
SLA	Service Level Agreement
SQL	Structured Query Language
TEE	Trusted Execution Environments
TLS	Transport Layer Security
TPM	Trusted Platform Module
TPU	Tensor Processing Unit
VM	Virtual Machine
YAML	Yaml Ain’t a markup language

1. Introduction

The purpose of Deliverable D2.1 is to define the COGNIT AI-enabled serverless cloud-edge computing platform for each of the three planned product innovation iterations. To this end, D2.1 will be released at the beginning of each development cycle at M3, M9 and M15 with an incremental definition of Use Cases and requirements (T2.1), sovereignty, sustainability, interoperability, and security requirements (T2.2), design and specifications for distributed FaaS model for edge application development (T2.3) and cloud-edge serverless framework (T2.4), and verification suite (T2.5). A final version of the report will be released at the end of the last cycle at M21.

D2.1 is a living document that is composed of an introductory section and eleven additional sections organised in three main blocks of content:

- **Part I** focuses on the definition of global requirements for the COGNIT Framework, as well as on the user requirements extracted from the project's Use Cases. Section 2 identifies the global sovereignty, sustainability, interoperability, and security requirements, ensuring that the cloud-edge serverless architecture, its deployment, and its exploitation are in line with European policies and priorities, relevant EU projects, and open standards. Section 3, on the other hand, summarises the main user requirements of the Use Cases, described in full in Deliverable 5.1.
- **Part II** focuses on the definition of the software requirements of the COGNIT Framework. Section 4 defines the overall architecture of the COGNIT Framework, consisting of two main parts: the *Distributed Serverless Model for Edge Application Development* and the *Cognitive Cloud-Edge Module*, which are described in Section 5 and Section 6, respectively. Section 7 introduces and defines advanced mechanisms and techniques to build a secure and trusted execution environment in the cloud-edge continuum. Section 8 presents the functional gaps and requirements grouped into the main building components of the COGNIT Framework. Section 9 provides the matching between software and user requirements.
- **Part III** focuses on verification and implementation. Section 10 presents the verification methodology and a list of verification scenarios. Sections 11 identifies an initial set of technologies to be used in the instantiation of the COGNIT Architecture. Section 12 provides an initial prioritisation of Software Requirements to be targeted during the next research and innovation cycles of the Project.

The document ends with a conclusion section.

PART I. Global and User Requirements

2. Sovereignty, Sustainability, Interoperability, and Security Requirements

This section summarises the results of an initial 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, will be updated as part of each release of the Architecture report.

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 3.1 below provides the list of sovereignty requirements to be met during the execution of the project:

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 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"*; this challenge will be studied in particular as part of the Energy Use Case (UC3), where the COGNIT Framework will have to 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"*.

The following interoperability requirements target the ability of the COGNIT Framework to connect and integrate with existing infrastructures, services, hardware. This means that the framework should be aware 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 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 will be 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 future 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"*; and will be expanded 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 Zeto 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 (also during the live migration of FaaS Runtimes).	All

Table 2.4. Global Security Requirements.

3. Use Case Requirements

This section summarises the user requirements extracted from the Use Cases, as described in more detail in Deliverable D5.1:

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 an authorization prior to establishing any further interaction with COGNIT.	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 application shall have the ability to dynamically scale resources for offloading function execution to maximise exploitation of resources in shared environments, while avoid saturation or resources 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 function.	UC2
UR2.2	COGNIT Framework should be able to adapt to rare events with sudden peaks of FaaS requests, in which the offloaded function requires much heavier computations and more frequent execution than usual.	UC2
UR2.3	Possibility for devices to request 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 scale up/down resources for function execution due to changes in the user preferences.	UC3
UR3.3	The SDK for the Device Client shall have support for the C programming language.	UC3

Table 3.4. User requirements for UC3 (Energy).

Id	Description	Source
UR4.1	The Device Client 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 live migrate of data/runtime to different edge locations based on policy and location of function execution (e.g. geographic security zones, distance to edge node).	UC4
UR4.3	The Device should be able to request the execution of a function as close as possible (in terms of latency) to the Device's location.	UC4

Table 3.5. User requirements for UC4 (Cybersecurity).

PART II. Architecture Definition

4. COGNIT Framework

COGNIT will develop a new distributed Serverless paradigm which will change how applications are processed, both technically and conceptually, in the cloud-edge continuum, and an innovative AI-enabled adaptive framework for the cognitive cloud-edge continuum, which will provide applications with seamless, secure, and interoperable access to a continuous computing and data processing service environment that abstracts the large-scale, geo-distributed infrastructure provided by the cloud-edge continuum. The framework will enable application developers to offload tasks with dynamic execution requirements in terms of special hardware devices (e.g. GPUs), infrastructure capabilities and capacities, specific execution environments, communication patterns, cost performance, latency, security, and energy efficiency.

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, using fast computational units, special-purpose resources and services (e.g. GPUs, HPC, DB), and low-latency connections (i.e. 5G). Using code offloading allows computationally intensive data processing to be executed outside the edge devices, sensors, and actuators; this translates to more efficient power management, fewer storage requirements, and better application performance where devices may not provide the hardware acceleration capabilities for increasing the performance of the critical regions of computations.

Although code offloading has been widely considered to save energy and increase responsiveness of mobile devices, the technique still faces many challenges pertaining to practical usage, namely the complexity of its integration with the cloud management environment, the dynamic configuration of the system, its scalability, and the lack of Offloading-as-a-Service implementations. COGNIT addresses existing code offloading limitations by developing a new distributed Serverless model that, integrated with a Cognitive cloud-edge management platform, facilitates the development of elastic and scalable edge-based applications.

In order to provide end-users with seamless access to a continuous data processing environment, COGNIT allows application developers to easily integrate cloud-edge processing in their coding logic by using a new distributed Serverless computing model that allows them to offload data processing code fragments and tasks from the end-user system, device, sensor, or actuator to the cognitive continuum in order to speed up computation, save energy, save bandwidth, or provide low latency.

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers. It has been rapidly adopted by developers because it relieves them of the burden of provisioning, scaling, and operating the underlying infrastructure. Different FaaS services (AWS Lambda, Azure Functions, Google Cloud Functions), and open source frameworks (Apache OpenWhisk, Iron Functions, Fission, Kubeless, OpenFaaS) are

available in the market, but they they assume that the cloud platform runs on large, highly homogeneous datacenters with commodity infrastructure, and the functions have a small footprint and short execution duration.

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 that follows an asynchronous model
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 / Stateful
Maximum runtime	Short (e.g. <900 seconds)	None
Maximum capacity	Limited (e.g. < 3GiB memory)	None
Communication patterns	Workflows and state machines	Results forwarding to other functions
Deployment requirements	Basic capacity (e.g. memory) & quotas	Performance, cost, security, and energy
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 (GPU), services (AI)

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

4.2. COGNIT Execution Model

The execution model introduced in this section describes the workflow that COGNIT will follow in order to meet the requirements of being able to deliver part of the application computation and workload in the cloud-edge continuum.

The COGNIT Framework is based on the architecture described in Figure 4.1:

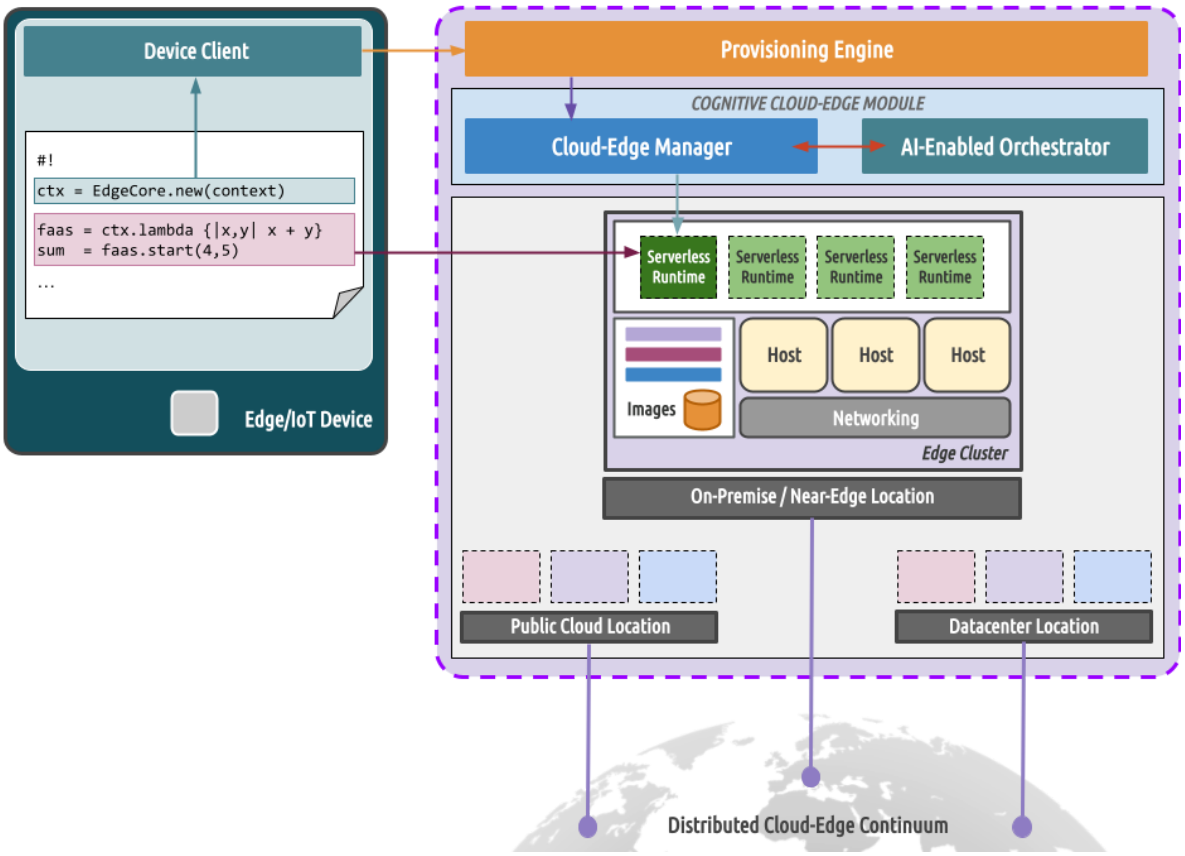


Figure 4.1. General view of the COGNIT Architecture.

The COGNIT Framework expands traditional serverless and FaaS solutions, where resources are hired just from a single cloud provider, considering infrastructure resources along the cloud-edge continuum, i.e. on-premise deployments, public cloud providers, edge providers. COGNIT provides a Serverless paradigm that allows applications and IoT devices to perform computations on any resource along the cloud-edge continuum hiding the unnecessary infrastructure management of a highly distributed and heterogeneous environment.

The architecture of the COGNIT Framework consists of five components as depicted in Figure 4.1. The first group of components is related to the **Distributed Serverless Model for Edge Application Development** that consists of the following:

- **Device Client**: SDK that allows to create Serverless runtime environments for offloading function executions on the cloud-edge continuum and then perform different tasks such as function execution, transfer data from external sources and upload data from the device itself.
- **Serverless Runtime**: main unit management component of the COGNIT Framework that allows the execution of device/application functions, upload and transfer data in a hardened and secure environment.

- **Provisioning Engine**: responsible for managing the lifecycle of the Serverless Runtimes; it can create, delete and update the Serverless Runtimes according to the operations that are sent by the Device.

The final two architectural components are grouped together as part of the **Cognitive Cloud-Edge Module** that allows the management of the cloud-edge continuum resources in an intelligent and adaptive way

- **Cloud-Edge Manager**: responsible for managing the highly distributed and heterogeneous resources of the cloud-edge continuum in order to schedule Serverless Runtimes according to the device/applications requirements.
- **AI-Enabled Orchestrator**: in charge of producing an initial deployment plan according to the device requirements. It will update the deployment plan in case of updated requirements sent by the device, but it will also dynamically optimise the Serverless Runtime placement, taking into account the monitoring of the resources (e.g. VMs, networks, etc.) used by the Serverless Runtime and the application metrics (e.g. application workload) sent by the Serverless Runtime.

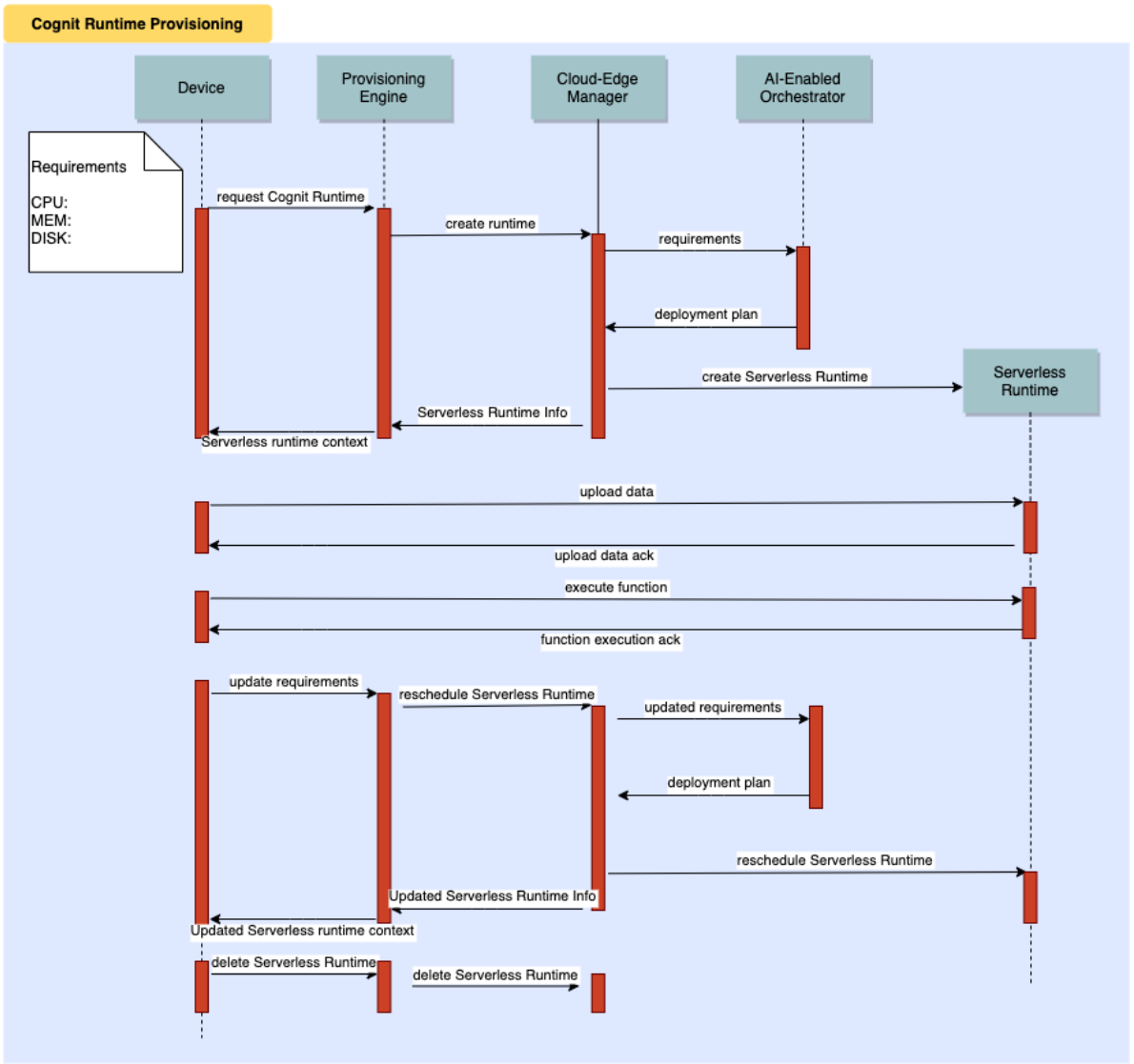


Figure 4.2. COGNIT components sequence diagram.

In Figure 4.2 a sequence diagram is reported to show the communications between the different components. Different phases can be identified in the diagram:

1) The device will request a Serverless Runtime to the Provisioning Engine that satisfies some requirements (e.g. CPU/memory needs, storage, latency, etc.).
2) The Provisioning Engine will communicate with the Cloud-Edge Manager, which according to the device initial requirements will deploy the Serverless Runtime.
3) The device polls the Provisioning Engine until the Serverless Runtime is created and ready.
4) Once the Serverless Runtime is ready, the device can start its operations such as offloading execution of tasks and functions and data upload/transfer.
5) The device can send updated requirements to the Provisioning Engine according to dynamical changes in the edge application.
6) The device can ask to delete the Serverless Runtime once the application does not need anymore to offload tasks to the cloud-edge infrastructure.

The next sections describe in detail the different components of the COGNIT Framework.

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

- **Device Client**
- **Serverless Runtime**
- **Provisioning Engine**

5.1. Device Client

The Device Client establishes a communication with the Provisioning Engine to request offloading of certain operations, such as running a function or data transfer to be executed on Serverless Runtime(s) on the distributed cloud-edge continuum. The client will be able to set certain requirements, such as latency, cost or energy consumption, to let the COGNIT Framework decide how and where to run the offloaded task:

Category	Requirement
Execution Context	Capacity (i.e. CPU, Memory, Disk)
	Network features
	Specific processing infrastructure (e.g. Cloud, HPC)
	Special Devices (e.g. GPU, TPM)
Runtime Flavour	Application-specific OS (e.g. openSUSE)
	Libraries and packages needed for function execution (e.g. PyTorch)
	Data service type (e.g. MinIO, MariaDB)
Communication Patterns	Parallel processing
	Map-Reduce
Deployment	Performance and Cost
	Security
	Latency
	Energy Consumption

Table 5.1. Device Requirements.

The device that is hosting the Device Client will need to be able to implement a HTTP client, as the requests will be forwarded in the form of an HTTP API.

The Device Client will have multilingual support, starting by covering Python and C as main languages, for enabling the Use Cases presented in this project, but opened to expand to other languages in the future if it is required. Clearly the considerations to offload tasks differ remarkably if the targeted language is an interpreted or a compiled one, so being the two starting languages a good example of each kind of language, they can be considered as standards for future expansions.

The libraries needed for the offloaded code to be executed in the correct location need to be properly preprocessed, so the code execution is flawless at the right time, therefore the client will need to handle this preprocessing.

The client will also be able to define if the offload is done in a synchronous way or asynchronous, and if the latter is the case, a polling mechanism will tell the client when the offloaded task is finished within the COGNIT Framework.

Moreover, a caching mechanism will be implemented in the COGNIT Framework, so the different functions can be registered for a more streamlined execution of recurrently occurring offload of tasks' situation.

The communication between the Device Client and the COGNIT Framework will be encrypted, and an authorisation mechanism based on certificates will be put in place, in such a way that authenticated and confidentiality preserving exchange will assure a secure entry point towards the COGNIT system from the device.

The Device Client will be implemented as an SDK according to the language supported by the device (either Python or C) and will provide the following methods:

Operation	Definition
`Create`	Allows the device to Initiate a session for offloading functions and data transfer. It returns to the device the context information of the Serverless Runtime created by the Provisioning Engine, according to the requirements passed as a parameter through the initial operation.
`Read`	Allows the device to read the context of the Serverless Runtime.
`Upload`	Allows the device to upload data to the Serverless Runtime.
`Copy`	Allows the data transfer from an external storage (e.g. an S3 cloud storage) to the Serverless Runtime.
`Execute`	Allows the device to offload computations on the Serverless Runtime.
`Update`	Allows the device to update the requirements of the Serverless Runtime.
`Delete`	Allows the device to terminate the session by asking the Provisioning Engine to delete the Serverless Runtime.

Table 5.2. Methods supported by the Device Client SDK.

As an example of the Device Client SDK, a Python-like code is reported in Table 5.3 corresponding to the Energy Use Case of charging a vehicle:

```
import cognit
requirements = { ‘CPU’: ‘2’, MEM: ‘4GB’, HW_constraints: ‘GPU’ }

def inference(smart_meter_data, weather_data, model_file):
      model.load(model_file)
      model.evaluate()
      return model.inference(weather_data, smart_meter_data)

ctx = cognit.create(requirements)

ctx.copy('http://path/to/bucket/weather_prediction.csv',faas:'/data/wea
ther_prediction.csv')

ctx.copy('http://path/to/bucket/model.torch', faas:'/data/model.torch')

charge_complete = False
while not charge_complete:
     smart_meter_data = device.get_data()
      prediction = ctx.FunctionEvaluate(inference, smart_meter_data,
'/data/weather_prediction.csv', '/data/model.torch')
     charge_complete = device.check_charge(prediction)
      sleep(10)
ctx.delete()
```

Table 5.3. Example of a Python-like code for the Device Client.

5.2. Serverless Runtime

Serverless Runtime is the main management unit of the COGNIT Framework. It is defined by a document (e.g. YAML file) that conveys all the information for its automatic deployment on the distributed cloud-edge continuum. The document containing the requirements for the Runtime is sent by using the Device Client to the Provisioning Engine.

The Serverless Runtime consists of two main components:

- A Function as a Service (FaaS) Runtime component that allows the execution of functions on resources of the cloud-edge continuum.
- A Data as a Service (DaaS) component that allows uploading data to the Serverless Runtime exploiting data locality. The DaaS can implement different protocols and backend storages (e.g. S3, SQL DB, etc.).

The Serverless Runtime is created and scheduled according to the requirements provided by the device that can be related to different constraints, as reported in Table 5.1. The Serverless Runtime provides different operations that are exposed through a REST API:

Operation	Definition
`Execute`	Allows the execution of a function.
`Upload`	Uploads data from the device into the Serverless Runtime.
`Copy`	Copies data from external data sources into the Serverless Runtime.

Table 5.4. Methods supported by the Serverless Runtime REST API.

The Serverless Runtime will push metrics, related to the performance, of the FaaS Runtime (e.g. average execution time, number of executions per second) and DaaS Runtime (e.g. IOPS, available free capacity) to the Cloud-Edge Manager so that it can be pulled and taken into account by the AI-Enabled Orchestrator.

5.3. Provisioning Engine

The Provisioning Engine is responsible for managing the lifecycle of the Serverless Runtimes, and will provide the following operations:

Operation	Definition
`Create`	Creates a new Serverless Runtime according to the initial requirements sent by the device.
`Read`	Reads the context of the Serverless Runtime.
`Delete`	Deletes the Serverless Runtime
`Update`	Updates the requirements of the Serverless Runtime

Table 5.5. Operations supported by the Provisioning Engine.

The Provisioning Engine exposes a REST API that is used by the Device Client to manage (create/read/delete/update) Serverless Runtimes.

The Provisioning Engine, once receives a request from the device, translates the requirements (see Table 5.1) sent by the device into a new document (e.g. a YAML or JSON file) divided in two main parts as reported in Table 5.6. and then it sends this document to the Cloud-Edge Manager:

Category	Requirement
Infrastructure Template	Number of CPUs/cores and FLOPS

	Memory Size and Bandwidth
	Storage size and IOPS (for the DaaS)
	Network bandwidth and latency
	Hardware Specifics (e.g. GPU, TPM)
	Energy constraint
	Security constraints
	Cost constraints
Serverless Template	One VM/container image ID (related to a VM appliance marketplace or a container registry) of the FaaS Runtime - containing specific OS and libraries for executing functions sent by the device.
	One or more VM/container images ID(s) (related to VM appliance marketplace or a container registry) of the DaaS Runtimes - containing applications for storing data such as a database or an object storage.

Table 5.6. Serverless Runtime Requirements.

The creation request of a new Serverless Runtime will be an asynchronous process, so the Device client will not be blocked during the request; the Provisioning Engine will return a unique Serverless Runtime ID (received from the Cognitive Cloud-Edge Module) that the Device client can use to poll the Provisioning Engine (at intervals that are set by the device itself) in order to know the status of the request. Once the request is completed, the Device can perform a read using the Serverless Runtime ID to get the information of the Serverless Runtime, such as the IP to connect, that can then be used to perform execution of functions or data operations.

The Provisioning Engine will establish secure communication channels both with the Device and with the Cognitive Cloud-Edge Module, and will provide a mechanism to authenticate the Device (see Section 6.1 for more details).

6. Cognitive Cloud-Edge Module

The Cognitive Cloud-Edge Module, in line with the Project's Research Challenge 4—*"Portable and adaptive execution of serverless workloads across a multi-provider cloud-edge continuum"* and Research Challenge 5—*"Automatic and intelligent adaptation of the cloud-edge continuum to the changing demands of the applications"*—will decide about the placement and mobility of Serverless Runtimes across the cloud-edge continuum and about the elasticity measures to be applied to the cloud-edge infrastructure, all that in order to respond to the application requirements sent by the edge devices and the need to proactively optimise placements, costs, and other utility functions (e.g. energy consumption) according to the dynamic changes in the conditions and requirements of the application or to those unexpected incidents affecting the underlying infrastructure.

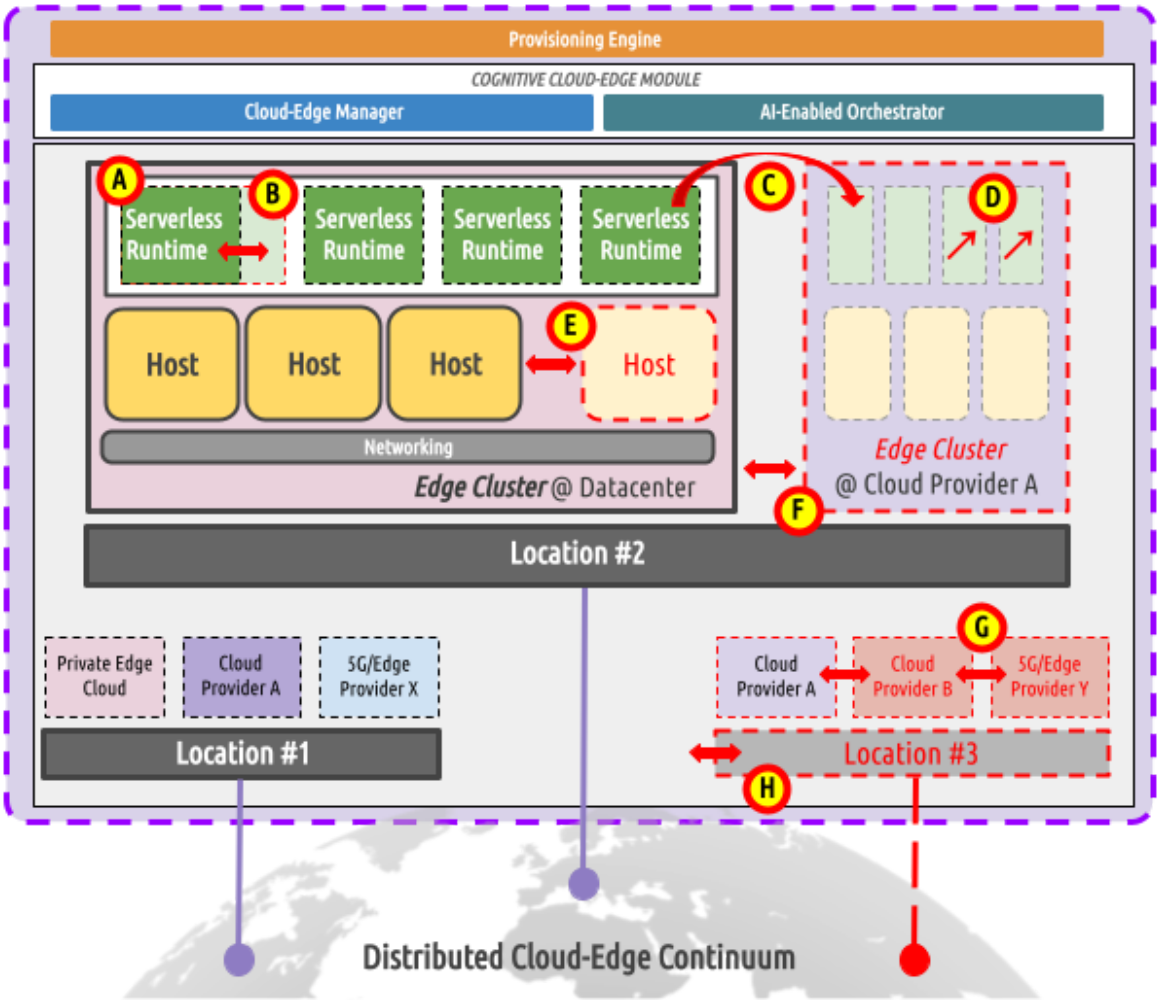


Figure 6.1. Workload and cloud-edge infrastructure adaptability supported by COGNIT.

The following table describes in more detail how the COGNIT Framework will deal with the adaptability required for the automatic placement and mobility of the Serverless Runtimes and for leveraging in an intelligent and optimal way the elasticity capabilities that the infrastructure across the cloud-edge continuum provides:

Feature	Description	Activation
(A) Optimal Serverless Runtime Placement	Instantiate a Serverless Runtime on an Edge Cluster able to meet its current requirements.	✔ **Reactive**: the edge application triggers the instantiation or update of a Serverless Runtime with a specific set of characteristics.
(B) Serverless Runtime Vertical Elasticity	Scale up/down the resources associated with the workloads (e.g. VMs) hosting the Serverless Runtime.	✔ **Reactive**: due to event-driven elasticity rules or changes in the Serverless Runtime requirements according to new specifications by the edge application. ✔ **Proactive**: due to AI-driven forecasting techniques.
(C) Serverless Runtime Mobility	Live migrate the Serverless Runtime to a different Edge Cluster.	✔ **Reactive**: due to event-driven mobility rules or changes in the Serverless Runtime requirements according to new specifications by the edge application. ✔ **Proactive**: due to AI-driven forecasting techniques.
(D) Optimal Global Workload Placement	Optimise the placement of existing Serverless Runtimes.	✔ **Proactive**: due to AI-driven optimisation techniques (e.g. reduce overall energy consumption).
(E) Infrastructure Cluster Elasticity	Scale up/down existing Edge Clusters by activating/deactivating hosts (i.e. servers).	✔ **Reactive**: due to event-driven elasticity rules or changes in the Serverless Runtime requirements according to new specifications by the edge application. ✔ **Proactive**: due to AI-driven forecasting techniques.
(F) Infrastructure Horizontal Elasticity	Increase the number of available Edge Clusters in a given location.	✔ **Reactive**: due to event-driven elasticity rules or changes in the Serverless Runtime requirements according to new specifications by the edge application. ✔ **Proactive**: due to AI-driven forecasting techniques.
(G) Additional Cloud/Edge Provider	Increase the number of available cloud/edge service providers offering infrastructure resources.	✔ **Reactive**: due to event-driven provisioning rules or changes in the Serverless Runtime requirements according to new specifications by the edge application.

(H) Additional Cloud/Edge Locations	Incorporate new geographical locations to the cloud-edge continuum.	✔ **Reactive**: due to event-driven provisioning rules or changes in the Serverless Runtime requirements according to new specifications by the edge application.

Table 6.1. COGNIT reactive and proactive adaptability features.

In order to respond to those adaptability requirements, the Cognitive Cloud-Edge Module (as shown below in Figure 6.2) will be based on two main components:

- The **Cloud-Edge Manager** is in charge of managing the cloud-edge continuum infrastructure and of performing actions to manage the lifecycle of the different Serverless Runtimes, collecting their metrics and monitoring the infrastructure resources used by the Serverless Runtimes.
- The **AI-Enabled Orchestrator** is responsible for optimising the placement of the Serverless Runtime using data-driven approaches based on the metrics collected by the Cloud-Edge-Manager. It checks periodically the status of the Serverless Runtimes created by the Provisioning Engine. It interacts appropriately with the Cloud-Edge Manager in order to schedule Serverless Runtimes on the cloud-edge continuum according to the device requirements. Then it will update the Serverless Runtime placement, according to the dynamic changes in the application requirements and to changes in the infrastructure, client device mobility, etc.

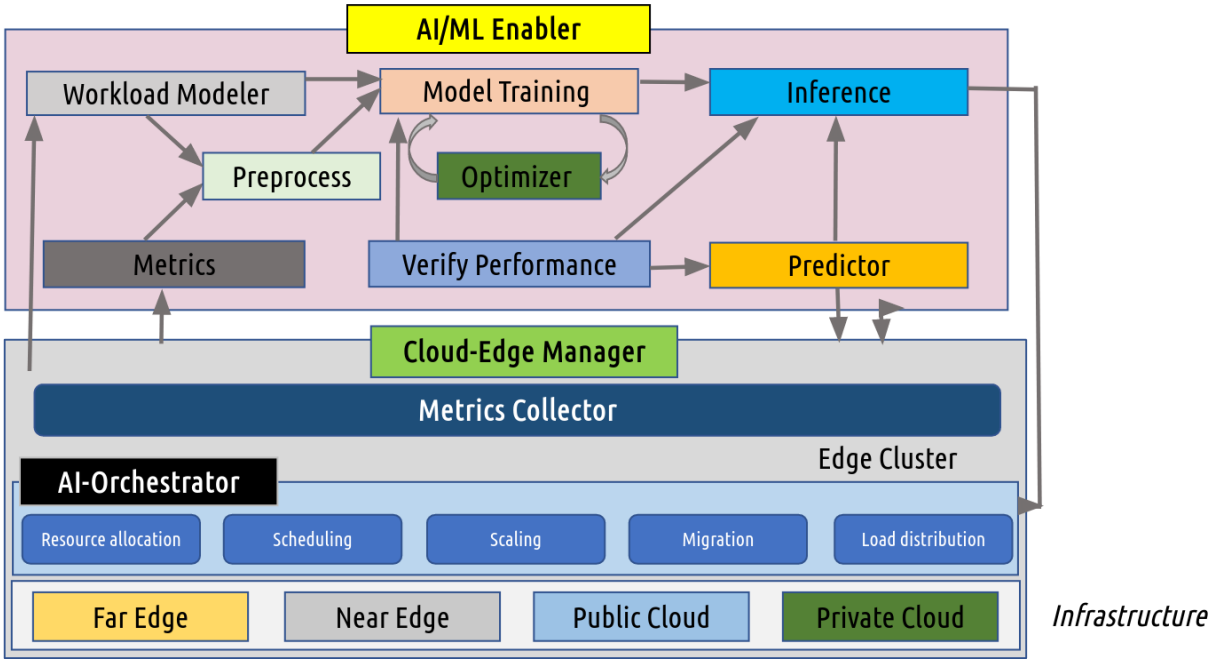


Figure 6.2. High-level AI/ML orchestration model.

As shown in Figure 6.2, the AI-Enabled Orchestrator obtains monitored metrics from the Cloud-Edge manager through a *metrics collector* and these metrics will be utilised to train

AI/ML models for smart placement and relocation of workloads across a distributed cloud-edge continuum based on a diverse and wide range of workloads. Each layer is explained below:

Device client/applications. As discussed in Section 5, each client interacts with the Serverless Runtime for running a function or data transfer across the distributed cloud-edge. Workloads will be classified based on unique features of resource usage patterns as s*hort-living*, *long-living*, *compute-intense*, *data-intense* or *memory-intensive*. Furthermore, each workload may be classified based on multi-dimensional requirements of resources (e.g. CPU, disk, network, memory) and Quality of Service (QoS) requirements (e.g. time constraints, throughput, latency). The extremely dynamic and unpredictable feature of heterogeneous workloads greatly increases the complexity of orchestration mechanisms.

Edge Cluster. An Edge Cluster is a semi-autonomous complete environment with a dedicated software-defined resource set (compute, network and storage). The AI-Enabled orchestrator is responsible for assigning Serverless Runtimes to nodes where runtimes are actually hosted and executed. The major functionalities involve:

- *Resource Allocation* assigns a specific amount of resources to each runtime. Such configurations and limitations are basic metrics in managing Serverless Runtime placement and resource isolation control between runtimes.
- *Scheduling* defines the policy for the initial placement for one or a group of Serverless Runtimes, by considering conditions such as resource constraints, communication costs, QoS requirements, and SLAs.
- *Scaling* is the resource configuration adjustment of Serverless Runtimes or compute nodes in response to any potential workload fluctuations.
- *Migration* is designed as a complementary mechanism to prevent severe infrastructure-level resource overloading or resource contention between co-located serverless runtimes by relocating one or a group of runtimes from one node to another.
- *Load Distribution* plans to distribute the network traffic among runtimes evenly with a policy like RoundRobin. It will improve system scalability, availability, and network performance.

AI/ML Enabler. The AI/ML enabler will build ML models for workload characterization, performance analysis, and adaptive deployment of Serverless Runtimes across the cloud-edge continuum based on analysis of monitoring data and system logs received from the Orchestrator. Furthermore, it will make future resource provisioning decisions based on the generated behavioural models and prediction results. AI/ML enablers can be entirely integrated or independent from the Orchestrator or adhere to the Orchestrator. The major components are:

- *Workload Modeler* is designed for ML-based workload characterization through analysing FaaS workloads and identifying their key characteristics.

- *Performance Verifier* generates application and system behaviour models through applying ML algorithms to application and infrastructure-level monitoring data acquired from the Orchestrator.
- *Predictor* forms forecasts of workload volumes or application/system behaviours, relying on the models obtained from Workload Modeler and Performance Verifier. The prediction results can be sent either to the Orchestrator or Inference engine.
- *Inference engine* combines the behaviour models and prediction results received from the above components with multiobjective optimization methods/schemes to further generate precise resource provisioning decisions that are fed back to the Orchestrator. The multiobjective optimization will take processed input from the metrics (Edge Clusters, FaaS, DaaS) to prioritise learning features that enable fine-grained models as well as inference after.

6.1. Cloud-Edge Manager

The main responsibilities of the Cloud-Edge Manager are:

- Exposing through an API the operations for managing the cloud-edge continuum infrastructure (i.e. physical computational hosts, networks and storages across multi-cloud providers and edge locations) and managing the Serverless Runtimes, that are used by the AI-Enabled orchestrator to optimise the execution of the applications based on the requirements sent by the device.
- Monitoring both the cloud-edge infrastructure and the Serverless Runtimes to provide the AI-Enabled Orchestrator with information to implement automatic and intelligent adaptation for the placement of the Serverless Runtimes .
- Providing authentication and authorization mechanisms for accessing and securing resources such as physical hosts, virtual machines, networks, services, etc.

Cloud-Edge Manager provisions resources across the cloud-edge continuum and offers a simplified, efficient way to deploy Serverless Runtimes (as virtualized workloads - VMs/micro-VMs/containers), according to the device/application requirements. The Cloud-Edge Manager is responsible for coordinating the use of a set of highly heterogeneous and distributed locations, providing an uniform view of the underlying resources, in such a way that the Serverless Runtimes can be deployed anywhere in the cloud-edge infrastructure without performing any additional configuration or setup.

Provider Catalogue

The Cloud-Edge Manager maintains a catalogue that contains a list of resource providers available in the cloud-edge continuum. The catalogue allows the AI-Enabled Orchestrator to select which providers/ locations are better suited according to the device requirements (in terms of cost, capacity, energy, latency, bandwidth, etc).

Each provider has an entry in the catalogue with:

- Unique ID
- List of locations, each location with a list of instance types
- Each instance type has
    - Name

    - Capacity
    - Price per hour
    - Price per megabyte (outbound and inbound)
    - Energy consumption per hour
- Link to the Provider Driver (custom adapter that uses the provider API)
- Additional characteristics:
    - Public IPs
    - GPUs
    - Backend storage types

The Provider Catalogue will contain entries related to public, private cloud or self hosted infrastructures hosted in the European Union. A set of filters (i.e. for latency, cost, energy, specific characteristics) are available to the AI-Enabled Orchestrator to select the appropriate provider according to the requirements provided by the device.

Edge Cluster Management

The main management unit of the Cloud-Edge Manager where to provision Serverless Runtime is the **Edge Cluster**. An Edge Cluster is a semi-autonomous complete environment with a dedicated software-defined resource set (compute, network and storage) able to work on unreliable infrastructures. Given the heterogeneity of cloud-edge systems, the Edge Cluster creates an abstraction layer enabling interoperability across the multi-provider cloud-edge infrastructure. Edge Clusters can be provisioned anywhere in the cloud-edge (on-premises, on public cloud and edge providers), but we assume that each Edge Cluster is defined only for a specific provider of the cloud-edge - a provider can be a public cloud provider, an edge local provider, a set of far edge devices, etc.

The Cloud-Edge Manager exposes an API that provides the following operations to the AI-Enabled Orchestrator for the management of Edge Clusters:

Operation	Definition
`Create`	Creates a new Edge Cluster on one of the locations available in the Catalogue.
`Delete`	Deletes an Edge Cluster
`List`	Lists existing Edge Clusters and return details for each of them
`Configure`	Configures services on the Edge Cluster's hosts.
`Scale`	Scales up/down the size of an existing Edge Cluster (i.e. add or remove hosts)

Table 6.2. Edge Cluster management operations provided by the Cloud-Edge Manager.

The provision of a new Edge Cluster is controlled by a document (i.e. a YAML or a JSON file) that describes in a declarative way the resources of the Edge Cluster associated with a specific provider (among the ones available in the Provider Catalog). Every time a new

Edge Cluster is created a unique id is associated with it. This document is created by the AI-Enabled Orchestrator according to the specifications related to the Infrastructure section as described in Table 5.6.

The Cloud-Edge Manager according to the document specifications will use a **Provider Driver** (a custom adapter for the provider API) that will automatically create the necessary resources for the Edge Cluster and will configure the different created entities (physical hosts, networks and storage) to provide software-defined components:

- Hypervisors for the deployment of workloads (i.e. Serverless Runtime) as VMs, microVMs or containers.
- Software-defined storage for storing the Serverless Runtimes images and for providing data blocks to the DaaS (e.g. DBs, object storage, …)
- Software-defined networks (i.e. virtual/overlay networks) for providing the communication of the Serverless Runtimes within the Edge Cluster and for the communication with the public Internet (i.e. the edge devices)

The provision document provided by the AI-Enabled Orchestrator contains the following information:

- Provider ID
- Resource specifications related to physical hosts, networks and storages
- Setup and configuration of the software-defined components:
    - Hosts configuration with components such as the hypervisor technology (VMs, microVMs, containers)
    - Datastores
    - Overlay networks

Serverless Runtime Management

The Cloud-Edge Manager exposes an API that allows the following operations related to the deployment of Serverless Runtime on Edge Clusters:

Operation	Definition
`Create`	Creates a new Serverless Runtime on an Edge Cluster identified by its ID.
`Check`	Checks the status of the Serverless Runtime (i.e. PENDING, DEPLOYING, RUNNING, SCALING, …)
`Update`	Update the Serverless Runtime deployment (e.g.. scheduling on another Edge Cluster)
`Scale`	Scales the FaaS/DaaS components of the Serverless Runtime (e.g. increase number of CPUs)
`Delete`	Deletes a Serverless Runtime

Table 6.3. Methods for Serverless Runtime management provided by the Cloud-Edge Manager.

The deployment of a new Serverless Runtime is controlled by a document (e.g. a YAML or JSON file) that contains the Serverless Runtime Template (i.e. the FaaS and DaaS runtime specs - image, resource requested, number of replicas, etc.). The Provisioning Engine will use the Serverless Runtime Template to issue the creation of a new Serverless Runtime that will be set to a *PENDING* state. The AI-Enabled Orchestrator will place the Serverless Runtime according to the requirements set in the Infrastructure Template (see Table 5.6).

Monitoring, Scaling, and Migration

The Serverless Runtime placement can be updated by the AI-Enabled Orchestrator according to the metrics that are collected by the Cloud-Edge Manager. The metrics are related to both the Edge Clusters (i.e. hosts, network, storage, software-defined components) and the Serverless Runtimes (application-defined metrics, e.g. average execution time of a function, DB IOPS, etc.). According to the information collected, the AI-Enabled Orchestrator can interact with the Cloud-Edge Manager to update the placement of the Serverless Runtime according to the following scenarios:

- Scaling up/down the Edge Cluster resources.
- Scaling up/down the Serverless Runtime.
- Migrating the Serverless Runtime from one host to another, or even from one Edge Cluster to another.

Also, the Cloud-Edge Manager should provide mechanisms in order to ensure a secure and trusted environment for the Serverless Runtimes, such as intrusion and anomaly detection. More details about secure and trusted execution is reported in Section 9.

As a requirement for the first research phases of the Project (until M15), we assume that a set of Edge Clusters are predefined, configured and provisioned according to the Use Case requirements (see Section 3). The Edge Clusters for the first phase are described in Deliverable D5.1, Section 8 ("*Summary of Use Case Cloud-Edge Infrastructure*"). Once the Edge Clusters are provisioned, the AI-Enabled orchestrator is responsible only with the scheduling/deployment, dynamic replacement and scaling of the Serverless Runtime across the different Edge Clusters available. In subsequents iterations of the project, this constraint will be relaxed and Edge Cluster provision configurations could be dynamically changed by the AI-enabled Orchestrator according to the dynamic changes in the requirements of the application.

Authentication and Authorization

The Cloud-Edge Manager provides an authentication system based on username and password, where information and secrets are stored in the Cloud-Edge Manager itself. Dedicated external user authentication drivers can be used to leverage additional authentication mechanisms or sources of information about the users (e.g. LDAP, OIDC). Users can belong to groups that are authorization boundaries for the users.

The Cloud-Edge manager provides an ACL authorization system that enables fine-tuning of the allowed operations for any user, or group of users. Each operation generates an authorization request that is checked against the registered set of ACL rules. The

Cloud-Edge Manager then can grant permission, or reject the request. This allows the system to tailor the user roles according to their cloud-edge infrastructure needs.

Each user entity can represent single or multiple devices that need to use cloud-edge resources to offload workloads. The devices are authenticated by the Provisioning Engine through a special mechanism as described in the following.

The Cloud-Edge Manager provides a mechanism to delegate the authentication process to the Provisioning Engine, that can use this to authenticate the requests from the devices and then forward the requested operations (e.g. create a Serverless Runtime) to the Cloud-Edge Manager. When a device interacts with the Provisioning Engine, it authenticates the request through the Cloud-Edge Manager (this step requires a user registered in the Cloud-Edge Manager associated with the device) and then it forwards the requested operation to the Cloud-Edge Manager. The forwarded requests between the Provisioning Engine and the Cloud-Edge Manager include the original user identifier, and are signed with the credentials of a special user.

The Provisioning Engine communicates with the Cloud-Edge Manager using a special user. This special user uses an authorization mechanism that allows the Provisioning Engine to perform an operation (i.e. creating a Serverless Runtime) on behalf of another user. In order to strengthen the security of the requests between the Provisioning Engine and the Cloud-Edge Manager, certificate-based authentication can be used. This provides mutual authentication using asymmetric cryptography techniques such as TLS.

Advanced mechanisms for authorization are described in Section 7.2.

6.2. AI-Enabled Orchestrator

The AI-Enabled Orchestrator is responsible for the placement of the Serverless runtime using AI/ML approaches according to the requirements specified by the devices.

The AI-Enabled Orchestrator periodically checks the status of the Serverless Runtimes that are issued by the Provisioning Engine and considering the device requirements, it will proceed to optimally place the Serverless Runtime following two phases.

- In the first phase, the AI-Enabled Orchestrator, according to the information related to the infrastructure category (see Table 5.6) selects the Provider and checks if an Edge Cluster with enough resources is already provisioned on it. Otherwise, it issues the creation of new Edge Clusters by providing the Cloud-Edge Manager with a provision document related to the Provider selected and with information to match the device requirements.
- In the second phase, the AI-Enabled Orchestrator updates the schedule of the Serverless Runtime on the Edge Cluster previously selected by invoking the corresponding method provided by the Cloud-Edge Manager.

The AI-Enabled Orchestrator is a proactive component that will dynamically optimise Serverless Runtime placement, taking into account changes in requirements sent by devices, the monitoring of resources (e.g. VM/microVM/container, storage, networks, etc.) used by the Serverless Runtime, and application metrics sent by the Serverless Runtime.

The AI-Enabled Orchestrator will utilise these metrics for building learning models (e.g. self-supervised learning, contrastive learning) that adapt to changes in Serverless Runtime requests from devices and changes in resources for smart placement across the distributed cloud-edge continuum. Furthermore, the orchestrator will optimise deployment of Serverless Runtimes using multiobjective optimization (e.g. NSGA-III) techniques.

Each time the AI-Enabled Orchestrator produces a new and smart deployment plan, it can issue an action for the Cloud-Edge Manager to reschedule the Serverless Runtime on different resources in the cloud-edge continuum.

The AI-Enabled Orchestrator monitors the states of the Serverless Runtime to keep it in a healthy state (through appropriate probes) and interacts with the Cloud-Edge Manager to get information about its state; based on this, it will interact with the Cloud-Edge Manager to take actions to bring it to a healthy state (e.g. through reboot or rescheduling).

AI/ML Methods

With increasingly diverse and dynamic cloud workloads processed by existing cloud service providers, it is still unclear how to automate the orchestration process for complex heterogeneous workloads under large-scale cloud computing systems. An important consideration is that serverless computing allows developers to write highly scalable, event-driven applications as a set of short-lived functions; this results in an idle cost (from a user perspective) of zero. In COGNIT, Serverless Runtimes will be orchestrated across distributed cloud-edge continuum using AI/ML methods to ensure efficient and automated workload orchestration - this has been highly successful in multiple applications such as image recognition, self-driving cars, recommendation systems, chatbots, etc.[2]

These successes are possible due to having large datasets combined with continuous improvement of computational power such as GPUs and TPUs. Machine learning (ML) algorithms are classified as supervised, unsupervised and reinforcement learning. *Supervised learning* refers to constructing models given a collection of training instances x_1, x_2, ... x_k and the corresponding response variable y, whereas in *unsupervised learning* there exist only predictors, hence the algorithms have to learn the structure of the training data.

Reinforcement learning solutions learn through trial-and-error, i.e. an agent learns to perform actions in an environment by interacting with it and receiving feedback regarding the performed actions. In contrast with many forms of machine learning, the learner is not told which actions to take, but instead, must discover which actions yield the most reward by trying them out (unsupervised learning). The goal of the agent is to maximise its cumulative reward, also referred to as expected return. Different reinforcement learning methods yield distinct behaviours for the agent to achieve their goal. Such solutions have drawn attention to automating and optimising workload placement across distributed cloud-edge continuum for serverless runtimes. However, when the goal is to predict a continuous or quantitative output value, the corresponding problem to be solved is called

[2] Tahseen Khan, Wenhong Tian, Guangyao Zhou, Shashikant Ilager, Mingming Gong, Rajkumar Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions", *Journal of Network and Computer Applications*, Volume 204, 2022, ISSN 1084-8045: https://doi.org/10.1016/j.jnca.2022.103405

regression, whereas the prediction of a categorical or qualitative output is known as a *classification problem*.

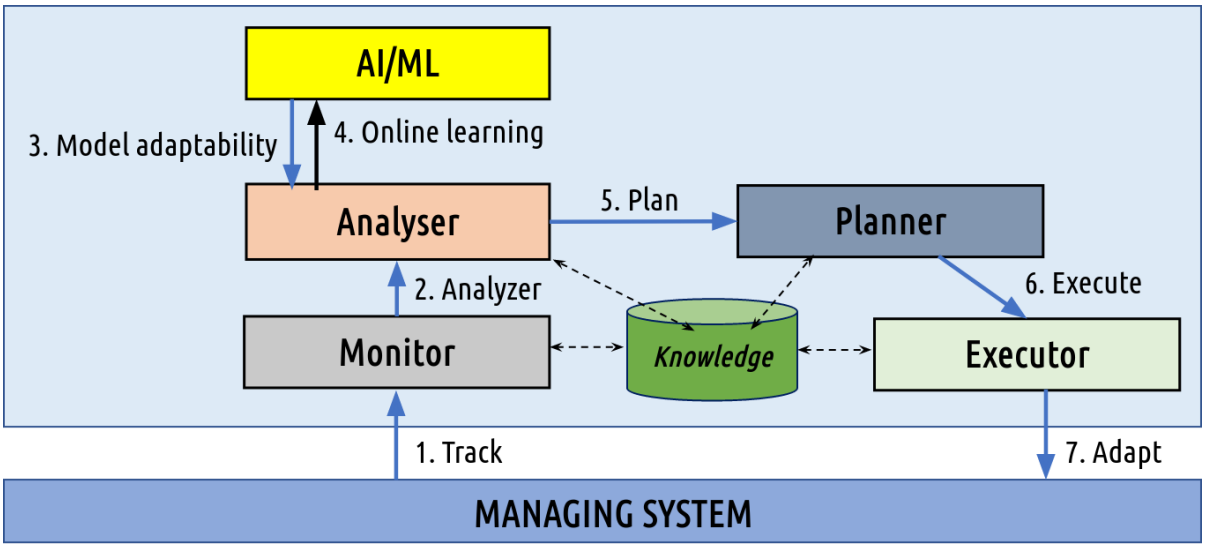


Figure 6.3. Typical steps in MAPE-k from monitoring to execution.

Machine learning methods can be parametric, where certain assumptions are made about the functional form of the model and training data is then used to fit its parameters, e.g. as in polynomial regression, or non-parametric, e.g. neural networks. Machine learning can also be used for inference tasks, i.e. to understand how the response variable is affected when the predictors change. A typical MAPE-k loop is given in Figure 6.3, which is primarily used to develop and deploy a learning model in computer systems for resource provisioning. It comprises seven steps: start from track to adapt, what is needed to accomplish a task.

Monitoring, Training, and Deployment

The development of a machine learning model is a complex and iterative process. A typical ML workflow or pipeline is given below in Figure 6.4. This workflow comprises four key elements 1) data management, 2) model learning, 3) model verification and 4) model deployment. The *Data management* module will collect and pre-process the data to make it ready for model training. *Model training* will learn the feature representation of data and optimise until a model suitable for a task is obtained. *Model verification* essentially investigates the performance of the model. Finally *model deployment* aims to deploy the trained and fine-tuned model for accomplishing certain tasks and to later update, maintain, and integrate again.

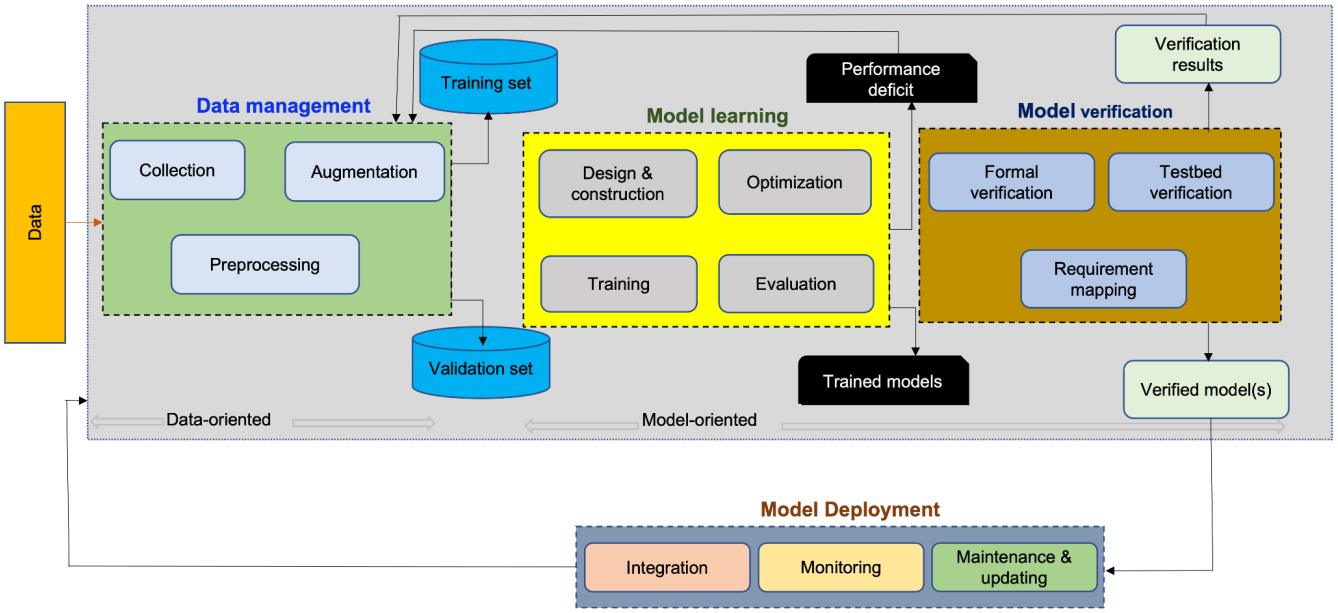


Figure 6.4. Generic ML workflow.

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

The Multi-Provider Cloud-Edge Continuum presents an increased attack surface, increasing the security risk. Visibility and control of the edge devices and nodes, for example, is not always guaranteed, and may be difficult to achieve, especially in highly distributed and heterogeneous edge cloud environments. In order to secure and provide a level of trust in COGNIT, we will research state-of-the art security mechanisms, i.e. advanced access control based on policies and attributes, trusted/confidential techniques, and the application of Federated Learning.

7.1. Risk analysis

The main risks on the COGNIT Architecture components are summarised in the table below. For each COGNIT component the main threats are identified, and their risk is given in terms of severity and likelihood:

Components	Threats	Severity / Likelihood
Device Client (DC)	• Threat on the runtime software causing unavailability: misconfiguration of the DC API service	High/Medium
	• Threat on the runtime software causing tampering: unauthorised access to inject vulnerability (ex. backdoor)	Medium/Medium
	• Threat on the runtime software causing data leak: insecure communications (ex. man-in-the-middle)	Medium/Medium
	• Threat on the runtime hardware causing unavailability: exhaustion of edge device compute/memory/network resources (ex. denial of service attack), physical access for memory or disk inspection	High/Medium
	• Threat on the runtime software causing data loss: ransomware	Medium/Medium
	• Threat on the wireless network causing unavailability: loss of connectivity, evil twin	High/Medium
Serverless Runtime (SR)	• Threat on the SR software causing unavailability: misconfiguration of the functions, data loss during live migrations	High/Medium

	• Threat on the SR software causing a data leak: Inadequate monitoring and alerting	Medium/Medium
	• Threat on the SR software causing tampering: over privileged access to resources for the functions	Medium/Medium
	• Threat on the SR hardware causing unavailability: exhaustion of edge node compute/memory/network resources (ex. denial of service attack)	High/Medium
Provisioning Engine (PE)	• Threat on the PE software causing causing unavailability: misconfiguration of the APIs to the devices and the Cloud/Edge	High/Low
	• Threat on the PE software causing a data leak: over privileged access for devices to runtimes (Read)	Medium/Low
	• Threat on the PE software causing tampering: over privileged access for devices to runtimes (Create, Update, Delete)	Medium/Low
	• Threat on the PE hardware causing causing unavailability: exhaustion of provisioning engine compute/memory/network resources (ex. denial of service attack)	High/Low
Cloud-Edge Manager (CEM)	• Threat on the CEM software causing unavailability: misconfiguration of the CEM APIs	High/Low
	• Threat on the CEM software causing a data leak: misconfigured access for users to cloud/edge resources	Medium/Low
	• Threat on the CEM software causing tampering: insecure secret management	Medium/Low
	• Threat on the CEM hardware causing unavailability: vulnerabilities in the underlying infrastructure (ex. SPECTRE)	High/Low
AI-Enabled Orchestrator (AIO)	• Threat on the AIO software causing causing unavailability: misconfigured orchestration workflows	High/Low
	• Threat on the AIO software causing a data leak: misconfiguration of access to AIO API, insecure orchestration workflows	Medium/Low

	• Threat on the AIO software causing tampering: training data poisoning for the orchestration ML algorithms	Medium/Low
	• Threat on the AIO hardware causing causing unavailability: vulnerabilities in the underlying infrastructure	High/Low

Table 7.1. Risks for COGNIT Architecture components.

This table presents high level risks for the COGNIT components, with examples of threats. The risk analysis will be refined in the next iterations of the project, and mitigation measures aiming to reduce the risks corresponding to the threats will be added.

7.2. Advanced access control

The complexity of FaaS workloads necessitates a very granular access control: who can run what function, what data can be accessed, when a workload can run and how much resources can be consumed. In order to implement these requirements, the COGNIT Framework can rely on the Cloud-Manager authorization services that secures resources such as hosts, virtual machines, networks, services, … using for example access control lists (ACLs), certificates or tokens. The CyberSecurity Use Case brings additional access control requirements, with the need for security policies on geographic zones that define a security level for specific geographic areas. We have identified for example Open Policy Agent as a solution to implement context-aware security policies in a declarative and portable way.

7.3. Confidential computing

Edge devices (and in a lesser measure edge nodes) are vulnerable to physical tampering, an attacker can potentially access them and exploit vulnerabilities that would not be possible to do remotely. In order to do that, confidential (or trusted) computing techniques can be applied to reduce this risk. Confidential computing (CC) relies on “Trusted Execution Environments (TEEs)” which are secure areas in processors that guarantee that the code and data are protected with respect to confidentiality and integrity against unauthorised actors. This way, data “in use” is protected while in RAM, which is an appreciable improvement on protecting the data in transit and at rest.

The original Intel® Software Guard Extensions (SGX) has been deprecated in latest processors but AMD proposes Secure Encrypted Virtualization - SEV and ARM has introduced its Confidential Compute Architecture (CCA) in the Armv9-A family. In order to expose the confidential computing capabilities of the testbed infrastructure to the COGNIT Framework, some adaptations would need to be performed. The Cloud-Edge Manager needs to provide in the Provider Catalogue specific filters to identify “CC capable” servers, and the AI-Enabled Orchestrator should support this “CC capable” deployment constraint for edge devices and nodes.

7.4. Federated Learning

The privacy of data produced by and stored on edge devices is particularly important. Those devices can for example record personal information of individuals that is covered by GDPR, or handle information that needs to stay locally and cannot be shared with other devices or systems. This poses a challenge for machine learning approaches such as AI-based anomaly detection algorithms that rely on data collected by various devices to train detection models, and then need to redistribute those models on various devices for inference. Federated learning is the main state of the art approach to solve this problem. Federated learning can use local datasets in the edge to train machine learning algorithms without exchanging the sensitive edge data samples.

This approach provides the added benefit of reducing the volume of exchanged data for training, which is something particularly interesting in edge contexts where bandwidth and connectivity can be limited and unreliable. The ecosystem for Federated learning frameworks and libraries is very dynamic, from plugins of established machine learning platforms such as TensorFlow Federated[3] to interoperable frameworks like Flower[4] that focuses on edge use cases. The applicability of federated learning techniques in a FaaS edge environment will be studied in the cybersecurity Use Case, where anomaly detection will be performed on vehicles that have confidentiality requirements preventing the exchange of information for training.

[3] https://www.tensorflow.org/federated
[4] https://flower.dev

8. Software Requirements

This section identifies the software requirements and functionality gaps derived from the sovereignty, sustainability, interoperability and security requirements defined in Section 2, from the user requirements summarised in Section 3 (and described in full detail in Deliverable D5.1), and from the definition of the main components of the COGNIT Architecture, as described in Sections 5, 6, and 7 of this document.

8.1. Device Client

SR1.1 Interface with Provisioning Engine

Description: Implementation of the communication with the Provisioning Engine.

- The Device Client shall be able to ask for the creation of a Serverless Runtime with specific requirements to the Provisioning Engine.
- The Device Client shall be able to ask for information about a Serverless Runtime (read) to the Provisioning Engine.
- The Device Client shall be able to ask for deletion of a Serverless Runtime to the Provisioning Engine.
- The Device Client shall be able to ask for an update of the requirements of a Serverless Runtime to the Provisioning Engine.

SR1.2 Interface with Serverless Runtime

Description: Implementation of the communication of with the Serverless Runtime

- The Device Client shall be able to upload data from the device to the Serverless Runtime.
- The Device Client shall be able to upload a function to the Serverless Runtime.
- The Device Client shall be able to request for executing a function to the Serverless Runtime.
- The Device Client shall be able to request to transfer data from external resources to the Serverless Runtime.

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: Device Client must be secured.
- All the communications between the Device Client and the Provisioning Engine shall be encrypted and signed.
- All the communications between the Device Client and the FaaS Runtime shall be encrypted and signed.
- Device Client shall take into account the latest legislative frameworks, such as the NIS2 directive, the GDPR, and the CRA.

8.2. Serverless Runtime

SR2.1 Secure and Trusted FaaS Runtimes

Description: Automated building of secure and trusted images (vulnerability scans, security assessment) related to different flavours of FaaS Runtimes.
- Cloud-Edge Manager shall provide a base FaaS image that exposes a REST API interface to the device for building and executing functions, provides a secure channel for the communication with the Device and pushes metrics to the Cloud-Edge Manager (for monitoring and auditing).
- Cloud-Edge Manager shall provide FaaS images (from the base one) adding specific libraries (e.g. python libraries for image segmentation) needed for the execution of functions according to the need of the different Use Cases.

SR2.2 Secure and Trusted DaaS Runtimes

Description: Automated building of secure and trusted images (vulnerability scans, security assessment) related to different flavours of DaaS Runtimes.
- Cloud-Edge Manager shall provide a DaaS image compliant with the EU GDPR that exposes a REST API interface to the device for data transfer, provides a secure channel for the communication with the device, pushes metrics to the Cloud-Edge Manager (for monitoring and auditing).
- Cloud-Edge Manager shall provide DaaS images (from the base one) adding specific applications, such as a relational DB (e.g. MariaDB), an object storage (e.g. MinIO) according to the needs of the different Use Cases.

8.3. Provisioning Engine

SR3.1 Provisioning Interface for the Device to manage Serverless Runtimes

Description: Provide an interface to the Device asking for a Serverless Runtime to offload functions and data transfer on any resource of the cloud-edge continuum.

- A document with requirements and attributes for the Provisioning Interface shall be defined and provided as input by the Device.
- Provisioning interface shall implement a REST API to create/read/update/delete Serverless Runtimes.
- Provisioning interface shall provide means of secure communication with the Device.
- Provisioning interface shall provide means of secure communication with the Cloud-Edge Manager and the AI-Enabled Orchestrator.

8.4. Cloud-Edge Manager

SR4.1 Provider Catalogue

Description: Implement a backend to persist information about the available providers of the cloud-edge infrastructure.

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

SR4.2 Edge Cluster Provisioning

Description: Provision Edge Clusters as a set of software-defined compute, network, storage on any cloud/edge location available in the Provider Catalogue.

- A provisioning template for Edge Cluster shall be defined and provided as an input by the AI-Enabled Orchestrator.
- Provisioning engine shall implement an API to provision/update/scale/migrate Edge Clusters.

SR4.3 Serverless Runtime Deployment

Description: Deploy Serverless Runtime as Virtualized Workloads (e.g. Containers or VMs/microVMs) on the cloud-edge infrastructure.

- A deployment template specifying the different components of the Serverless Runtime (i.e. FaaS Runtime & DaaS Runtimes) and their dependencies shall be defined and provided as an input by the Provisioning Engine.

- Provisioning Engine shall provide an API to deploy/update/scale/migrate 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 and datastores).
- A distributed system shall be used for collecting metrics from Serverless Runtimes deployed across the cloud-edge infrastructure.
- Intrusion and anomaly detection of the different Edge Cluster entities and Serverless Runtimes shall be available.

SR4.5 Authentication & Authorization

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

- Provisioning engine shall be able to use mechanisms for delegation of authentication and authorization.
- The device shall be able to use x509 certificates for requests to the Cloud-Edge Manager through the Provisioning Engine.

8.5. AI-Enabled Orchestrator

SR5.1 Building Learning Model

Description: Implement AI/ML model based on collected metrics from Edge Cluster entities and serverless runtimes deployed across the distributed cloud-edge continuum.

- AI-Enabled Orchestrator shall be based on a learning model component that trains with data having temporal dependency, different distributions, data size, data correlations, etc. and interacts with the Cloud-Edge Manager and the Provisioning Engine.

SR5.2 Smart Deployment of Serverless Runtimes

Description: Implement a Smart Workload Orchestrator (SWO) that exposes a REST API used by the Cloud-Edge Manager for requesting the deployment plans used for provisioning the Serverless Runtimes.

- The SWO component shall interact with the REST API for offering the decision obtained from the Learning Model (SR5.1).
- A component for updating the deployment policies of Serverless Runtimes on demand from the Cloud-Edge Manager that receives requests from Device Clients shall be implemented.

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 FaaS workloads at the hardware level using Confidential Computing (CC) techniques.

- The resource definition model shall be expanded to allow CC-capable devices and 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 FaaS AI workloads that have confidentiality requirements preventing the exchange of information for training.

- Cloud-Edge Manager shall provide a FL framework, consisting of a server to manage the FL process and agents integrated in the Serverless Runtime.
- The Orchestrator shall include the FL-capable constraint to deploy workloads requiring FL to devices and hosts provisioned with FL agents.

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	Secure and trusted Device clients	UR0.1 UR0.2 UR0.3 UR0.4 UR0.8 UR1.4 UR1.5 UR2.3 UR2.4 UR3.1 UR3.2 UR3.3 SER0.1 SER0.3

Table 9.1. System requirements for the Device Client.

Id	Description	Source
SR2.1	Secure and Trusted FaaS Runtimes	UR0.1 UR0.5 UR1.4 UR2.1 UR2.3 SUR0.1 IR0.1 SER0.1 SER0.2 SER0.4 SER0.6
SR2.2	Secure and Trusted DaaS Runtimes	UR0.2 UR0.3 UR0.4 SOR0.4

		IR0.1 SER0.2 SER0.4

Table 9.2. System requirements for the Serverless Runtime.

Id	Description	Source
SR3.1	Provisioning Interface for the Device to manage Serverless Runtimes	UR0.5 UR0.8 UR4.1 IR0.3 SER0.1 SER0.2 SER0.5

Table 9.3. System requirements for the Provisioning Engine.

Id	Description	Source
SR4.1	Provider Catalogue	UR0.5 UR4.1 SOR0.1 SUR0.1 IR0.2 IR0.3
SR4.2	Edge Cluster Provisioning	UR0.5 UR1.2 SOR0.3 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2
SR4.3	Serverless Runtime Deployment	UR0.5 UR1.1 IR0.1 IR0.2 IR0.3 SER0.1 SER0.2 SER0.4

SR4.4	Metrics, Monitoring, Auditing	UR1.4 UR2.1 IR0.2 IR0.3
SR4.5	Authentication & Authorization	UR0.8 UR1.1 IR0.2 IR0.3

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

Id	Description	Source
SR5.1	Building Learning Model	IR0.2 SUR0.2 SUR0.3 UR1.2 UR2.1 UR2.2
SR5.2	Smart Deployment of Serverless Runtimes	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 Envs.

PART III. Verification and Implementation Plan

10. Software Build and Verification

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

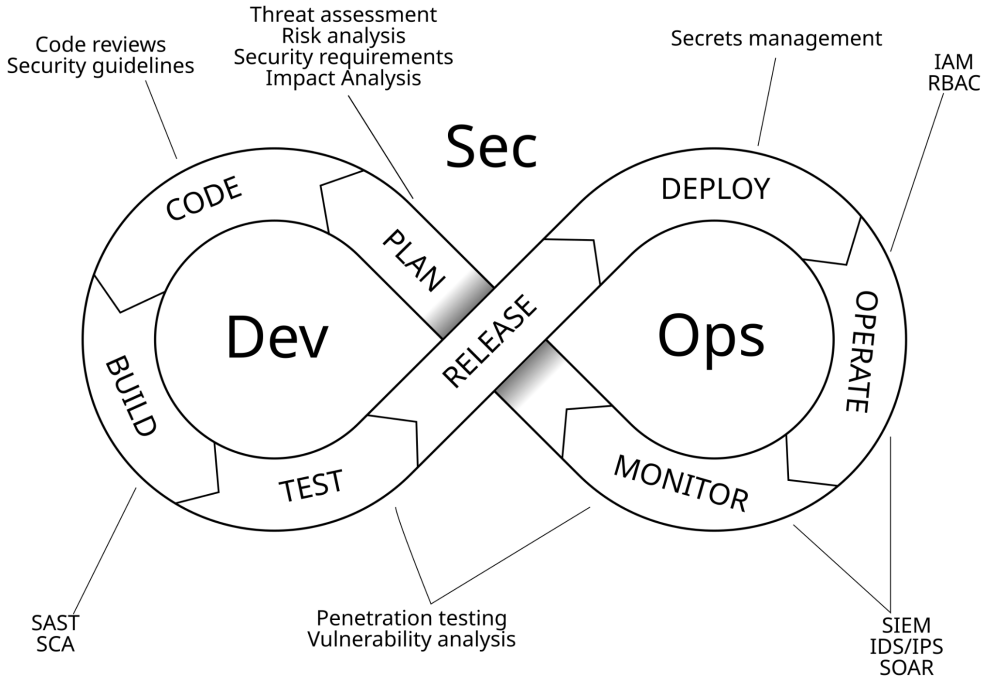


Figure 10.1. DevSecOps integrates development, operations, and security activities.

10.1. Verification Methodology

The goal of the verification process is to assess that the functional components of the software platform conforms to the Software Requirements identified in Section 8. This, in turn, will validate that the COGNIT Framework is feature-complete and able to achieve the objectives of the Projects' Use Cases as defined in Deliverable D5.1.

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 platform component is presented in a separated table that includes a brief description of each scenario.

SW Req.	Verification Scenario
SR1.1	VS1.1.1 The Device Client asks the Provisioning Engine for a Serverless Runtime with certain characteristics and it receives the ID of the created Serverless Runtime. VS1.1.2 The Device Client asks the Provisioning Engine for information about the created Serverless Runtime. VS1.1.3 The Device Client requests the Provisioning Engine to update the features of a Serverless Runtime. The Device Client requests again information about the Serverless Runtime to verify that the Serverless Runtime has been modified. VS1.1.4 The Device Client requests the Provisioning Engine to delete a Serverless Runtime. The Device Client requests information about the Serverless Runtime to verify that it no longer exists.
SR1.2	VS1.2.3 The Device Client uploads data from the device to the Serverless Runtime and receives an acknowledgement. VS1.2.4 The Device Client uploads a function to the Serverless Runtime and receives an acknowledgement. VS1.2.5 The Device Client requests the execution of a function to the Serverless Runtime and receives the result of the execution. VS1.2.6 The Device Client requests transfer data from external resources to the Serverless Runtime and receives an acknowledgement.
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 asks the Provisioning Engine for a Serverless Runtime with the data encrypted and signed and the request is accepted. VS1.5.2 The Device Client asks the Provisioning Engine for a Serverless Runtime without the data encrypted or signed and the request is refused.

Table 10.1. Verification scenarios for Device Client.

SW Req.	Verification Scenario
SR2.1	VS2.1.1 Build and instantiate a FaaS Runtime image for Python language and test the execution of a function using a secure communication channel VS2.1.2 Build and instantiate a FaaS Runtime image for C language and test the execution of a function using a secure communication channel
SR2.2	VS2.2.1 Build and instantiate a DaaS Runtime image for SQL DB (e.g. MariaDB) and test uploading and copying data using a secure communication channel VS2.2.1 Build and instantiate a DaaS Runtime image for Object Storage (e.g. MinIO) and test uploading and copying data using a secure communication channel

Table 10.2. Verification scenarios for the Serverless Runtime.

SW Req.	Verification Scenario
SR3.1	VS3.1.1 A YAML file with the device requirements is provided to the Provisioning Engine and it returns the Serverless Runtime ID. VS3.1.2 Query the Provisioning Engine to return the status of a

	Serverless Runtime identified by its ID. VS3.1.3 A YAML file with the updated device requirements is provided to the Provisioning Engine that updates the associated Serverless Runtime. VS3.1.4 Delete a Serverless Runtime providing its ID.

Table 10.3. Verification scenarios for the Provisioning Engine.

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 a cost per hour threshold VS4.1.4 Filtering the providers according to energy consumption per hour threshold VS4.1.5 Filtering the providers according to a some specific hardware characteristics (e.g. GPUs, 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

	VS4.2.3 Query the Cloud-Edge Manager to scale up/down the resources (CPU, memory and disks) of a Serverless Runtime identified by its ID VS4.2.4 Query the Cloud-Edge Manager to update the deployment of the Serverless Runtime identified by its ID VS4.2.5 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 Provisioning Engine using x509 certificates.

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 ready to take as input to the Model VS5.1.3 Feedback-aware performance check when train 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 Users Quality of Service(QoS) /Quality of Experience(QoE) will check for each Smart workload orchestrator decision for deployment of serverless runtimes

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

SW Req.	Verification Scenario

SR6.1	VS6.1.1 Define a security policy that based on geographic zone attribute VS6.1.2 Check enforcement of new security policy when edge device moves closer from one edge node than 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 Envs.

11. Instantiation of the COGNIT Architecture

This section identifies the gaps and the initial set of technologies that will be used to instantiate the COGNIT 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 involved in the implementation plan of the different software requirements has prioritised the use of European open source alternatives:

SR	Description	Implementation
SR1.1	Secure and Trusted Device Clients Engine	New component (HTTPS client plus certificate management tool).
SR2.1	Secure and Trusted FaaS Runtimes	New component developed as an OpenNebula Virtual Appliance (e.g. based on openSUSE and KVM).
SR2.2	Secure and Trusted DaaS Runtimes	New component developed as an OpenNebula Virtual Appliance (e.g. based on openSUSE and KVM), and incorporating specific applications such as a relational database (e.g. MariaDB) or object storage (e.g. MinIO).
SR3.1	Provisioning Interface for the Device to manage Serverless Runtimes	New component.
SR4.1	Provider Catalogue	Expansion of the OpenNebula OneProvision Edge Catalogue.
SR4.2	Edge Cluster Provisioning	Expansion of the OpenNebula OneProvision component.
SR4.3	Serverless Runtime Deployment	Expansion of the OpenNebula OneFlow and *oned* components, with Serverless Runtimes defined as OneFlow Services.
SR4.4	Metrics, Monitoring, Auditing	Enhancement of OpenNebula OneGate component, plus Prometheus/Zabbix integration, and Edge Cluster monitoring enhanced with auditing (e.g. for intrusion and anomaly detection).
SR4.5	Authentication & Authorization	Expansion of the OpenNebula Authentication component and support to ACLs.
SR5.1	Building Learning Model	New component developed using ML/AI frameworks like PyTorch or TensorFlow.

SR5.2	Smart Deployment of Serverless Runtimes	Enhancement of the OpenNebula Scheduler through the integration with an external module for AI-Enabled Orchestration.
SR6.1	Advanced Access Control	Enhancement of the OpenNebula support to ACLs.
SR6.2	Confidential Computing	Enhancement of the OpenNebula drivers that expose CC hardware capabilities.
SR6.3	Federated Learning	New component developed using Federating Learning frameworks such as Flower.

Table 11.1. Implementation plan per Software Requirement.

12. Prioritisation of Software Requirements

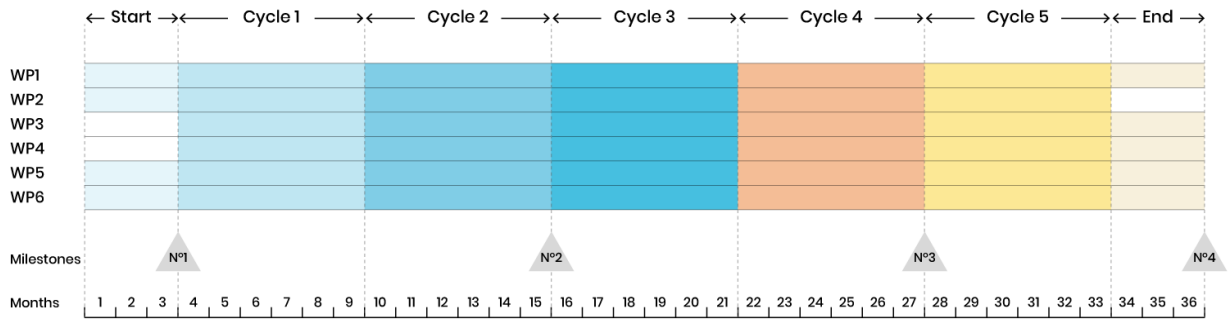


Figure 12.1. COGNIT research and innovation cycles, by Milestone.

As defined in the Project's proposal, an agile approach towards software R&D has been adopted, based on shorter, iterative cycles (see Figure 12.1). The research and development of the framework components will take place in iterative 6-month cycles which will be conducted in parallel to the validation of the new components and product versions from the previous cycles against the Use Cases. This will allow the COGNIT team to obtain quicker feedback, identify unanticipated issues at an earlier stage, and guide the objectives and definition of the development in each subsequent cycle, incorporating new emerging technologies and undertaking any required corrective action.

WP	Task	Expectations for MS2 per Component	SR
WP3	T2.3 T3.1	**Device Client:** *First version of distributed FaaS model components: DSL; Device Runtime with support for specification of FaaS Runtime context, images, patterns, and attributes by developers.*	SR1.1 SR1.2 SR1.3 SR1.4 SR1.5
WP3	T2.3 T3.3 T3.4	**Serverless Runtime:** *FaaS Runtime and Images. Security processes and controls are automated and integrated into the runtime lifecycle following a DevSecOps approach.*	SR2.1 SR2.2
WP3	T2.3 T3.2	**Provisioning Engine:** *Provision Engine to enable communication between Device Runtime and FaaS Runtime.*	SR3.1
WP4	T2.4 T4.1	**Cloud-Edge Manager:** *First version of a Cloud-Edge Serverless Manager offering a multi-provider abstraction layer for highly distributed execution of the FaaS Runtimes, and automatic deployment and fault tolerance of edge PoPs.*	SR4.1 SR4.2 SR4.3 SR4.4 SR4.5
WP4	T2.4 T4.3	**AI-Enabled Orchestrator:** *First version of Workload Orchestration with baseline functionality to enable AI-based techniques for smart runtime placement and workload relocation.*	SR5.1 SR5.2

Table 12.1. Main tasks involved in implementing the SW requirements towards Milestone 2 (M15).

MEASURABLE & VERIFIABLE RESULT	KPI	Device Client					Serverless Runtime		Prov. Eng.	Cloud-Edge Manager					AI-Enabled Orchestrator		Secure & Trusted Exec of Comp.Envs.		
		SR1.1	SR1.2	SR1.3	SR1.4	SR1.5	SR2.1	SR2.2	SR3.1	SR4.1	SR4.2	SR4.3	SR4.4	SR4.5	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.									3									
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.												2–4		2–4	2–4			
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.										3								
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.									2–3									
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.									2–3	2–3	2–3							
Seamless access for end users.	**[KPI2.1]** Meet experience level agreements with dynamic needs without intervention of the user.														2–4	2–4			
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.	2	2	2	2				2										
Developers can change the deployment requirements of the FaaS Runtime according to execution flow.	**[KPI2.3]** Migration across edge PoPs to meet application needs.	3							3										
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.					2	2–3	2–3											
Efficient orchestration of serverless workloads across the Continuum.	**[KPI3.1]** Backward compatibility with virtualization tech (e.g. KVM, Firecracker) and containers (e.g. K8s).									2–3		2–3							
Able to migrate workloads in geo-distributed edge clouds, combining edge/cloud resources.	**[KPI3.2]** Migrate workloads across the Continuum with minimal downtime.										2–3								
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.												3–4						
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.													2				3–4	4
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.														2–4	2–3			
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.														3–4				
Hierarchical multi-constraint resource management layer to minimize environmental impact.	**[KPI4.3]** Reduction of at least 10% energy consumption compared with manual edge deployment.														4	4			
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.																3–4		

Table 12.2. Expected contribution of each Software Requirement to meeting the Project's Milestones and global KPIs.

13. Conclusions and Next Steps

This report identifies and analyses the main sovereignty, sustainability, interoperability, and security requirements, as well as the user requirements, derived from the European context and from the Project's specific Use Cases. They will guide the research and development of the project, having played a central role in this initial definition of the architecture of the COGNIT Framework. From those global and user requirements, a list of Software Requirements and functional gaps to be implemented by the components of the COGNIT Framework have been identified, followed by a definition of the methodology and scenarios required to verify their fulfilment and applicability in the Project's Use Cases.

The new open source software components and expansions needed to meet the Software Requirements will be specified and developed within the Work Packages WP3 and WP4, with these new functionalities being tested, verified, and demonstrated as part of the Use Cases in their respective associated tasks in WP5.

This first version of the COGNIT Framework Architecture report has been released at the end of the start phase (M3). It will be updated with incremental releases at the end of each research and innovation cycle (i.e. M9, M15, M21, M27, M33).