A Cognitive Serverless Framework for the Cloud-Edge Continuum

D4.5 COGNIT Serverless Platform - Scientific Report - e

Version 2.0

1 March 2026

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 centres. The continuum and their automatic and smart adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This standalone document comprehensively describes the research and development carried out across the different tasks within WP4 "AI-enabled Distributed Serverless Platform and Workload Orchestration" during the whole project in accordance with the COGNIT architecture, and provides details about the development and integration activities associated to key components of the COGNIT Framework such as the Cloud-Edge Manager and AI-Enabled Orchestrator.

Copyright © 2025 SovereignEdge.Cognit. All rights reserved.

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

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

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:	WP4. AI-enabled Distributed Serverless Platform and Workload Orchestration
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	2.0
Contractual Date of Delivery:	30/09/2025
Actual Date of Delivery:	01/03/2026
Lead Author:	Monowar Bhuyan (UMU) & Paul Townend (UMU)
Authors:	Yashwant Singh Patel (UMU), Adil Bin Bhutto (UMU), Rohail Gulbaz (UMU), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), David Eklund (RISE), Javad Forough (UMU), Idoia de la Iglesia (Ikerlan), Iván Valdés (Ikerlan), Fátima Fernández (Ikerlan), Carlos Lopez (ACISA), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Marco Mancini (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Thomas Ohlson Timoudas (RISE), Iván Valdés (Ikerlan), Cristina Cruces (Ikerlan), Martxel Lasa (Ikerlan), Victor Palma (OpenNebula), Michal Opala (OpenNebula), Jakub Walczak (OpenNebula), Carlos Moral (OpenNebula), Aitor Garciancia (Ikerlan), Francesco Renzi (Nature 4.0), Riccardo Valentini (Nature 4.0)
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	01/12/2025	Draft	Initial Draft
0.2	22/12/2025	Peer-Reviewed	Reviewed Draft
1.0	31/12/2025	Submitted	Final Version
1.1	27/02/2026	Draft	Initial Draft
1.2	28/02/2026	Peer-Reviewed	Reviewed Draft
2.0	01/03/2026	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.2	29/12/2025	Antonio Álvarez (OpenNebula)
1.2	28/02/2026	Marco Mancini (OpenNebula)

Summary of Changes from Previous Versions

Second Version of Deliverable D4.5

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

Executive Summary

This is the fifth and final “COGNIT Serverless Platform - Scientific Report” produced in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration”. It is a standalone document that describes in detail the research activities and the work carried out during the whole project to achieve the complete implementation and subsequent integration of the Software Requirements linked to these main components of the COGNIT Framework under WP4:

Cloud-Edge Manager

- **SR4.1** Provider Catalogue

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

- **SR4.2** Edge Cluster Provisioning

  The Cloud-Edge Manager must be able to provision Edge Clusters as a set of software-defined compute, network, storage on any cloud/edge location

- **SR4.3** Serverless Runtime Deployment

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

- **SR4.4** Metrics, Monitoring and Auditing

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

- **SR4.5** Authentication and authorization

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

- **SR4.6** Plan Executor:

  Plan Executor is responsible for the execution of plans produced by the AI-Enabled Orchestrator related to the placement and migration of Serverless Runtimes within an Edge Cluster.

AI-Enabled Orchestrator

- **SR5.1** Building Learning Models:

  Provide AI/ML models based on collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters, Serverless Runtimes, and infrastructure usage.

- **SR5.2** Smart management of Cloud-Edge resources:

  AI-Enabled Orchestrator is responsible for managing and optimizing the lifecycle of Edge Clusters and serverless runtimes within Edge Clusters according to the application requirements, infrastructure and virtual resource usage, and energy-aware policies.

Table of Contents

Abbreviations and Acronyms

ADWIN	Adaptive windowing
AE	Auto Encoder
AI	Artificial Intelligence
AI-O	AI-Enabled Orchestrator
API	Application Programming Interface
ARUR	Average Resource Utilization Ratio
CLI	Command Line Interface
CMA	Carbon-aware Model Agent
CPCA	Common Principal Component Analysis
CSP	Cloud Service Provider
DB	Database
DCs	Data Centres
DL	Deep Learning
DTW	Dynamic Time Warping
EC	Energy Consumption
EP	Energy Provider
EVPN	Ethernet VPN
FaaS	Function as a Service
FFNN	Feed-Forward Neural Network
GC	Global Controller
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HRUR	Host Resource Utilisation Ratio
HTTP	Hypertext Transfer Protocol
HVMC	Host-VM Combination
IDEC	Improved Deep Embedded Clustering
ILP	Integer Linear Programming
IP	Internet Protocol
IPAM	IP Address Management
JSON	Javascript Object Notation
KVM	Kernel Virtual Machine
LC	Local Controller
LSTM	Long Short-Term Memory

MAPE-K	Monitoring, Analysis, Planning, Execution, and Knowledge
MI	Million Instructions
MIPS	Million Instructions Per Second
ML	Machine Learning
MOGA	Multi-Objective Genetic Algorithm
MSE	Mean Squared Error
MTS	Multivariate Time Series
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OS	Operating System
PDN	Power Distribution Network
PH	Page-Hinkley
PSRs	Power Source Regions
QoS	Quality of Service
RAE	Relative Absolute Error
RAPL	Running Average Power Limit
REST	Representational State Transfer
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
ROCKET	Random Convolutional Kernel Transform
RUR	Resource Utilization Ratio
SGD	Stochastic Gradient Descent
SLA	Service Level Agreement
SLO	Service Level Objective
SoC	State of Charge
SPEA2	Strength Pareto Evolutionary Algorithm 2
SRTs	Serverless Runtimes
SVD	Single Value Decomposition
TCN	Temporal Convolutional Network
VM	Virtual Machine

1. Introduction

This document describes the AI-enabled Distributed Serverless Platform and Workload Orchestration components of the COGNIT Framework implemented under WP4.

The implementation follows a **MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge)** autonomic computing architecture, which enables COGNIT to autonomously manage and optimize resources across the cloud-edge continuum.

The MAPE-K architecture is foundational to the COGNIT design, enabling a proactive and automated approach to infrastructure management. By using predictions to anticipate workload changes, identify optimization opportunities, and execute infrastructure adaptations before performance degradation occurs, COGNIT continuously optimizes resource allocation, minimizes energy consumption, and maintains service quality across the distributed cloud-edge continuum, acting on predictions rather than waiting for issues to manifest.

The following components have been developed and integrated to realize the MAPE-K architecture:

Cloud-Edge Manager (Section 2): The core infrastructure management component implemented using OpenNebula, extended with COGNIT-specific components. It provides:

- **Provider Catalogue (SR4.1)**: Unified abstraction layer for managing multi-provider infrastructure through OpenNebula OneForm, enabling seamless integration of resources from public cloud providers, private clouds, and edge locations.
- **Edge Cluster Provisioning (SR4.2)**: Automated provisioning of complete Edge Clusters using declarative infrastructure-as-code approaches through OneForm, deploying compute, storage, and networking resources across heterogeneous providers.
- **Serverless Runtime Deployment (SR4.3)**: Lifecycle management of Serverless Runtime workloads using OneFlow Services, enabling automated deployment, scaling, migration, and termination of Serverless Runtimes across Edge Clusters.
- **Metrics, Monitoring and Auditing (SR4.4):** Comprehensive monitoring system using customized OpenNebula probes that collect infrastructure metrics, energy consumption data, and application performance metrics, integrated into the OpenNebula Monitoring System.
- **Authentication and Authorization (SR4.5)**: Token-based authentication using Biscuit tokens, providing secure, decentralized verification and multi-tenant isolation across the distributed cloud-edge continuum.
- **Plan Executor (SR4.6)**: Execution engine that translates AI-Enabled Orchestrator optimization plans into concrete OpenNebula API calls, enabling autonomous infrastructure management operations.

AI-Enabled Orchestrator (Section 3): Smart resource management component that provides predictive analytics and optimization capabilities:

- **Building Learning Models (SR5.1)**: AI/ML models and extensions to the OpenNebula AIOps SDK that enable predictive analytics for proactive resource management. The extension adds OneFlow Service-specific predictions for

Serverless Runtime scaling requirements, workload forecasting, and performance prediction. The Device Load Forecasting component estimates device workload by aggregating system-wide metrics from OneFlow services, enabling informed device-to-cluster assignment decisions.

- **Smart Management of Cloud-Edge Resources (SR5.2)**: Multi-level optimization through the Multi-Cluster Optimizer (global device-to-cluster assignments) and Cluster Optimizer (intra-cluster placement and workload optimization), both incorporating energy-aware policies to minimize interference, energy usage and carbon footprint.

COGNIT implements a MAPE-K loop (shown in Figure 1.1) as described in the following:

- **Monitor**. Monitoring is performed by the Cloud-Edge Manager using customized probes integrated within the OpenNebula monitoring system, which collect metrics from hosts, VMs, and applications. These metrics include infrastructure health, resource utilization, energy consumption (via Scaphandre integration), and application performance metrics (via Prometheus exporters on Serverless Runtime VMs).
- **Analyze**. The AI-Enabled Orchestrator performs Analysis via ML models that predict workloads and identify optimization opportunities. The extended OpenNebula AIOps SDK processes historical metrics through ML models (including Regressions, ARIMA, Fourier analysis, and gradient boosting techniques) to generate predictions for Serverless Runtime scaling requirements and workload forecasting.
- **Plan**. The AI-Enabled Orchestrator performs Planning via optimization algorithms that generate optimal resource allocations. The Multi-Cluster Optimizer optimizes device-to-cluster assignments across the entire cloud-edge continuum, while the Cluster Optimizer optimizes Serverless Runtime placement within individual Edge Clusters.
- **Execute**. The Plan Executor performs Execution by translating plans into OpenNebula API calls to deploy, migrate, or power off Serverless Runtime VMs. Plans are expressed as XML documents and executed sequentially, with dependency handling, retry logic, and error recovery.
- **Knowledge**. The Knowledge base is distributed across multiple storage systems: SQLite databases distributed on all hosts store time-series metrics collected by monitoring probes, metrics are also pushed to the central OpenNebula database available in the OpenNebula frontend for infrastructure state and monitoring data, the OpenNebula AIOps SDK provides access to metrics from both the distributed host databases and the central database, and a shared SQLite database stores device-to-cluster assignments, application requirements, and estimated loads. This knowledge is continuously updated as metrics are collected, plans are executed, and optimization results are fed back to refine ML models and improve future decisions.

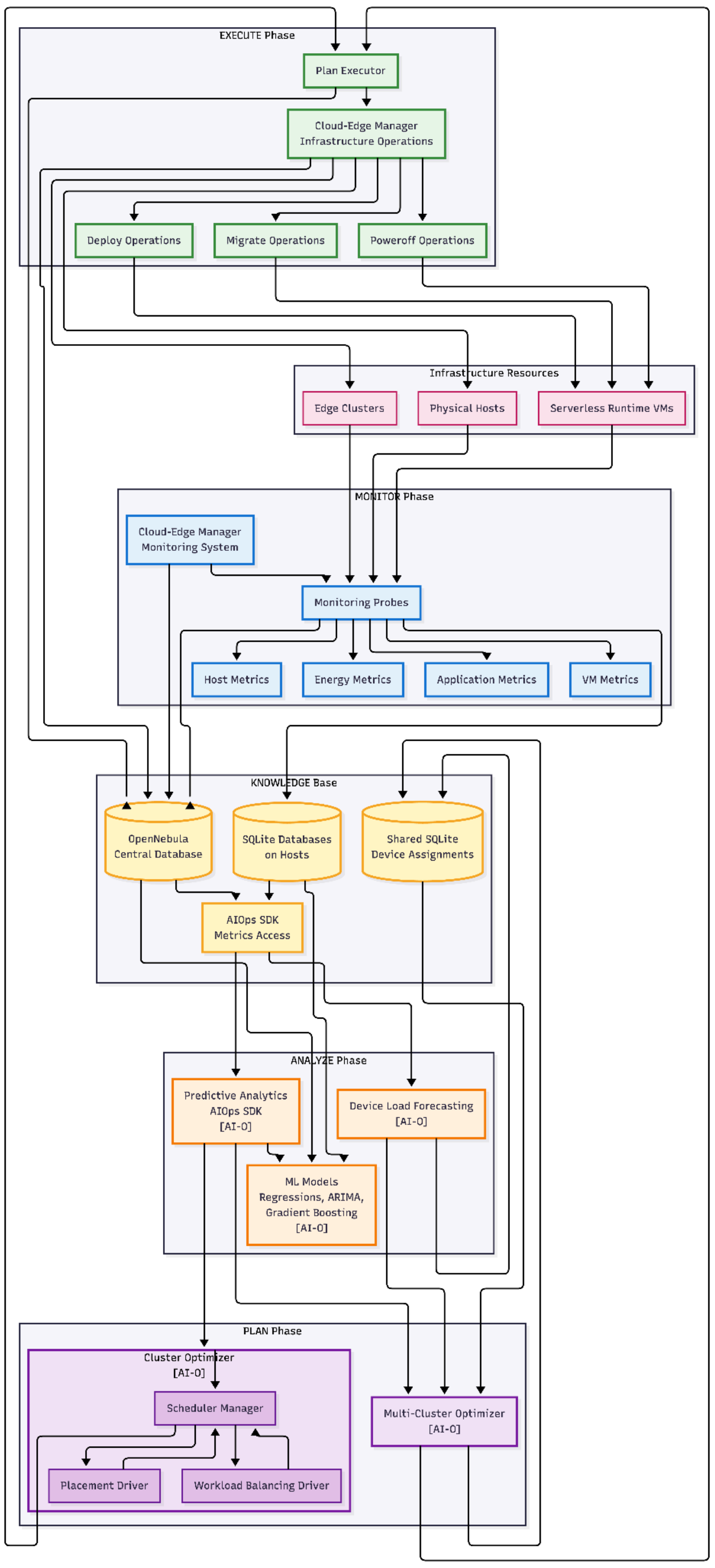

Figure 1.1: MAPE-K architecture diagram showing how COGNIT components implement the autonomic computing loop.

The following subsections provide detailed descriptions of each software requirement, presenting the final state of the implementation and integration.

2. Cloud-Edge Manager

The Cloud-Edge Manager is the core infrastructure management component of the COGNIT Framework, responsible for orchestrating distributed cloud-edge continuum resources across multiple providers and on-premises locations. The Cloud-Edge Manager is implemented using OpenNebula, extended with COGNIT-specific components and integrations.

Its primary responsibilities include:

- **Multi-provider infrastructure management** through the Provider Catalogue, enabling seamless integration of resources from public cloud providers, private clouds, and edge locations.
- **Automated Edge Cluster provisioning** using declarative infrastructure-as-code approaches that deploy complete, pre-configured Edge Clusters with networking, storage, and compute resources.
- **Serverless Runtime lifecycle management** orchestrating the deployment, scaling, migration, and termination of Serverless Runtime workloads across Edge Clusters.
- **Comprehensive monitoring and metrics collection** capturing infrastructure health, resource utilization, energy consumption, and application performance metrics for real-time operational decisions and AI/ML model training.
- **Fine-grained authentication and authorization** using token-based mechanisms that secure device access to Edge Cluster resources while maintaining multi-tenancy isolation.
- **Plan execution capabilities** that translate AI-Enabled Orchestrator decisions into concrete infrastructure operations, enabling smart, data-driven resource optimization.

The following subsections provide detailed descriptions of each software requirement (SR4.1-SR4.6), synthesizing the work performed across all development cycles and presenting the final state of the implementation.

2.1 [SR4.1] Provider Catalogue

2.1.1 Description

The Provider Catalogue is a component of the COGNIT Framework that addresses the management of a heterogeneous cloud-edge continuum spanning multiple infrastructure providers: public cloud providers, private cloud installations, on-premises data centers, and geographically distributed edge locations. The Provider Catalogue provides a unified abstraction layer that enables the COGNIT Framework to discover, register, configure, and manage connectivity to these disparate infrastructure providers through a standardized interface.

2.1.2 Architecture

The Provider Catalogue is implemented as part of OpenNebula OneForm, a Ruby-based Sinatra application that extends OpenNebula with advanced multi-provider provisioning capabilities. OneForm was developed within the European IPCEI-CIS project and has been integrated into the COGNIT Framework to satisfy SR4.1 and SR4.2 requirements. The Provider Catalogue component specifically manages the registration and lifecycle of infrastructure providers, storing provider metadata as OpenNebula Document JSON objects within the OpenNebula database.

A Provider in the COGNIT context represents a connection to a specific infrastructure source, encapsulating all necessary information to authenticate, authorize, and interact with that provider's API.

The Provider Catalogue plays a foundational role in the COGNIT Framework's multi-provider strategy:

- **Infrastructure Discovery**: During initial deployment, COGNIT administrators register available infrastructure providers through the OneForm API or web interface.
- **Edge Cluster Provisioning Prerequisite:** Before deploying an Edge Cluster (SR4.2), the AI-Enabled Orchestrator or administrator selects a target provider from the catalogue. The selected provider's connection parameters and driver are used to orchestrate the infrastructure provisioning process.
- **Device Application Requirements**: Device Clients can specify a list of providers in their application requirements, constraining function execution to Edge Clusters that are associated with providers in the catalogue. When a device submits an application requirement with provider constraints, the COGNIT Frontend and AI-Enabled Orchestrator filter available Edge Clusters to only those provisioned through the specified providers. This enables organizations to enforce data sovereignty, compliance requirements, or cost optimization policies by restricting workload execution to specific infrastructure providers.

2.1.3 Data Model

Each Provider in the catalogue contains the following key information:

- **Identity and metadata:** Provider name and description.
- **Driver specification:** Identifies the provisioning driver responsible for translating high-level provisioning requests into provider-specific API calls.
- **Connection parameters:** Authentication credentials, API endpoints, region identifiers, and other provider-specific connection details required to establish secure communication channels.
- **User input schemas:** Defines configurable parameters that users can customize when provisioning Edge Clusters on this provider (e.g., instance types, network configurations, storage options).
- **Provision tracking:** Maintains a list of all Edge Cluster provisions currently deployed on this provider, enabling resource accounting and dependency management.

2.1.4 API & Interfaces

The Provider Catalogue exposes a RESTful API through OneForm that allows it to perform provider management operations. The API supports complete CRUD (Create, Read, Update, Delete) operations on providers, with validation ensuring data consistency and integrity, as described in the following table:

Action	**Verb**	**Endpoint**	**Request Body**	**Response**
List all providers	GET	`/providers`	None	200 OK with JSON array of providers
Get provider details	GET	`/providers/:id`	None	200 OK with provider JSON object 404 Not Found if provider doesn't exist
Create new provider	POST	`/providers`	Provider JSON template	201 Created with provider ID 400 Bad Request for validation errors
Update provider	PUT	`/providers/:id`	Partial provider JSON	200 OK with updated provider 400 Bad Request for validation errors 409 Conflict if provision dependencies exist
Delete provider	DELETE	`/providers/:id`	None	204 No Content on success 409 Conflict if active provisions exist

Action	Verb	Endpoint	Request Body	Response
				404 Not Found if provider doesn't exist
Get provider driver path	GET	`/providers/:id/path`	None	200 OK with filesystem path to driver

Table 2.1.1: Provider Catalogue API Specification

2.2 [SR4.2] Edge Cluster Provisioning

2.2.1 Description

An Edge Cluster is a self-contained, geographically localized cloud infrastructure unit consisting of compute resources (hypervisor hosts), storage systems (image datastores, system datastores), and networking infrastructure (public and private networks). Each Edge Cluster serves as an execution environment for Serverless Runtimes, providing the computational substrate necessary to execute functions offloaded from Device Clients while satisfying application requirements for latency, data locality, confidential computing, and provider constraints.

2.2.2 Architecture

COGNIT addresses this challenge by integrating OpenNebula OneForm, which provides declarative infrastructure-as-code capabilities for Edge Cluster provisioning. OneForm enables administrators to define Edge Clusters through high-level specifications that encapsulate the desired state: target provider, resource quantities (number of hosts, network ranges, storage capacity), and configuration parameters. OneForm translates these specifications into provider-specific infrastructure operations, executes the provisioning workflow, monitors progress, handles failures gracefully, and automatically registers the provisioned resources with the Cloud-Edge Manager for lifecycle management.

OneForm provides several key capabilities that are essential for COGNIT's Edge Cluster provisioning:

- **Multi-provider Support**: OneForm supports provisioning Edge Clusters across multiple cloud providers and on-premises infrastructure, enabling COGNIT to deploy resources wherever they are needed.
- **Declarative Provisioning:** Edge Clusters are defined through declarative templates that specify the desired infrastructure state, enabling reproducible deployments and version-controlled infrastructure configurations.
- **State Management:** OneForm tracks infrastructure state throughout the provisioning lifecycle, enabling incremental updates, resource destruction, and drift detection.

- **Dynamic Scaling:** Edge Clusters can be scaled up (adding hosts) or scaled down (removing hosts) post-deployment to adapt to changing workload demands, enabling COGNIT to respond to workload fluctuations within specific geographic regions.
- **Elastic IP Management:** Public IP addresses are allocated dynamically through IPAM (IP Address Management) drivers specific to each provider, with automatic address range expansion and contraction based on demand.

2.2.3 API & Integration

OneForm exposes a RESTful API that is integrated into the Cloud-Edge Manager, enabling programmatic Edge Cluster provisioning and management. The API is accessible through the Cloud-Edge Manager's web interface. The main endpoints relevant for COGNIT integration include:

Action	HTTP Method	Endpoint	Description
List provisions	GET	`/provisions`	Retrieve all Edge Cluster provisions (optionally including completed ones)
Get provision	GET	`/provisions/:id`	Get detailed information about a specific Edge Cluster provision
Create provision	POST	`/provisions`	Create a new Edge Cluster provision specification
Start provision	POST	`/provisions/:id/start`	Begin the provisioning process for an Edge Cluster
Scale provision	POST	`/provisions/:id/scale`	Scale an existing Edge Cluster (add or remove hosts)
Deprovision	POST	`/provisions/:id/deprovision`	Remove an Edge Cluster and deprovision its resources

Action	HTTP Method	Endpoint	Description
Recover provision	POST	`/provisions/:id/recover`	Retry a failed provisioning operation from a specific state

Table 2.2.1: OneForm API Endpoints for Edge Cluster Provisioning

2.3 [SR4.3] Serverless Runtime Deployment

2.3.1 Description

Serverless Runtime Deployment addresses the challenge of managing the execution environment for offloaded functions within the COGNIT Framework. A Serverless Runtime is a lightweight, specialized virtual machine instance that executes user functions submitted by Device Clients through the Edge Cluster Frontend.

Serverless Runtimes are dynamically provisioned, scaled, and terminated based on application demand, embodying the serverless computing paradigm where infrastructure management is abstracted from application developers.

2.3.2 Architecture & Components

The Cloud-Edge Manager orchestrates Serverless Runtime deployment using OpenNebula OneFlow, a multi-VM service orchestration component that treats groups of virtual machines as single logical entities.

The Serverless Runtime deployment component provides the following key capabilities:

- **OneFlow Service-Based Deployment**: Serverless Runtimes are deployed as OpenNebula OneFlow Services, enabling multi-VM orchestration where groups of virtual machines are managed as single logical entities. Service templates define roles (FaaS execution units), dependencies, elasticity policies, and lifecycle management rules.
- **Flavour-Specific Templates:** Pre-configured VM templates exist for each COGNIT use case (SmartCity, WildFire, Energy, Cybersecurity), with pre-installed libraries, executables, and tools tailored to specific application requirements. These templates are built using KIWI and pre-populated in the Cloud-Edge Manager during OpsForge deployment.
- **Smart Placement:** The AI-Enabled Orchestrator provides smart placement and migration of Serverless Runtimes based on predicted workloads, energy usage, and multi-objective optimization algorithms. Placement decisions consider host capacity, power consumption, resource contention, and data locality.
- **Automatic Scaling:** Serverless Runtimes can be automatically scaled up or down based on queue depth metrics, workload predictions, and performance indicators. The Edge Cluster Frontend exposes scaling endpoints that enable the AI-Enabled Orchestrator to dynamically adjust Serverless Runtime cardinality, with graceful drain algorithms preventing interruption of in-flight executions.

- **Monitoring Integration**: Serverless Runtimes integrate with Cloud-Edge Manager monitoring system for metrics collection, including CPU utilization, memory usage, queue depths, and execution status. These metrics inform scaling decisions and enable the AI-Enabled Orchestrator to make data-driven optimization choices [SR4.4].
- **Secure Authentication**: Biscuit token-based authentication is integrated for Serverless Runtime operations, replacing username/password authentication with cryptographically secure, capability-based tokens that support fine-grained access control and delegation [SR4.5].

2.3.3 Data Model

A Serverless Runtime OneFlow Service is represented as a JSON document:

```json
{
  "name": "Serverless-Runtime-Nature-Finland",
  "deployment": "straight",
  "description": "Serverless Runtime for Nature use case in Helsinki",
  "roles": [
    {
      "name": "FaaS",
      "cardinality": 3,
      "vm_template": 42,
      "elasticity_policies": [
        {
          "type": "CHANGE",
          "adjust": 1,
          "expression": "CPU > 80",
          "period_number": 2,
          "period": 60
        }
      ],
      "scheduled_policies": [],
      "vm_template_contents": "CPU=\"1.0\" MEMORY=\"2048\" VCPU=\"2\"
CONTEXT=[COGNIT_FLAVOUR=\"nature\", RABBITMQ_HOST=\"10.0.1.5\",
EDGE_CLUSTER_FRONTEND=\"http://192.168.1.50:1339\"]
PROMETHEUS_EXPORTER=\"9100\" READY_SCRIPT=\"nc -vz localhost 8000\""
    }
  ]
}
```

In the following a description of the key attributes:

- `cardinality`: Number of Serverless Runtime VMs to deploy for this role.
- `vm_template`: ID of the OpenNebula VM template to use for instantiation.
- `elasticity_policies`: Automatic scaling rules. The example policy scales up by 1 VM if CPU utilization exceeds 80% for 2 consecutive 60-second periods.

- `vm_template_contents`: Additional VM attributes merged with the base VM template, enabling runtime customization without modifying the base template. Includes contextualization variables, scheduling constraints, and resource specifications.

2.3.4 API & Interfaces

The Cloud-Edge Manager exposes Serverless Runtime deployment operations through the OpenNebula OneFlow REST API:

Action	Verb	Endpoint	Request Body	Response
Instantiate Service (deploy Runtime)	POST	`/service_template/:template_id/action`	`{"merge_template": {...}}`	201 Created with service ID
Get Service (Runtime) info	GET	`/service/:id`	None	200 OK with service JSON
Scale Service (Runtime)	POST	`/service/:id/scale`	`{"role": "FaaS", "cardinality": 5}`	200 OK
Delete Service (Runtime)	DELETE	`/service/:id`	None	204 No Content
Recover Service	POST	`/service/:id/recover`	None	200 OK (retry failed operations)

Table 2.3.1: Serverless Runtime Deployment API Specification

Serverless Runtime deployment is triggered by multiple actors in the COGNIT ecosystem:

1. **COGNIT Frontend - Initial Deployment:** When a Device Client registers application requirements for the first time, the COGNIT Frontend pre-deploy Serverless Runtimes in the assigned Edge Cluster to reduce cold-start latency. This proactive provisioning ensures that when the device first offloads a function, execution begins immediately without waiting for VM instantiation.
2. **Edge Cluster Frontend - Scaling Operations**: The Edge Cluster Frontend exposes a `/v1/scale` endpoint that the AI-Enabled Orchestrator invokes to adjust Serverless Runtime cardinality based on workload predictions and queue depth metrics.
3. **AI-Enabled Orchestrator - Placement Optimization:** When deploying new Serverless Runtimes, the AI-Enabled Orchestrator computes optimal host assignments based on predicted resource requirements, power consumption, data locality, and hardware capabilities (GPU, TEE). The scheduler drivers integrated with

Cloud-Edge Manager ensure that VMs are placed on hosts that maximize performance while minimizing energy consumption.

2.4 [SR4.4] Metrics, Monitoring and Auditing

2.4.1 Description

Metrics, Monitoring, and Auditing form the observability foundation of the COGNIT Framework, providing comprehensive visibility into infrastructure health, resource utilization, application performance, power consumption. These capabilities serve dual purposes: enabling real-time operational decisions by the AI-Enabled Orchestrator, and supplying training data for machine learning models that predict workloads, optimize placements, and proactively adapt to changing conditions.

Traditional cloud monitoring systems focus on infrastructure metrics (CPU usage, memory consumption, network throughput) but lack visibility into edge-specific concerns: energy consumption, geolocation, latency characteristics, and application-level performance. COGNIT extends OpenNebula's native monitoring infrastructure with specialized components that capture these edge-critical metrics.

2.4.2 Architecture & Components

The monitoring architecture integrates metrics collection through the OpenNebula Monitoring System[2], which uses a probe-based architecture to collect metrics from various sources. COGNIT extends this system with custom probes for Scaphandre energy metrics and Serverless Runtime application metrics.

The **OpenNebula Monitoring System** provides a framework for collecting, storing, and accessing infrastructure and application metrics. The system uses monitoring probes that run on hypervisor hosts and collect metrics from various sources, storing them in OpenNebula's database and making them accessible through the OpenNebula API and metadata attributes.

Scaphandre Energy Monitoring: Scaphandre is deployed as a system daemon on each hypervisor host, continuously monitoring CPU power consumption using RAPL interfaces exposed by modern Intel and AMD processors (requires Linux kernel ≥ 6.0 for AMD Ryzen). Scaphandre correlates CPU time consumed by each process with overall power consumption, producing per-process power estimates in microwatts. For COGNIT, each Serverless Runtime VM is a distinct process, enabling per-VM energy attribution. A custom OpenNebula monitoring probe pulls metrics from the Scaphandre Prometheus exporter and integrates them into OpenNebula's monitoring system. The probe reads host-level and VM-level power consumption metrics, correlating them with OpenNebula VMs using the `one-<vm_id>` naming convention. These metrics are stored in OpenNebula's database and exposed through host and VM metadata attributes.

[2]https://docs.opennebula.io/7.0/product/cloud_system_administration/resource_monitoring/monitoring_system

Serverless Runtime Application Metrics Probe: Serverless Runtime VMs expose Prometheus exporters on port 9100 that report execution-specific metrics: function execution count, execution duration, parameter sizes, error rates, and queue depth (RabbitMQ `messages_ready` counts). A custom OpenNebula monitoring probe uses the QEMU Guest Agent to execute `curl localhost:9100/metrics` inside each Serverless Runtime VM, fetches the Prometheus metrics, and parses them to extract application-specific data.

Geocoder Integration: An OpenNebula hook triggered when hosts enter the MONITORED state. The hook extracts the host's public IP address, queries the geocoder API to obtain latitude and longitude coordinates, and updates the host's template with a `GEOLOCATION` attribute (e.g., `GEOLOCATION="60.1699,24.9384"` for Barcelona). This metadata is static, so it's only computed once during host registration.

2.4.3 Data Model

Monitoring data is structured hierarchically:

Host-Level Metrics:
- **CPU**: Total CPU utilization (%), per-core utilization, CPU frequency.
- **Memory**: Total memory (MB), used memory, cached memory, available memory.
- **Network**: Bytes transmitted/received, packet counts, error counts.
- **Disk**: I/O operations per second, read/write throughput, disk utilization.
- **Power**: Total host power consumption (microwatts), trend over time.
- **Geolocation**: Latitude and longitude coordinates (static metadata).

VM-Level Metrics:
- **Resource Usage**: CPU usage (%), memory usage (MB), disk I/O, network throughput.
- **Power Consumption**: Estimated VM power usage (microwatts), derived from Scaphandre process metrics.
- **Application Metrics**: Function execution count, execution duration histogram, error rate, current queue depth.
- **State**: VM lifecycle state (RUNNING, POWEROFF, SUSPENDED, UNKNOWN).

Monitoring data flows through several consumption paths:
1. **OpenNebula API and Metadata:** All collected metrics are stored in OpenNebula's database and accessible through the OpenNebula API. Metrics are also exposed as VM and host metadata attributes, enabling direct querying by other COGNIT components without requiring external monitoring systems.
2. **AI-Enabled Orchestrator Runtime Decisions**: The orchestrator queries OpenNebula's API and metadata for current metrics (CPU, memory, queue depth, power consumption) to inform placement decisions. For example, when deploying a new Serverless Runtime, the orchestrator selects the host with sufficient available capacity and lowest projected power consumption based on metrics collected by the monitoring probes.

3. **AI-Enabled Orchestrator Training Data**: Historical metrics stored in OpenNebula's database can be exported for ML model training. The extended OpenNebula AIOps SDK accesses these metrics to train models that predict future workloads based on historical patterns, enabling proactive scaling and optimization.
4. **Real-Time Operational Visibility**: Metrics collected by OpenNebula monitoring probes are accessible through OpenNebula's web interfaces and API, providing administrators and the AI-Enabled Orchestrator with real-time visibility into host health, VM status, application performance, and energy consumption trends.

2.5 [SR4.5] Authentication and Authorization

2.5.1 Description

Authentication and Authorization mechanisms secure access to COGNIT Framework resources, enforcing identity verification and access control policies across the distributed cloud-edge continuum. In the COGNIT architecture, authentication challenges are particularly complex due to the multi-tier, multi-tenant nature of the system: devices authenticate with the COGNIT Frontend, the Frontend authenticates with the Cloud-Edge Manager, and Edge Cluster Frontends needs to authenticate and authorize devices when executing functions.

Traditional username/password authentication is inadequate for this environment due to security vulnerabilities (password reuse, credential leakage), operational challenges, and architectural mismatches (stateless token-based systems scale better than session-based systems). COGNIT addresses these challenges through cryptographic token-based authentication using **Biscuit tokens**, a modern authorization token format that supports decentralized verification, attenuation (token capability restriction), and expiration policies.

Biscuit is a bearer token format designed for distributed systems; Biscuit tokens can be attenuated (restricted) by intermediate parties without access to the original signing key. This enables powerful delegation patterns: a token issued by the COGNIT Frontend can be attenuated by the Edge Cluster Frontend to restrict its capabilities before passing it to a Serverless Runtime.

2.5.2 Architecture & Components

The core component of the authentication system is the Biscuit Authentication Driver, implemented as a Ruby script that conforms to OpenNebula's authentication driver interface specification. OpenNebula's authentication system is designed with a modular architecture where external authentication drivers can be plugged in to support various authentication methods (LDAP, Active Directory, custom token systems, etc.). The Biscuit Authentication Driver leverages this extensibility to introduce Biscuit token verification into OpenNebula's authentication workflow.

The following components provide the authentication and authorization mechanism in COGNIT.

The **Biscuit Authentication** driver:

1. Receives authentication requests from OpenNebula's API server containing username and token.
2. Retrieves the user's public key from OpenNebula's database (stored in the password field).
3. Invokes the Biscuit CLI tool to verify the token signature against the public key.
4. Parses the Biscuit CLI output to extract verification results (valid/invalid, expiration status, embedded facts).
5. Returns success or failure to OpenNebula, which grants or denies API access accordingly.

The **Biscuit CLI** is a standalone executable that provides token generation, verification, inspection, and attenuation operations. COGNIT uses the CLI for:

- **Keypair Generation**: `biscuit keypair` generates Ed25519 private and public keys.
- **Token Creation**: `biscuit generate <private_key> <authority_block>` creates a new token with specified capabilities.
- **Token Verification**: `biscuit inspect <token> --public-key <public_key>` verifies token validity.
- **Token Attenuation**: `biscuit attenuate <token> <attenuation_block>` adds restrictions to an existing token.

The **COGNIT Frontend** exposes a `/v1/public_key` endpoint that returns its Biscuit public key. Edge Cluster Frontends fetch this key on startup and cache it in memory. When a device submits a function execution request, the Edge Cluster Frontend verifies the device's token using this public key, ensuring the token was issued by the COGNIT Frontend.

The authentication flow for a function execution request is as follow:

1. **Device Authentication**: Device sends credentials to COGNIT Frontend /v1/authenticate endpoint.
2. **Token Issuance**: Frontend validates credentials against OpenNebula's user database, generates Biscuit token, returns token to device.
3. **Application Requirements Registration**: Device sends application requirements to /v1/app_requirements with Biscuit token in header. Frontend verifies token, stores requirements.
4. **Cluster Assignment**: Device queries /v1/app_requirements/{id}/ec_fe with Biscuit token. Frontend verifies token, returns assigned Edge Cluster Frontend endpoint.
5. **Function Execution**: Device sends function execution request to Edge Cluster Frontend with Biscuit token. Frontend verifies token using COGNIT Frontend's public key, authorizes request, dispatches execution.

2.5.3 API & Interfaces

Authentication occurs at multiple API boundaries:

API Boundary	Authentication Method	Token Type
Device → COGNIT Frontend	HTTP Basic Auth	Username/password
COGNIT Frontend → Cloud-Edge Manager	Biscuit token in OpenNebula API credentials	Biscuit token (signed by Frontend)
Device → Edge Cluster Frontend	Biscuit token in HTTP token header	Biscuit token (issued by Frontend)
Edge Cluster Frontend → Serverless Runtime	(Internal, queue-based, no explicit auth)	Execution ID for result correlation

Table 2.5.1: Authentication Methods by API Boundary

2.6 [SR4.6] Plan Executor

2.6.1 Description

The Plan Executor is the operational component of the COGNIT Framework's AI-driven orchestration system, responsible for translating high-level optimization plans produced by the AI-Enabled Orchestrator into concrete infrastructure actions executed by the Cloud-Edge Manager. This component bridges the gap between smart decision-making and physical resource management, enabling the Framework to dynamically adapt infrastructure to changing workload demands, energy availability, and application requirements.

The Plan Executor was developed as part of the IPCEI-CIS project's new scheduler architecture for OpenNebula and integrated into COGNIT. It serves as the execution component of the MAPE-K architecture, enabling the proactive, AI-driven management capabilities described in the Introduction.

Plans are expressed as XML documents conforming to a predefined schema. Each plan targets a specific Edge Cluster and contains a sequence of actions to be performed on Serverless Runtimes. Actions include:

- **deploy**: Provision a new Serverless Runtime on a specified host.
- **migrate**: Live-migrate an existing Serverless Runtime to a different host within the same Edge Cluster.
- **poweroff**: Shut down a Serverless Runtime.

The Plan Executor processes plans sequentially, executing actions in order while handling dependencies and retries on transient failures.

2.6.2 Architecture & Integration

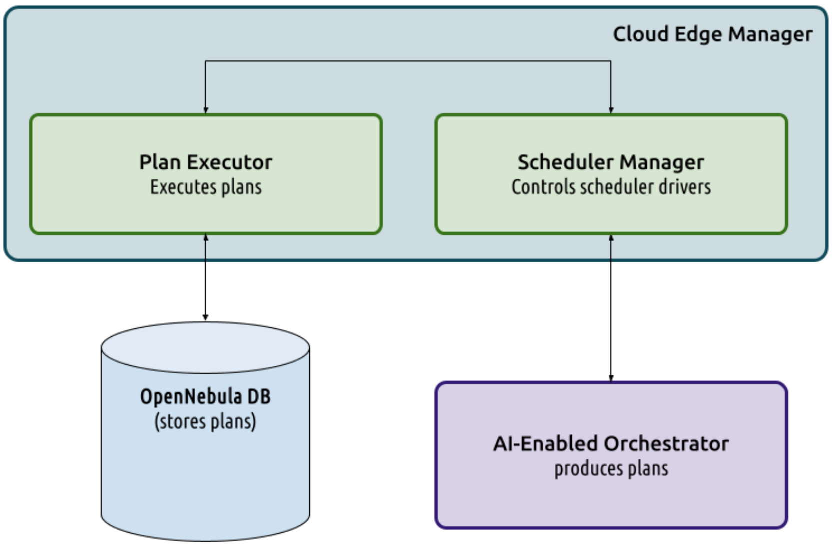

Figure 2.6.1: OpenNebula Resource Scheduler Framework Architecture

The Plan Executor is part of the OpenNebula Resource Scheduler Framework[3] (see Figure 2.6.1), which was developed in the IPCEI-CIS project. Within this framework, the Plan Executor receives optimization plans from the Scheduler Manager, which coordinates the scheduling process and invokes AI-Enabled Orchestrator drivers (schedulers) to generate plans. The Scheduler Manager and AI-Enabled Orchestrator drivers are described in detail in Section 3 (AI-Enabled Orchestrator).

The Plan Executor is the execution component that translates XML plans into concrete infrastructure actions. For each action in the plan:

1. **Validation**: Verifies that the target VM exists, the source/destination hosts are operational, and prerequisites are satisfied.
2. **Execution**: Invokes the appropriate OpenNebula API method:
   - deploy: VM.deploy(host_id, enforce=true)
   - migrate: VM.migrate(destination_host_id, live=true, enforce=true)
   - poweroff: VM.poweroff(hard=false)
3. **Monitoring**: Polls VM state until the action completes (state transitions to RUNNING for deploy, RUNNING on new host for migrate, POWEROFF for poweroff).
4. **Error Handling**: Retries transient errors (e.g., host temporarily unavailable), aborts plan on irrecoverable errors (e.g., insufficient capacity).

The Plan Executor enables the execution of several smart orchestration capabilities:

1. **Proactive Scaling**: The AI-Enabled Orchestrator predicts workload increases (based on historical patterns) and generates plans to deploy additional Serverless Runtimes before demand arrives, eliminating cold-start latency.

[3] https://docs.opennebula.io/7.0/product/cloud_system_administration/scheduler/overview

2. **Energy-Aware Rebalancing**: The orchestrator, based on the power consumption across hosts, generates migration plans to consolidate workloads onto fewer hosts, enabling idle hosts to enter low-power states

2.6.3 Data Model

Plans are XML documents with the following structure:

```xml
<PLAN>
  <ID>2</ID>  <!-- OpenNebula Cluster ID -->
  <ACTION>
    <VM_ID>423</VM_ID>
    <OPERATION>deploy</OPERATION>
    <HOST_ID>12</HOST_ID>
  </ACTION>
  <ACTION>
    <VM_ID>424</VM_ID>
    <OPERATION>migrate</OPERATION>
    <HOST_ID>15</HOST_ID>
  </ACTION>
  <ACTION>
    <VM_ID>425</VM_ID>
    <OPERATION>poweroff</OPERATION>
  </ACTION>
</PLAN>
```

In the following the plan XML schema:

XML Path	**Description**	**Required**
`PLAN/ID`	Edge Cluster identifier	Yes
`PLAN/ACTION`	Array of actions to execute	Yes (≥1)
`PLAN/ACTION/VM_ID`	Target Serverless Runtime VM ID	Yes
`PLAN/ACTION/OPERATION`	Operation type: `deploy`, `migrate`, `poweroff`	Yes
`PLAN/ACTION/HOST_ID`	Target host ID (required for `deploy` and `migrate`)	Conditional

Table 2.6.1: Plan XML Schema

3. AI-Enabled Orchestrator

The AI-Enabled Orchestrator is a critical component of the COGNIT Framework that provides smart resource management and optimization capabilities across the cloud-edge continuum. It addresses the fundamental challenge of efficiently managing distributed infrastructure resources while satisfying application requirements, minimizing energy consumption, and optimizing resource utilization. The orchestrator integrates AI/ML models for workload prediction with optimization algorithms for resource placement and scaling decisions.

It is structured around two main software requirements:

- **Building Learning Models (SR5.1)**: AI/ML models and predictive analytics based on metrics from the Cloud-Edge Manager monitoring service (Edge Clusters, Serverless Runtimes, infrastructure usage). These enable workload prediction, scaling recommendations, and device load forecasting, informing proactive resource allocation.
- **Smart Management of Cloud-Edge Resources (SR5.2)**: Multi-level optimization, global device-to-cluster assignments and scaling (Multi-Cluster Optimizer), and Serverless Runtime placement and workload balancing within Edge Clusters (Cluster Optimizer), using energy-aware policies.

Sections 3.1 and 3.2 below provide the detailed description of these two software requirements.

3.1 [SR5.1] Building learning models

3.1.1 Description

SR5.1 requires providing AI/ML models based on collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters, Serverless Runtimes, and infrastructure usage. These models enable predictive analytics for proactive resource management, allowing the COGNIT Framework to anticipate workload changes and optimize resource allocation before performance degradation occurs.

3.1.2 Architecture & Integration

The learning models capability is implemented through the integration of the OpenNebula AIOps SDK, developed in the IPCEI-CIS project, which provides AI-driven resource and application management for OpenNebula deployments.

In COGNIT, the AIOps SDK has been also extended with two main contributions:

1. **Extension to OneFlow Services**: Support for OneFlow Service-specific predictions, enabling the orchestrator to:
    a. **Predict Serverless Runtime Scaling Requirements**: Analyze historical CPU, memory, and queue depth metrics from OneFlow services to predict when scaling operations are needed

   b. **Workload Forecasting**: Use time-series analysis to forecast future resource demands for Serverless Runtimes within Edge Clusters
   c. **Performance Prediction**: Estimate response times and resource utilization based on current and predicted workload patterns
2. **Support for Integration of ML/AI Models**: While the AIOps SDK includes classical standard techniques (such as linear/rigde regression, ARIMA, Fourier analysis, and gradient boosting), it can support different AI/ML models, including the models developed in COGNIT[4].

The extended SDK integrates with Cloud-Edge Manager monitoring infrastructure to collect metrics from Edge Clusters, Serverless Runtimes, and infrastructure components. These metrics are processed through ML models to generate predictions that inform scaling and placement decisions.

The following subsections describe how these extensions are realised in the Python pyoneai library (the OpenNebula AIOps SDK implementation). The library introduces a clear entity model, monitoring accessors that aggregate metrics at Service and Cluster level, and a predictor accessor that feeds ML models with the appropriate time-series for scaling and placement.

3.1.2.1 OpenNebula AIOps SDK Extensions

To support predictions at the right granularity, both for individual Serverless Runtime VMs and for whole services or clusters, the SDK distinguishes several entity types.

Each entity is identified by an **EntityUID(type, id)**. The supported types are:

- `HOST` (physical hypervisor),
- `CLUSTER` (OpenNebula cluster, i.e. a group of hosts),
- `VIRTUAL_MACHINE` (e.g. a Serverless Runtime VM),
- `SERVICE` (OneFlow Service as a logical group of VMs),
- `SERVICE_ROLE` (a role within a OneFlow Service, such as "FaaS")

This model allows the orchestrator to request metrics and predictions either per VM or aggregated per service or per cluster, as needed for scaling and placement decisions.

Monitoring accessors implement a common interface for retrieving time-series metrics. The `BaseMonitoringAccessor` provides time-window expansion (with tolerance for the monitor interval), post-processing such as counter restoration, gap filling, resampling and interpolation, and metric-type handling (COUNTER, GAUGE, RATE, HISTOGRAM). On top of this, two aggregating accessors supply Service- and Cluster-level views.

The `ServiceAggregatingAccessor` exposes OneFlow Service and Service Role metrics by aggregating VM-level metrics (i.e. Serverless Runtime metrics). It is configured with a VM monitoring backend and a service topology that maps each service to its roles and the list of VM IDs per role.

[4] https://github.com/SovereignEdgeEU-COGNIT/ai-models/tree/main/ml-models

For a given SERVICE or SERVICE_ROLE entity, the accessor resolves the corresponding VMs, fetches their time-series from the underlying VM accessor, and aggregates them along the entity dimension.

The aggregation rule depends on the metric type:

- counters are summed,
- gauges are averaged by default
- rates are summed;
- custom aggregation (min, max, median) can be specified per metric.

The service topology can be updated at runtime via `update_service_topology(service_id, roles)` and `get_service_topology(service_id)` without recreating the accessor, which is important when Serverless Runtimes scale and the set of VMs in a service changes.

The `ClusterAggregatingAccessor` plays the same role at cluster level; it aggregates host-level metrics according to a cluster topology (cluster ID to list of host IDs). It uses the same aggregation rules and exposes `update_cluster_topology(cluster_id, host_ids)` and `get_cluster_topology(cluster_id)` for dynamic updates. Thus, the AI-Enabled Orchestrator can obtain cluster-wide views (e.g. average CPU or total power) for capacity and placement decisions.

The PredictorAccessor wraps a prediction model and uses an observator accessor (for example a Service or Cluster aggregating accessor) to retrieve historical time-series. It requests a history window whose length is determined by the model's sequence length and the requested time resolution, then invokes the model's predict() method to produce forecasts for the requested instant or period. In this way, the orchestrator can obtain Service- or Cluster-level predictions (e.g. aggregate CPU or queue depth) directly, so that scaling and placement logic can work at the appropriate granularity without duplicating aggregation logic.

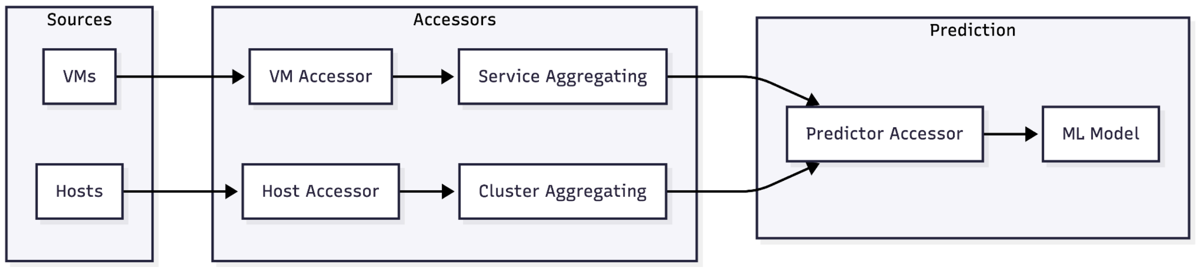

Figure 3.1.1. Data flow from monitoring sources through accessors to the predictor and ML model.

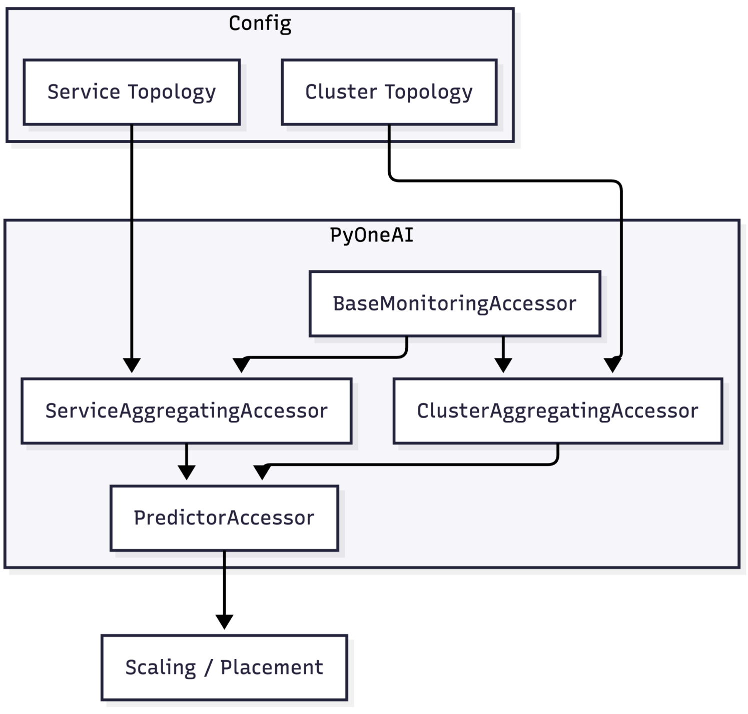

Figure 3.1.2. Architecture of the pyoneai extension: topologies, accessors, and predictor feeding scaling and placement decisions.

Figure 3.1.1 illustrates the conceptual data flow from raw sources (hosts and VMs) through the accessors to the predictor and ML model. Figure 3.1.2 summarises the architecture: configuration (service and cluster topologies) drives the aggregating accessors, which in turn feed the predictor accessor; the predictions are then supplied to the optimizers (Multi-Cluster Optimizer and Cluster Optimizer), which use them, rather than observations alone, to produce scaling and placement plans.

3.1.2.2 Device Load Forecasting

The **Device Load Forecasting** component provides workload estimation for individual devices by aggregating system-wide metrics from OneFlow services. This component:

- **Periodically collects metrics** from all OneFlow services deployed across Edge Clusters, including CPU usage and queue depth metrics from Edge Cluster Frontends and Serverless Runtimes.
- **Predict estimated load** for each registered device based on total CPU usage and backlog.
- **Updates device assignments** in the shared database with current estimated load values, enabling the multi-cluster optimizer to make informed decisions about device-to-cluster assignments.

The component operates as a standalone daemon that periodically queries OpenNebula's REST API and OpenNebula database to fetch OneFlow service metrics, processes them to calculate device-level load estimates, and updates the shared SQLite database that is used by both the COGNIT Frontend and the multi-cluster optimizer.

3.1.2.3 Integration within COGNIT Framework

The learning models integrate within COGNIT Framework through several key interfaces:

1. **Metrics Collection**: The extended AIOps SDK collects metrics from Cloud-Edge Manager Monitoring system, which monitors Edge Clusters, Serverless Runtimes, and infrastructure components. These metrics include CPU utilization, memory usage, network I/O, queue depths, and energy consumption (via Scaphandre integration).
2. **Prediction Generation**: ML models process collected metrics to generate predictions for Serverless Runtime scaling requirements. These predictions are used by the orchestrator to proactively scale Serverless Runtimes before performance degradation occurs.
3. **Device Database Integration**: The Device Load Forecasting component updates device load estimates in the shared database, which is accessed by the COGNIT Frontend for initial device-to-cluster assignments and by the multi-cluster optimizer for global optimization.
4. **OneFlow Service Integration**: Predictions are specifically tailored for OneFlow Services, enabling the orchestrator to understand the relationship between workload patterns and Serverless Runtime cardinality requirements across different use cases (SmartCity, WildFire, Energy, Cybersecurity).

3.2 [SR5.2] Smart Management of Cloud-Edge Resources

3.2.1 Description

SR5.2 requires the AI-Enabled Orchestrator to manage and optimize the lifecycle of Edge Clusters and Serverless Runtimes within Edge Clusters according to application requirements, infrastructure and virtual resource usage, and energy-aware policies. This capability enables COGNIT to dynamically adapt resource allocation across the cloud-edge continuum, balancing performance, energy consumption, and cost objectives.

3.2.2 Architecture

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

The **Multi-Cluster Optimizer** operates at the global level: it assigns devices to Edge Clusters and determines how many Serverless Runtimes (VMs) each cluster should run, with the aim of minimizing carbon footprint across the cloud-edge continuum while respecting application requirements.

The **Cluster Optimizer** operates at the level of each Edge Cluster: given the set of Serverless Runtimes (VMs) in a cluster, it decides which host each VM runs on (initial placement) and periodically rebalances VMs across hosts (workload optimization), minimizing energy usage, i.e. power consumption and resource contention within the cluster.

Together, the Multi-Cluster Optimizer drives which clusters serve which devices and at what capacity; the Cluster Optimizer then ensures that, within each cluster, VMs are placed on hosts in an energy-efficient and contention-aware way. Figure 3.2.0 gives a high-level view of the two components and their scope.

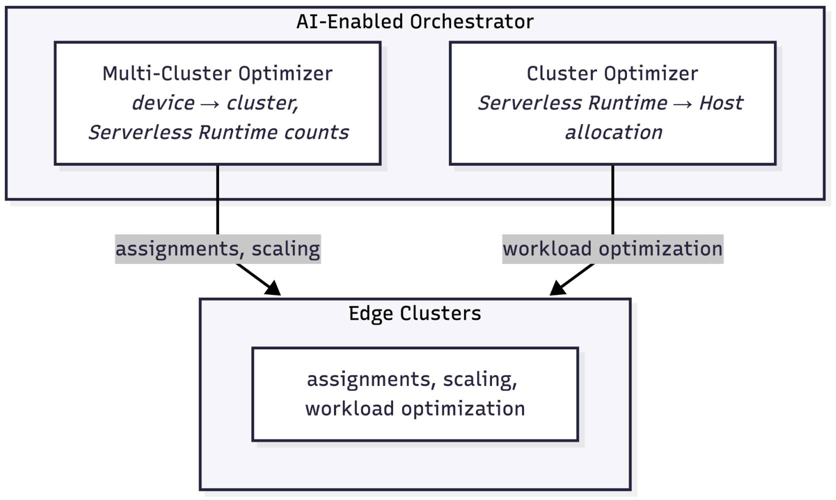

Figure 3.2.1. AI-Enabled Orchestrator: Multi-Cluster Optimizer and Cluster Optimizer driving Edge Clusters (assignments, scaling, workload optimization).

3.2.2.1 Multi-Cluster Optimizer

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

- **Reads device-related data** from the shared database, including device IDs, current cluster assignments, application requirements, and estimated load values calculated by the Device Load Forecasting component.
- **Fetches application requirements** from the Cloud-Edge Manager using application requirement IDs, including constraints such as flavour, confidentiality requirements, provider preferences, and geolocation constraints.
- **Filters feasible clusters** for each device based on application requirements, ensuring that only clusters that can satisfy device constraints are considered.
- **Runs Mixed-Integer Linear Programming (MILP) optimization** to find optimal cluster assignments that minimize carbon footprint across all clusters, while avoiding performance degradation.
- **Triggers cluster scaling operations** in parallel based on optimization results, calling Edge Cluster Frontend scaling endpoints to adjust Serverless Runtime cardinality.
- **Updates device assignments** in the database as clusters finish scaling, ensuring consistency between optimization decisions and actual resource allocation.

The Multi-Cluster Optimizer minimizes the greenhouse gas emissions, obtained by combining carbon intensity and power consumption across clusters. It adjusts the device-to-cluster assignments and the number of used Serverless Runtimes, according to the available resources and predicted requirements (including predictions supplied by the learning models described in Section 3.1).

First, it tries to perform allocation without any resource contention, if feasible in any way, keeping the clusters running with high performance and additional available capacities. When that is not feasible, a second optimization run allows contention and penalises it in the objective so that the solution remains acceptable.

The Multi-Cluster Optimizer operates as a daemon that periodically runs optimization cycles, reading current device assignments, computing optimal reassignments, and executing scaling operations. It considers multiple objectives simultaneously: minimizing energy consumption, reducing carbon footprint, and satisfying application requirements.

The data model used by the Multi-Cluster Optimizer is as follows.

- Each Cluster is characterised by an identifier (ID), current capacity (VM count), maximum capacity (e.g. total number of CPU cores), a piecewise linear energy model given as breakpoints (cpu_usage, power), and a carbon intensity value. Cluster energy breakpoints are derived from host-level CPU_ENERGY and NUMA/contention data; carbon intensity is taken from the cluster template (e.g. CARBON_INTENSITY).
- Each Device is characterised by an identifier, a load value (used for the energy model), a capacity load used for capacity constraints, and the list of feasible cluster IDs determined from application requirements.

Formally, the optimization problem is a MILP. The inputs to the MILP problem are the set of all clusters *C* and the set of all devices *D*, with their data, as described above.

The decision variables are:

- Binary assignment variables x_{ij}, which determine whether the device D_i is assigned to the cluster C_j, from the list of the suitable clusters for the device
- Integer variables n_j, which determine the number of VMs assigned to the cluster C_j, according to x_{ij} and predefined VM capacity

The main constraints of the MILP problem are:

- Each device is allocated to exactly one cluster: $\sum_i x_{ij} = 1$, for each cluster C_j
- The total load (i.e. capacity load) of all devices assigned to a cluster cannot be higher than the capacity of the cluster: $\sum_i x_{ij} l_i \le c_j$, for each cluster C_j, where c_j is the capacity of the cluster C_j

- The number of allocated VMs on a cluster correspond to the number and capacity load of the devices, for example: $\sum_i x_{ij} l_i \le n_j < \sum_i x_{ij} l_i + 1$, for each cluster C_j, if each VM has only one CPU core
- The total used capacity of all clusters is between the predefined lower and upper bounds (c_{lb} and c_{ub}), if these bounds are provided: $c_{lb} \le \sum_j n_j \le c_{ub}$

The objective function is the minimization of the carbon footprint, i.e. greenhouse gas emission, expressed as the product of the carbon intensity of the cluster (Γ_j) and the power consumption of the cluster (P_j), summed across all clusters:

$$min \sum_{j, C_j \in C} \Gamma_j P_j$$

The power consumption of a cluster C_j is determined as a piecewise linear function of the total CPU usage of all devices assigned to that cluster, $P_j(\sum_i x_{ij} w_i)$, where w_i is the load (i.e. CPU usage that corresponds to the device D_i). This objective function is further penalized if the CPU contention exists.

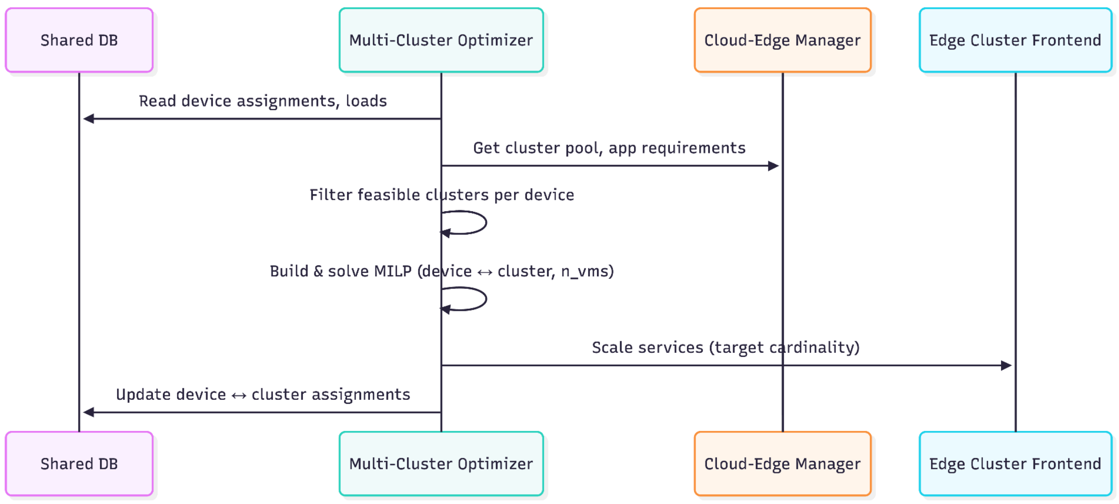
Shared DB
Multi-Cluster Optimizer
Cloud-Edge Manager
Edge Cluster Frontend
Read device assignments, loads
Get cluster pool, app requirements
Filter feasible clusters per device
Build & solve MILP (device ↔ cluster, n_vms)
Scale services (target cardinality)
Update device ↔ cluster assignments
Shared DB
Multi-Cluster Optimizer
Cloud-Edge Manager
Edge Cluster Frontend

Figure 3.2.2. Multi-Cluster Optimizer: flow from database and Cloud-Edge Manager through MILP optimization to scaling and database update.

The implementation builds the MILP optimization problem with the PuLP library (class `DeviceOptimizer`) and uses the two-step contention strategy in the method `DeviceOptimizer.optimize` and wrapper function `optimize_contention()`. It first builds Device and Cluster lists from the database and the Cloud-Edge Manager, then calls the optimizer, and finally triggers scaling and database updates. Figure 3.2.2 summarises the flow from the shared database and Cloud-Edge Manager through the optimizer to scaling and back to the database.

3.2.2.2 Cluster Optimizer

The **Cluster Optimizer** is a component that addresses resource management at the cluster level, optimizing workload distribution within individual Edge Clusters. The Cluster Optimizer implements **energy-aware** optimization algorithms, using Mixed-Integer Linear Programming (MILP) optimization that simultaneously minimizes power consumption and resource contention when placing Serverless Runtimes on hosts within a cluster.

The Cluster Optimizer:

- **Uses power consumption metrics** to make energy-efficient placement decisions
- **Evaluates resource contention** by analyzing co-location of Serverless Runtimes on the same host, predicting interference effects that could degrade performance
- **Considers host capacity** and current utilization when making placement decisions, ensuring efficient resource utilization while minimizing energy consumption
- **Implements** initial placement and workload optimization within a cluster:
    - **Initial placement:** When new Serverless Runtimes are instantiated, the Cluster Optimizer selects the optimal host based on capacity and requested resources (this happens during scaling operations).
    - **Workload optimization:** Periodically generates energy-aware optimization plans to redistribute Serverless Runtimes across hosts within a cluster, minimizing resource contention and power consumption and reducing energy usage.

The energy-aware algorithm implemented in the Cluster Optimizer achieves energy consumption minimization by simultaneously addressing two complementary objectives: power consumption minimization and resource contention reduction. The energy-aware policy combines two optimization strategies in a dual-objective approach:

1. **Power Consumption Minimization**: Minimizes the total power consumed by all hosts in the cluster by intelligently distributing workloads based on host power efficiency characteristics.
2. **Resource/CPU Contention Reduction**: Reduces overall resource contention across hosts and related performance degradation. By minimizing CPU contention, the policy ensures that workloads run efficiently, which in turn contributes to energy efficiency since contention leads to wasted CPU cycles and increased power consumption.

The Cluster Optimizer analyzes the power consumption profile of each host, using RAPL (Running Average Power Limit) measurements via Scaphandre integration; evaluates resource contention patterns, particularly CPU contention, across hosts; and considers the current CPU usage and workload distribution. It generates optimization plans that simultaneously: (a) consolidate workloads onto the most power-efficient hosts, (b) minimize resource contention to ensure efficient CPU utilization, and (c) minimize the number of active hosts when possible, allowing less efficient or idle hosts to be powered down.

By minimizing both power consumption and resource contention simultaneously, the energy-aware algorithm achieves superior energy efficiency: lower overall power usage and reduced contention ensure effective CPU utilization and maintained performance, with fewer active hosts when possible and a lower carbon footprint.

The Cluster Optimizer uses a specific two-step approach that enables the consideration of both resource contention and power consumption:

- In the first step, it applies fuzzy programming and finds the minimal possible resource contention.
- In the second step, it minimizes the power consumption, by constraining the resource contention to its previously established minimum.

The following subsections report the input variables, decision variables, and objective functions of the mathematical model used by the Cluster Optimizer. The formulation implements the two-step approach described above: resource contention is expressed via fuzzy satisfaction (maximized first), then power consumption is minimized subject to that contention level.
The inputs to the mathematical model[5] represent the requirements of all considered VMs (i.e. Serverless Runtimes), and capacities of the hosts of the Edge Cluster:

- Set of all VMs which need to be allocated to the physical infrastructure, $\mathbb{V} = \{V_1, V_2, ..., V_n\}$
- Set of all physical nodes, i.e. hosts available to the infrastructure, $\mathbb{H} = \{H_1, H_2, ..., H_m\}$

Each VM has the following input variables:

- v_i^m: the amount of memory requested for V_i, $\forall i = 1, 2, ..., n$
- v_i^c: the number of CPU cores requested for V_i, $\forall i = 1, 2, ..., n$
- v_i^{cu}: the CPU usage of V_i, $\forall i = 1, 2, ..., n$

Each physical node has the following input variables:

- h_j^m: the available memory on H_j, $\forall j = 1, 2, ..., m$
- h_j^c: the available number of CPU cores on H_j, $\forall j = 1, 2, ..., m$

[5] The energy-aware algorithms were partially defined in the OneEdge5G project for the initial placement. In COGNIT, the algorithms have been modified and completely integrated for both initial placement and workload optimization.

The combinations of VMs and physical nodes are taken into account with:

- $\widehat{x}_{ij}$: the current allocation of V_i, which is equal to 1 if V_i is currently allocated to H_j and 0 otherwise, $\forall i = 1, 2, ..., n$, $\forall j = 1, 2, ..., m$
- $\mathbb{S}_i$: the set of the indices of all suitable hosts for a virtual machine V_i, $\forall i = 1, 2, ..., m$, i.e. the nodes that are able to satisfy all the requirements of V_i, $\mathbb{S}_i \subseteq \{1, 2, ..., m\}$
- $\mathbb{F}_j$: the set of the indices of all VMs for which the allocation to H_j might be a feasible solution, $\forall j = 1, 2, ..., m$, i.e. whose requirements can be satisfied by H_j, $\mathbb{F}_j \subseteq \{1, 2, ..., n\}$

The main decision variables indicate the VM allocation related to the physical infrastructure:

- x_{ij}: the allocation of V_i to H_j, which is equal to 1 if V_i will be allocated to H_j and 0 otherwise, $\forall i = 1, 2, ..., m$, $\forall j \in \mathbb{S}$.
- y_j: 1 if H_j will be used and 0 otherwise, $\forall j = 1, 2, ..., m$.

Resource contention is implemented using the fuzzy programming approach, which maximizes the overall satisfaction related to resource usage. The fuzzy membership functions are defined in such a way that the level of satisfaction related to a physical node H_j and metric q (CPU Usage), μ_j^q, is equal to 1 if the utilization, v_i^q, is below a predefined threshold, t_j^q, and drops linearly towards zero otherwise:

$$\mu_j^q = 1 \text{ if } v_i^q \leq t_j^q \text{ else } \mu_j^q = \frac{h_{j,max}^q - \sum_{i \in \mathcal{F}_j} x_{ij} v_i^q}{h_{j,max}^q - t_j^q}$$

To achieve this, a new continuous nonnegative decision variable μ_j^q is added for each node H_j and quantity q, which represents the satisfaction level and obeys the constraints:

$$0 \leq \mu_j^q \leq 1$$

$$\mu_j^q \leq \frac{h_{j,max}^q - \sum_{i \in \mathbb{F}_j} x_{ij} v_i^q}{h_{j,max}^q - t_j^q}$$

The objective is to maximize the overall satisfaction level:

$$max\, \lambda$$

$$\lambda \leq \mu_j^q, \ \forall j = 1, 2, ..., m$$

The power consumption of a node is approximately expressed as a piecewise linear function of the CPU usage of all VMs allocated to that node. The piecewise linear function is defined for each node H_j, $\forall j = 1, 2, ..., m$, with a set of breakpoints whose coordinates are $(h^{cu}_{j,b}, e_{j,b})$, $\forall b = 0, 1, ..., p_j$ where:

- $h^{cu}_{j,b}$ is the CPU usage of H_j for the breakpoint b
- $e_{j,b}$ is the corresponding power consumption
- p_j is the number of segments (pieces) of the power piecewise linear function for H_j

To integrate power piecewise linear functions into the model, two sets of auxiliary decision variables are added for each node H_j:

- $z_{j,b}$, $\forall b = 1, 2, ..., p_j$: binary SOS2 decision variables that indicate whether the point whose coordinates are the actual host CPU usage and power consumption is on the segment b, i.e. between the breakpoints $(h^{cu}_{j,b-1}, e_{j,b-1})$ and $(h^{cu}_{j,b}, e_{j,b})$:

$$z_{j.b} = 1 \text{ if and only if } h^{cu}_{j,b-1} \le \sum_{i\in\mathbb{F}_j} x_{ij} v^{cu}_i \le h^{cu}_{j,b} \text{ and } e^{cu}_{j,b-1} \le \sum_{i\in\mathbb{F}_j} E_j \le e^{cu}_{j,b}$$

- $s_{j,b}$, $\forall b = 1, 2, ..., p_j$: nonnegative continuous decision variables, i.e. segment variables, that show the exact location of the point on a segment

If the node H_j is running, exactly one of the variables $z_{j,b}$ must be 1 and all others must be 0. Otherwise, all indicator variables must be 0. This is ensured by adding the following constraint:

$$\sum_{b=1}^{p_j} z_{j,b} = y_j,\ \forall j = 1, 2, ..., m$$

The segment variables are bounded from above with their corresponding indicator variables:

$$s_{b,j} \le z_{b,j},\ \forall j = 1, 2, ..., m,\ \forall b = 1, 2, ..., p_j$$

The following constraints connect the indicator and segment variables with the CPU usage and power consumption for each host:

$$\sum_{i\in\mathcal{F}_j} x_{ij} v^{cu}_i = \sum_{b=1}^{p_j} z_{j,b} h^{cu}_{j,b-1} + s_{j,b}(h^{cu}_{j,b} - h^{cu}_{j,b-1}),\ \forall j = 1, 2, ..., m$$

$$E_j = \sum_{b=1}^{p_j} z_{j,b} e^{cu}_{j,b-1} + s_{j,b}(e^{cu}_{j,b} - e^{cu}_{j,b-1}),\ \forall j = 1, 2, ..., m$$

Finally, the total power consumption of all considered physical nodes from $\mathbb{H}$ is:

$$E_{tot} = \sum_{j=1}^{m} E_j$$

The strength of this approach comes from the fact that, in many cases, reasonably low resource contention might correspond to a number of different VM allocation combinations, possibly with a wide range of power consumption values. It minimizes resource contention, thus allowing good runtime performance when possible, while preserving the power consumption as low as possible for the target contention level.

The optimization plans generated by the Cluster Optimizer are executed by the Plan Executor (described in Section 2.6), which handles VM deployment and migration operations.

3.2.3 Integration

The smart resource management components integrate with COGNIT through multiple interfaces:

1. **Database Integration**: Multi-cluster optimizer accesses the shared SQLite database to read device assignments and application requirements. The multi-cluster optimizer updates cluster assignments after optimization.
2. **Cloud-Edge Manager Integration**: The multi-cluster optimizer queries the Cloud-Edge Manager to fetch application requirements and cluster metadata. The cluster optimizer integrates directly with OpenNebula's scheduler framework, receiving placement requests and optimization triggers from the OpenNebula core daemon.
3. **Edge Cluster Frontend Integration**: The multi-cluster optimizer triggers scaling operations by calling Edge Cluster Frontend scaling endpoints (/v1/scale?target_cardinality=N), which adjust Serverless Runtime cardinality within OneFlow services.
4. **OneFlow Service Integration:** Both optimizers work with OneFlow Services, understanding the relationship between Serverless Runtime cardinality and workload capacity.
5. **Energy- and Carbon-Aware Policies**: Both optimizers incorporate energy-aware policies, with the multi-cluster optimizer preferring clusters with lower carbon intensity values that usually correspond to higher renewable energy percentages, and the cluster optimizer minimizing energy consumption through efficient host utilization and workload consolidation.

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

4. Conclusions

This deliverable presents the final implementation and integration of the AI-enabled Distributed Serverless Platform and Workload Orchestration components of the COGNIT Framework, completing all software requirements defined for Work Package 4. The work described in this document represents a comprehensive achievement in building an autonomous and smart infrastructure management system for the cloud-edge continuum.

The implementation successfully realizes a complete MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) autonomic computing architecture that enables COGNIT to proactively manage and optimize resources across distributed cloud-edge infrastructure.

The Cloud-Edge Manager provides a robust foundation for multi-provider infrastructure management through the Provider Catalogue, enabling seamless integration of resources from public cloud providers, on-premise infrastructures, and edge locations. Edge Cluster provisioning through declarative infrastructure-as-code approaches ensures consistent, reproducible deployments across heterogeneous environments. Serverless Runtime lifecycle management via OneFlow Services enables dynamic scaling, migration, and optimization of execution workloads. Comprehensive monitoring and metrics collection provides visibility into infrastructure health, resource utilization, energy consumption, and application performance. Biscuit token-based authentication and authorization mechanisms provide secure, decentralized access control across the distributed system. The Plan Executor bridges the gap between AI-driven optimization decisions and concrete infrastructure operations, enabling autonomous resource management.

The AI-Enabled Orchestrator delivers smart resource management through predictive analytics and multi-level optimization. Extensions to the OpenNebula AIOps SDK enable OneFlow Service-specific predictions for Serverless Runtime scaling requirements, workload forecasting, and performance prediction. The Device Load Forecasting component provides workload estimation for individual devices, enabling informed device-to-cluster assignment decisions. Multi-level optimization through the Multi-Cluster Optimizer and Cluster Optimizer addresses resource management at both global and cluster levels, incorporating carbon-aware and energy-aware policies to minimize carbon footprint and energy consumption while satisfying application requirements.

A key contribution of the work is the integration of energy-aware and carbon-aware optimization policies throughout the resource management system. The Multi-Cluster Optimizer minimizes greenhouse gas emissions by considering carbon intensity values when making device-to-cluster assignments, preferring clusters with higher renewable energy percentages. The Cluster Optimizer minimizes energy usage through efficient host utilization and workload consolidation, using power consumption metrics collected via Scaphandre integration. This dual-level approach to energy optimization enables COGNIT to address sustainability objectives at both global (cluster selection) and local (Serverless Runtime placement) levels.