ADV 5G TWN

NEARBY COMPUTING

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ADV5G-TWINS- ORCHESTRATION

Final Summary Deliverable

Version 1.0

Main authors	Maria A. Serrano, Berend Gort, Godfrey M. Kibalya, Angelos Antonopoulos, Universitat Politècnica de València
Deliverable Type	Document
Delivery date	22/10/2025

Avanzando-5G-Gemelos Digitales. Plataforma 5G-IoT con computación en el borde y la nube (TSI-063000-2021-112)

Financiado por la Unión Europea NextGenerationEU

MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA

UNICO I+D

Plan de Recuperación, Transformación y Resiliencia

Project Details

Project Title	Advanced 5G infrastructure and edge computing for logistics digital twins
Acronym	ADV5G-TWINS- ORCHESTRATION
Activity title	Dynamic, distributed orchestration of compute resources for digital twins and Energy-efficient orchestration of compute resource
Code	A8 and A7
Awardee	Nearby Computing
Start Date	29/09/2023
End date	30/09/2025
File No.	MY23/ITEAM/SE/19

Description

This is a deliverable summarizing the core activities and contributions that have been undertaken throughout the execution of ADV5G-TWINS-ORCHESTRATION project. The deliverable captures outcome of both activities A8 related to orchestration and A7 related to energy efficiency.

Versions and contributions

Versions

#	Description	Taxpayers
0.1	Initial version	Nearby Computing

Taxpayers

Partner	Authors
Nearby Computing	Lazaros Liatsas, Berend Gort, Godfrey M. Kibalya, Angelos Antonopoulos, Maria A. Serrano

Reviewers

Proofreader	Partner
María Crespo Aguado	UPV

Abbreviations

AGCRN	Adaptive Graph Convolutional Recurrent Network
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
E2E	End to End
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
ML	Machine Learning
NBI	Northbound Interface
QoS	Quality-of-Service
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SBI	Southbound Interface
SLO	Service Level Objective
SMAPE	Symmetric Mean Absolute Percentage Error
UPV	Polytechnic University of Valencia

Table of contents

Table of Figures

1. Introduction

ADV5G-TWINS-ORCHESTRATION project focuses on the design and implementation of orchestration techniques that support digital twin applications leveraging advanced 5G and 6G technologies in industrial and logistics environments. Towards this goal, several innovations and technical activities were executed targeting orchestration performance and Energy efficiency of applications and the digital-twin platform.

Activity 8 (A8) of the ADV5G-TWINS-ORCHESTRATION project focuses on Dynamic, distributed orchestration of compute resources for digital twins and while Activity 7 (A7) focusses on Energy-efficient orchestration of compute resource in IoT-edge-cloud platforms for real-time digital twins.

This deliverable captures a summary of the core project realizations for both of the activities.

1.1. Objective of this document

This document summarizes the activities and contributions that have been undertaken throughout the execution of the ADV5G-TWINS-ORCHESTRATION project for both A7 and A8.

1.2. Structure of the document

The remainder of this document is organized as follows: Section 2 captures the dissemination activities related to the ADV5G-TWINS-ORCHESTRATION project including scientific publications, project deliverables and participation in events. The core technical contributions are presented in section 3. Section 4 documents the Final demonstration activities and the deliverable is concluded in section 5.

2. Dissemination

In the context of the ADV5G-TWINS-ORCHESTRATION project, we have worked toward scientific publications, while we have also participated in important events to disseminate the concepts and our technical contributions. In addition, a number of technical deliverables have been submitted. The current list of achievements regarding scientific publications, events and deliverables are captured below.

2.1. Scientific Publications

Efforts have been dedicated to advancing scientific knowledge including conference papers and journal submissions, that reflect our work on cutting-

edge topics related to edge-cloud orchestration. A list of both conference and Journal contributions is given below.

1. B. Gort, G. Kibalya, A. Umbert, A. Antonopoulos, **OmniFORE: Attention-Based Generalization Framework for Edge-Cloud Workload Predictions,** IEEE ISCC 2025, 2-5 July 2025, Bologna, Italy.
2. L. Liatsas, G. Kibalya, A. Antonopoulos, **Explainable AI for Resource-Efficient Microservice Management in Cloud-Native Networks**, IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), 26-29 May 2025, Barcelona, Spain.
3. B. Gort, G. Kibalya, A. Antonopoulos, **Optimization of Distinct Time-Series Neural Architectures for Cloud-Edge Workload Prediction,** IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), 26-29 May 2025, Barcelona, Spain.
4. L. Liatsas, G. Kibalya, A. Antonopoulos, **XAI-driven Model Design for Resource Utilization Forecasting in Cloud-native 6G Networks,** IEEE MeditCom, 8-11 July 2024, Madrid, Spain
5. B. Gort, G. Kibalya, A. Antonopoulos, **AERO: Adaptive Edge-Cloud Orchestration with a Sub-1K-Parameter Forecasting Model,** IEEE Transactions on Machine Learning in Communications and Networking (2025)
6. B. Gort, G. Kibalya, M. Serrano, A. Antonopoulos**, Forecasting Trends in Cloud-Edge Computing: Unleashing the Power of Attention Mechanisms**, IEEE Communications Magazine (2025)
7. B. Ojaghi, M. M. Dehshibi, A. Antonopoulos, **A Supervised Active Learning Method for Identifying Critical Nodes in IoT Networks**, Journal of Supercomputing (2024)

2.2. Participation in events

We have participated in a number of important events to disseminate the concepts and our technical contributions. These include:

- Participation in the Mobile World Congress (MWC), Barcelona, Spain,.2025
- Participation in the Mobile World Congress (MWC), Barcelona, Spain, 26-29 Feb.2024
- ICMLCN 2025 Barcelona
- EUCNC 2025

2.3. Project deliverables

A number of deliverables have been documented and submitted throughout the project execution targeting both orchestration performance and Energy efficiency as related to activities A8 and A7 respectively. These are documented below.

2.3.1. Activity A8 deliverables

Activity A8 of the project realized the following deliverables.

- A8-D2.1 - Arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales. In this deliverable, the state of art orchestration algorithms with focus on machine learning based algorithms for the IoT-edge-cloud continuum are introduced. In addition, the deliverable discusses the requirements for the NearbyOne orchestrator to manage the lifecycle of digital twin services and the edge computing infrastructure.
- A8-D4.1 - Planificación del laboratorio y de los pilotos de las plataformas 5G IoT-edge-cloud: This deliverable introduced the proposed solution that addresses the imperative need for orchestrating IoT-edge-cloud architectures within digital twin environments. At the heart of the introduced solution is the NearbyOne Orchestrator, an intelligence driven platform designed to seamlessly integrate and manage the complex interplay between IoT devices, edge computing infrastructure, and cloud resources. In addition, the details of the NearbyOne orchestrator and its requirements are discussed in this deliverable.
- A8-D2.2 - Evolución de la arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales: The deliverable describes the proposed zero-touch closed loop orchestration framework where NearbyOne, presented in Deliverable *D4.2* is the central component. To support AI/ML-driven orchestration, two frameworks are also presented in deliverable D2.2: the observability stack, for real-time system metrics scraping, data storage and visualization across the distributed edge to cloud infrastructure, which can be leveraged to create datasets; and the ML framework to efficiently manage the lifecycle of machine learning models, from design and development to deployment. Finally, the proposed AI/ML algorithms enabling an efficient orchestration of computing resources and services are also Introduced.
- A8-D2.3 - Informe final de la arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales: This deliverable documents the S2.2 software. This deliverable contributes to the proposed orchestration solution for the IoT-edge-cloud platform, and to documenting the work with the IoT-edge-cloud platform of the PLATFORM-A8 activity. In addition, the deliverable describes the dataset that can be leveraged by the proposed AI/ML algorithms. The deliverable also presents the final design of the AI/ML algorithms and how the proposed algorithms can be leveraged for orchestration performance.

- A8-D4.2 - Puesta en marcha de las plataformas 5G IoT-edge-cloud en el laboratorio y en los pilotos de gemelos digitales. This deliverable details the implementation of the 5G IoT-edge-cloud platforms in the iTEAM-UPV laboratory and in the pilots, integrating, testing and validating the different solutions and components available in the project, documenting the P4.1 prototype and the VT4.1 validation pilot. A closed loop orchestration pilot demo that integrates and validates all the orchestration components is also presented in this deliverable including the results of the demo.
- A8-D4.3 - Pruebas finales del laboratorio y pilotos de gemelos digitales en tiempo real con plataformas 5G IoT-edge-cloud. This deliverable documents the setup and results of the real-time digital twin pilots with 5G IoT-edge-cloud platforms in relation to activity A8 of the project.

In addition to the above technical deliverables, the following deliverables related to software/demo/prototype were accomplished and submitted.

- A8-S2.1 - Algoritmos de orquestación de plataformas IoT-edge-cloud
- A8-S2.2 - Algoritmos basados en AI/ML para la orquestación de plataformas IoT-edge-cloud
- A8-P4.1 - Creación del laboratorio y piloto de gemelos digitales para las plataformas 5G IoT-edge-cloud
- A8-P4.2 - Actualización de las plataformas 5G IoT-edge-cloud para el laboratorio y los pilotos
- A8-VT4.1 - Primera fase de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud
- A8-VT4.2 - Fase final de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud

2.3.2. Activity A7 deliverables

The following technical deliverables related to activity A7 were submitted.

- A7-D2.1 - Arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales. This deliverable discusses the energy efficiency aspect of the IoT-edge-Cloud platform. The deliverable gives state-of-the-art IoT-edge-cloud architecture and the Telco architecture discussing the energy aspects under such settings. In addition, the deliverable gives the state-of-the-art machine learning based algorithms for orchestration in the IoT-edge-cloud platform including how these can be tailored towards realizing an energy-efficient edge-cloud platform. The deliverable also discusses the existing solutions for exploiting AI

models and algorithms that are cognizant of the constrained energy nature of edge nodes.

- A7-D4.1 - Planificación del laboratorio y de los pilotos de las plataformas 5G IoT-edge-cloud: This deliverable presents the work plan to set up a real-time digital twin laboratory and a 5G IoT-edge-cloud platform at iTEAM-UPV, and to deploy the pilots of the project. In particular, the work plan is detailed considering the design and implementation phases, and the test plan.
- A7-D2.2 - Evolución de la arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales. This deliverable describes the architecture of the different components to enable the IoT-edge-cloud orchestration, like the closed loop orchestration framework and the observability stack that, together with the proposed AI/ML algorithms, enable an efficient orchestration of computing resources and services. A closed loop orchestration pilot demo that integrates and validates all the orchestration components is also presented in the deliverable.
- A7-D2.3 - Informe final de la arquitectura de las plataformas IoT-edge-cloud y soluciones de AI/ML para gemelos digitales: This deliverable presents the final design of AI/ML algorithms for energy efficiency and the orchestration actions that can be driven by the developed algorithms. In addition, the deliverable describes the datasets with focus on energy efficiency that can be extracted with the observability stack
- A7-D4.2 - Puesta en marcha de las plataformas 5G IoT-edge-cloud en el laboratorio y en los pilotos de gemelos digitales: The document details the implementation of the 5G IoT-edge-cloud platforms in the iTEAM-UPV laboratory and in the pilots, integrating, testing and validating the different solutions and components available in the project, documenting the P4.1 prototype and the VT4.1 validation pilot. Specifically, the deliverable presents a closed loop orchestration pilot demo that integrates and validates all the orchestration components. In addition, the deliverable presents the NearbyOne orchestration solution architecture, and a description of how the currently available ADV5G-TWINS edge infrastructure is provisioned and how to onboard 3rd party applications and services into the orchestration platform to be deployed and orchestrated into the infrastructure.
- A7-D4.3 - Pruebas finales del laboratorio y pilotos de gemelos digitales en tiempo real con plataformas 5G IoT-Edge-Cloud. The deliverable documents the setup and results of the real-time digital twin pilots with 5G IoT-edge-cloud platforms in relation to activity A8 of the project.

In addition to the above technical deliverables, the following deliverables for activities A8 and A7 related to software/demo/prototype were accomplished.

- A8-S2.1 - Algoritmos de orquestación de plataformas IoT-edge-cloud
- A8-S2.2 - Algoritmos basados en AI/ML para la orquestación de plataformas IoT-edge-cloud
- A8-P4.1 - Creación del laboratorio y piloto de gemelos digitales para las plataformas 5G IoT-edge-cloud
- A8-P4.2 - Actualización de las plataformas 5G IoT-edge-cloud para el laboratorio y los pilotos
- A8-VT4.1 - Primera fase de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud
- A8-VT4.2 - Fase final de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud
- A7-S2.1 - Algoritmos de orquestación de plataformas IoT-edge-cloud
- A7-S2.2 - Algoritmos basados en AI/ML para la orquestación de plataformas IoT-edge-cloud
- A7-P4.1 - Creación del laboratorio y piloto de gemelos digitales para las plataformas 5G IoT-edge-cloud
- A7-P4.2 - Actualización de las plataformas 5G IoT-edge-cloud para el laboratorio y los pilotos
- A7-VT4.1 - Primera fase de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud
- A7-VT4.2 - Fase final de pilotos de gemelos digitales con las plataformas 5G IoT-edge-cloud

3. Technical Contributions

The project realized the following technical contributions.

3.1. Closed-loop orchestration framework

The ADV5G-TWINS-ORCHESTRATION project developed an automated service management system that optimizes operations based on user-defined goals instead of manual configurations. This framework is leveraged for both activities A7 and A8.

With this framework, administrators specify desired outcomes, and the system dynamically deploys, configures, and adjusts services in a closed-loop process. This loop continuously monitors application performance and refines orchestration decisions accordingly. The project's closed-loop framework first presented in deliverable D2.2 is illustrated in Figure 1.

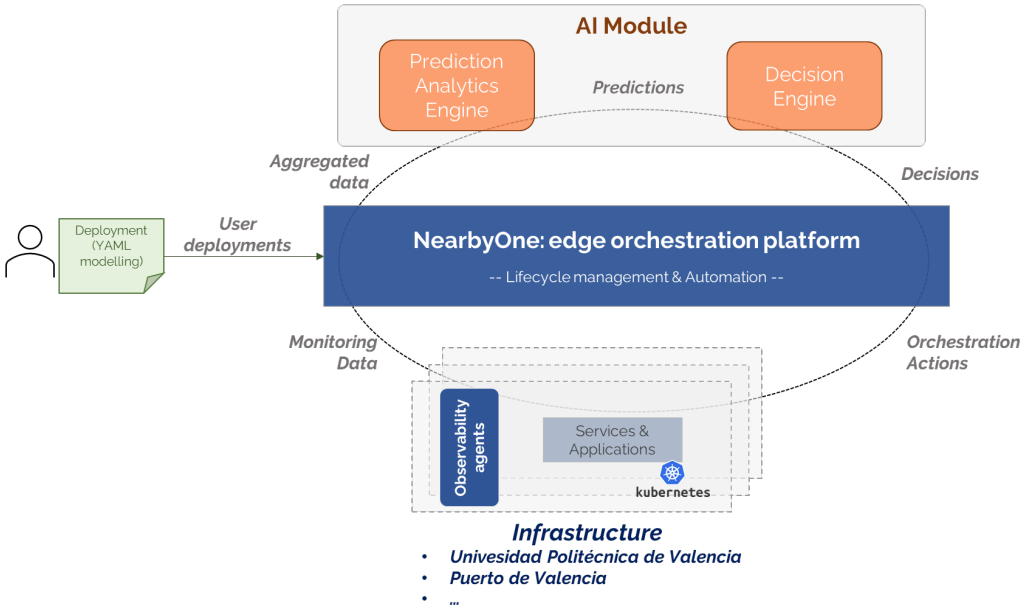


Figure 1: Closed-loop Orchestration framework

The orchestration framework targets to achieve zero-touch service orchestration through feedback mechanisms. The developed architecture is composed of a number of components including the following:

3.1.1. Infrastructure

This serves as a host for the different applications and observability agents that are used for monitoring metrics related to the infrastructure and deployed applications such as resource utilization, energy consumption and system health, among others. In particular, the infrastructure for ADV5G-TWINS consists of two sites located in the UPV and the Port of Valencia. Each site corresponds to a dedicated edge server, Lenovo SE450 that has been provisioned with a Kubernetes system to manage containerized applications.

3.1.2. AI Module

The orchestrated applications, services and network functions, and the underlying infrastructure where they are deployed, generate a continuous flow of data that the AI Module can consume and analyze. This analytical process yields valuable feedback that the orchestrator can use to detect and correct anomalies or to reconfigure existing deployments. To do so, the AI Module enables advanced automation based on the integration of AI/ML techniques.

The AI Module is composed of the *Prediction Analytics Engine* and the *Decision Engine (DE)*, thus combining artificial intelligence and data analysis functionalities. The *Prediction Analytics Engine* leverages a machine learning model for forecasting system state based on the historical data aggregated in the observability stack. These ML models managed via mlflow are used to analyze historical and real-time metrics to predict future system requirements, enabling proactive resource allocation.

The *Decision Engine* takes as input the output from the *Prediction Analytics Engine* and current network state to generate orchestration decisions, that target to ensure compliance with the user specified objectives. The logic within the DE determines actions such as scaling, migration, or service termination to align with the user's intent (e.g., maintaining QoS without overprovisioning resources). These orchestration decisions are then passed to the orchestration platform through the northbound interface.

3.1.3. The E2E Orchestration platform

The E2E orchestration engine is based on NearbyOne, introduced in ADV5G-TWINS-ORCHESTRATION deliverable *D4.1* and further described in D4.2. The orchestration platform of the closed-loop system is responsible for full lifecycle management of distributed network Infrastructure and applications encompassing all layers, from the edge to the cloud, seamlessly managing all the areas of operations that take part in edge deployments: the infrastructure, and the applications. The orchestration platform performs all tasks related to the orchestration of applications in terms of lifecycle management including service instantiation, migration, scaling and termination, among others as needed, considering the requirements defined in the user deployments and the orchestration decisions triggered by the decision engine.

3.1.4. Enabling interfaces

ADV5G-TWINS-ORCHESTRATION closed-loop leverages a number of interfaces for communication between the different components of the framework. These include the northbound interface for communication between the orchestration platform and third-party logic such as the decision engine, the Southbound Interface (SBI) for communication between the orchestrator and the underlying infrastructure, the interface for accessing aggregated data and the interface for user deployment, among others. These interfaces were detailed in deliverable D2.2: "Evolución de la arquitectura de las plataformas IoT-Edge-Cloud y soluciones de AI/ML para gemelos digitales"

3.2. AI/ML algorithms

To support AI driven pro-active orchestration and energy efficiency, the project developed AI algorithms. Specially, we introduce **OmniFORE** (Framework for Optimization of Resource Forecasts in Edge-Cloud Networks) which leverages the attention mechanism-based informer model [1] and **AERO** (Adaptive Edge-cloud Resource Orchestration) [2], a lightweight forecasting model with under 1,000 parameters, designed for dynamic workload prediction in resource-constrained edge environments. These algorithms are discussed below.

3.2.1. Attention based Framework for Optimization of Resource Forecasting in Edge-Cloud Networks

To enable proactive and resource-efficient orchestration in cloud-native environments, deliverable D2.2: "Evolución de la arquitectura de las plataformas IoT-Edge-Cloud y soluciones de AI/ML para gemelos digitales" and activity A7 of deliverable D2.3 introduced a workload prediction model based on attention mechanisms, designed to improve both prediction accuracy and inference time for workloads with long-term dependencies.

However, beyond these attributes, the dynamic nature and spatiotemporal variability of workload traces associated with resource consumption across distributed cloud-native nodes demand models with strong generalization capabilities, thus, able to maintain high performance during sudden traffic shifts or previously unseen patterns.

The ADV5G-TWINS-ORCHESTRATION project developed OmniFORE (Framework for Optimization of Resource Forecasts in Edge-Cloud Networks), which combines attention-based time-series models with temporal clustering to improve generalization and workload prediction in dynamic edge-cloud networks. By leveraging informers from activity A8 of the D2.3, which use sparse attention to capture long-term dependencies, OmniFORE outperforms traditional RNNs and CNNs. We introduced a temporal clustering scheme for strategic training on representative data subsets, balancing short-term stability with long-term variations. In addition, an efficient data sampling strategy which reduces computational overhead while ensuring robust predictions is introduced. Real-world experiments confirm OmniFORE's superior accuracy and faster inference compared to state-of-the-art methods.

In the sections below, we introduce the system model followed for the performance analysis and design of the algorithms including the obtained results.

3.2.1.1. System Model

In this section, we introduce the system model used for the design and evaluation of OmniFORE against SoA algorithms. The system model focusses on container resource traces' representation, heterogeneity, and workload forecasting using historical data.

Traces

We consider an edge-cloud network with containers operating at all infrastructure layers. Each container generates resource traces, represented as $T(t) \in \mathbb{R}^{d_t \times d_f}$, where d_t denotes the time dimension and d_f represents the number of features. These features include metrics such as CPU usage,

memory consumption, network bandwidth, and storage I/O operations. The matrix representation of the historical time-series data for a single trace is

$$T_i(t) = \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,d_t} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,d_t} \\ \vdots & \vdots & \ddots & \vdots \\ d_{d_f,1} & d_{d_f,2} & \cdots & d_{d_f,d_t} \end{bmatrix} \in \mathbb{R}^{d_f \times d_t}. \quad (1)$$

Let N_{all} denote the total number of container traces in the network. Hence, a set of container traces can be defined as

$$\mathbf{T}(t) = \{T_1, T_2, \cdots, T_{N_{\text{all}}}\} \in \mathbb{R}^{d_f \times d_t \times N_{\text{all}}}.$$

Workload Forecasting

The primary objective of the workload forecasting task is to accurately forecast future resource usage, denoted as $\hat{T}$, for a given prediction length d_y, using historical data.

The forecasting model f_θ, parameterized by θ, is designed to make these predictions. Specifically, the model uses past observations of the resource usage trace T_i to predict the future usage, given by

$$\hat{T}_i[t+1:t+d_y] = f_\theta(T_i[t-d_x+1:t]), \quad (2)$$

where d_x represents the sequence length, indicating the number of past observations used for predictions, while d_y represents the prediction length. The model parameters, denoted by θ, are the variables that need to be optimized.

Our goal is to minimize the Mean Absolute Error (MAE) for each container trace in the network, i.e.,

$$\min \frac{1}{d_y} \sum_{i=1}^{N_{\text{all}}} \| T_i[t+1:t+d_y] - \hat{T}_i[t+1:t+d_y] \|_1, \quad (3)$$

where $\hat{T}_i[t+1:t+d_y]$ is the prediction vector and $T_i[t+1:t+d_y]$ is the actual resource usage vector at time $t+d_y$ for trace i.

3.2.1.2. OmniFORE: framework for optimization of resource forecasts in edge-cloud networks

OmniFORE brings together several important innovations: a strategic method for sampling data, advanced techniques for clustering time-series data, an in-depth introduction to various attention mechanisms, hyperparameter optimization, and scaling strategies to handle large numbers of traces. This section discusses these components of the OmniFORE framework.

Trace Sampling Strategy

To achieve robust generalization across diverse workloads, OmniFORE employs a novel approach to extract a representative set of historical traces

from the edge-cloud network. This strategy ensures that the model trained on these traces can effectively generalize to various workload types. The detailed steps (S_1 - S_5) of this process are outlined in Figure 2.

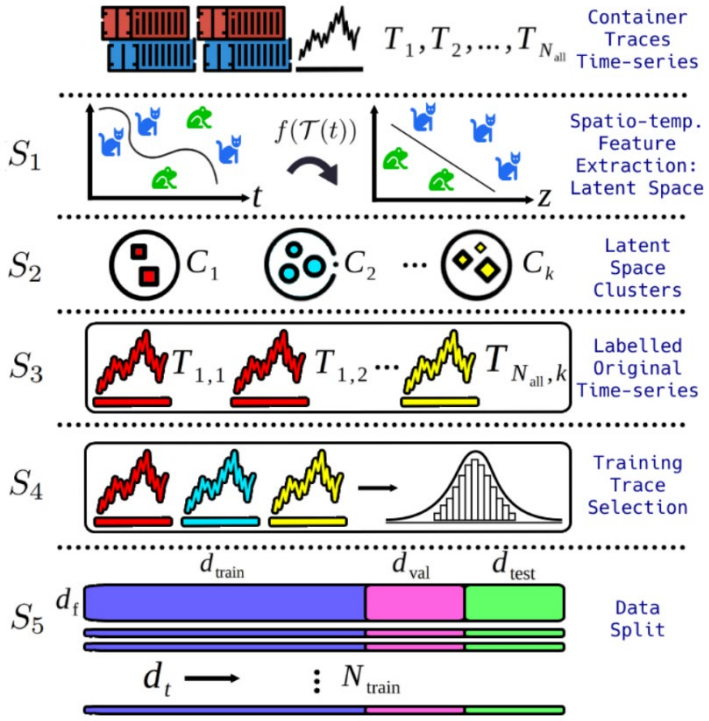


Figure 2. Trace sampling process for generalization training. Historical workload data are processed to extract container traces, which are clustered into representative groups.

The goal of these steps is to extract a set of historical traces, denoted as $\mathbf{T}_{\text{train}}(t)$, where N_{train} represents the number of training traces. This set is a representative subset of the entire set of traces, $\mathbf{T}_{\text{all}}(t)$, which contains N_{all} traces. "Representative" means that $\mathbf{T}_{\text{train}}(t)$ captures the essential characteristics and variations present in $\mathbf{T}_{\text{all}}(t)$, ensuring that the training set accurately reflects the diversity and distribution of the full dataset:

$$\mathbf{T}_{\text{train}}(t) \in \mathbb{R}^{d_f \times d_t \times N_{\text{train}}} \subset \mathbf{T}_{\text{all}}(t) \in \mathbb{R}^{d_f \times d_t \times N_{\text{all}}}, \qquad (4)$$

where $N_{\text{train}} < N_{\text{all}}$. Using all traces for training would be computationally prohibitive and costly.

The highly dynamic nature of container data, marked by their time-dependent spatio-temporal patterns, presents significant challenges for analysis. Traditional clustering algorithms often struggle with such complex time-series data[3]. To overcome this, we propose transforming the data into a latent space to create a dense representation, facilitating more efficient clustering and differentiation [4]. This latent space captures essential features, enabling better management of the data's complexities.

First (S_1), OmniFORE extracts spatio-temporal features from the container trace time-space, denoted as $\mathcal{T}$. Each trace $T_i(t) \in \mathbb{R}^{d_f \times d_t}$ is then transformed

into a lower-dimensional latent space, $\mathcal{Z} \in \mathbb{R}^z$, where $z \ll d_f \times d_t$. This transformation, defined as $\mathcal{Z}(z) = f_{\text{encoder}}(\mathcal{T}(t))$, is optimized to capture the essential spatio-temporal patterns of the data over time. By converting the time-dependent space $\mathcal{T}$ into the latent space $\mathcal{Z}$, OmniFORE optimizes for the most informative features for clustering, enhancing both efficiency and effectiveness by reducing dimensionality while preserving essential temporal information.

Second (S_2), the latent space $\mathcal{Z}$ is then clustered. Each cluster C_k groups all container traces with similar spatio-temporal patterns over time into K groups. The clustering process can be represented as

$$\{C_1, C_2, \dots, C_K\} = \text{Cluster}(\mathcal{Z}(z)). \quad (5)$$

Third (S_3), each cluster C_k labels the original time-series traces, preserving the unique characteristics of each trace for accurate model training while leveraging the efficiency of latent space clustering. This can be mathematically depicted as

$$\mathbf{T}_{\text{all, cluster}}(t) = \{T_{1,1}(t), T_{2,1}(t), \dots, T_{N_{\text{all}},k}(t)\}, \quad (6)$$

where $T_{1,1}(t)$ denotes the first trace in cluster 1.

Fourth (S_4), representative traces from each cluster C_k are selected for training the prediction model. The selection process is optimized to ensure that the training data is diverse and representative of the broader dataset. Given a total number of traces N_{all} and a desired subset of N_{train} traces, the selection process is proportional to the number of traces in each cluster.

Let N_{C_k} denote the number of traces in cluster C_k. The proportion of traces in each cluster compared to the total number of traces is

$$p_k = \frac{N_{C_k}}{N_{\text{all}}} \in [0,1]. \quad (7)$$

The number of traces selected from each cluster C_k for the training set is

$$N_{\text{train},k} = \lfloor p_k \cdot N_{\text{train}} \rfloor, \quad (8)$$

where $N_{\text{train},k}$ is the number of traces selected from cluster C_k for training.

Therefore, the total number of traces in the training set T_{train} is the sum of the selected traces from each cluster, such that

$$N_{\text{train}} = \sum_{k=1}^{K} N_{\text{train},k}. \quad (9)$$

Finally (S_5), the traces in the sampled training set are split into train/validation/test segments along the time dimension ($d_t = d_{\text{train}} + d_{\text{val}} + d_{\text{test}}$).

OmniFORE ensures that data is effectively utilized for training, validating, and testing the prediction models, resulting in robust and generalizable performance.

Unsupervised Temporal Clustering

Figure 3 provides an overview of the latent space extractor. The input time-series $\mathcal{T}(t)$ is transformed into a latent space $\mathcal{Z}(z)$ by the encoder function f_{encoder}. The decoder function f_{decoder} then reconstructs $\mathcal{Z}(z)$ into the output time-series $\mathcal{R}(t)$, which approximates $\mathcal{T}(t)$. This neural network structure compresses the data into a latent space, captures its critical features, and then reconstructs the original data. Successful reconstruction with the latent space acting as a bottleneck indicates that the latent space effectively encapsulates the essential information of the input signal.

To utilize this dense representation, we modify a supervised (using correct cluster labels for each trace) CNN-based approach for time-series clustering from [5] to an unsupervised (without cluster labels) version within OmniFORE. This process involves feature extraction through convolutional and pooling operations on raw data, followed by clustering in the latent space.

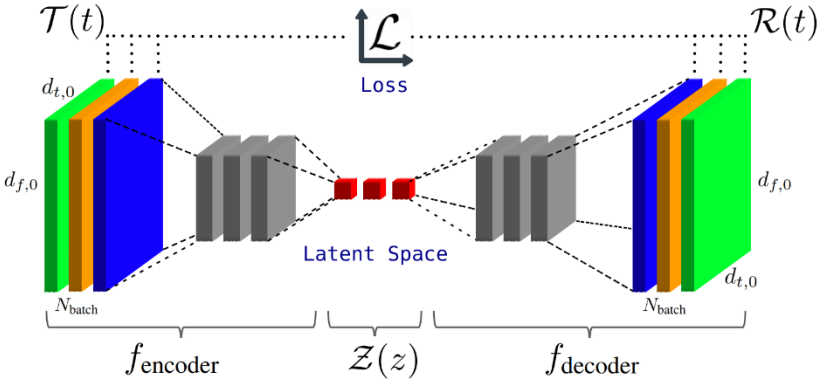


Figure 3. Overview of the latent space extractor. The input T(t) is encoded into latent space Z(z) by fencoder and then decoded by fdecoder to reconstruct R(t)≈T(t).

The architecture of the latent space extractor comprises the following layers:

1. **Encoder**: Transforms an input trace T_i into the latent space $\mathcal{Z}(z)$ using $\mathcal{Z}(z) = f_{\text{encoder}}(\mathcal{T}(t))$.
   - Input Layer: $d_{f,0} \times d_{t,0}$ neurons, where $d_{t,0}$ is the series length and $d_{f,0}$ is the number of features.
   - Several repetitions r of combined convolutional and pooling layers, where $d_{f,r}$ is the number of filters and $d_{t,r}$ is the size of the sequence dimension after the r-th convolutional layer. The transformed sequence is $T'(t') \in \mathbb{R}^{d_{f,r} \times d_{t,r} \times N_{\text{all}}}$.
     - Convolutional Layers $\mathrm{C}_r(t')$: Parameters are convolution stride (s_r), filter size ($d_{f,r-1} \times l_r$), activation functions, trainable weights **($\boldsymbol{\omega}_r$)**, and biases ($\mathbf{b}_r$).

- Pooling Layers $\mathrm{P}_r(t')$: Downsampling strategy to reduce feature map size by averaging regions.

- Latent Space Layer: Converts the flattened output of the last convolutional layers, $\mathbf{C}_r$, into a lower-dimensional latent space, $\mathcal{Z} \in \mathbb{R}^{z \times N_{\text{batch}}}$, where $z \ll d_{f,0} \times d_{t,0}$.

2. **Decoder**: Converts $\mathcal{Z}(z)$ back to $\mathcal{T}(t)$ as $\mathcal{R}(t) = f_{\text{decoder}}\big(\mathcal{Z}(z)\big) \approx f_{\text{encoder}}^{-1}\big(\mathcal{Z}(z)\big)$.
   - Multiple Transposed Convolution Layers: Reconstruct the time-series input data from the latent representation, back to the original data shape. Transposed convolutions, also known as deconvolutions, upsample the input by reversing the operations of standard convolutional layers.

The training process can be described as:

1) **Initialize**: Set up the Encoder-Decoder architecture, initialize weights and biases.
2) **Load Input Data**: Prepare input traces as $\mathbf{T}_{\text{all}} \in \mathbb{R}^{d_{t,0} \times d_{f,0} \times N_{\text{all}}}$, split into N_{batch} batches. Here, N_{batch} denotes the number of batches.
3) **Forward Pass**: Compute the output of each layer for each batch.
4) **Loss Function**: Calculate the mean-square error across all batches between the reconstructed time-series and the actual input time-series as

$$\mathcal{L} = \frac{1}{2N_{\text{batch}}} \sum_{n=1}^{N_{\text{batch}}} \sum_{t=1}^{d_t} (\boldsymbol{T_i}(t) - \boldsymbol{R_i}(t))^2. \quad (10)$$

5) **Update**: Use gradient descent to update the weights and biases of the (transposed) convolutions (ω_r, b_r) and optimize for minimizing the loss function.

6) **Iterate**: Repeat with new batches until the model converges.

This unsupervised approach compresses time-series data into a latent space, capturing essential features and enabling reconstruction. By adapting CNN-based clustering to work without labels, it achieves efficient feature extraction for selecting appropriate model training traces.

Scaling

Container traces in edge-cloud computing exhibit significant variability in resource usage, necessitating individual normalization to ensure balanced training. OmniFORE normalizes each trace such that the mean of the metric under study is zero (i.e., $\mu_i = 0$) and the variance is one (i.e., $\sigma_i^2 = 1$), thereby preventing traces with higher means and variances from overshadowing others and enhancing model generalization.

Given a training set $\mathbf{T}_{\text{train}} \in \mathbb{R}^{d_t \times d_f \times N_{\text{train}}}$, each trace $T_i(t)$ is normalized as follows:

$$\tilde{T}_i(t) = \frac{T_i(t)-\mu_i}{\sigma_i}, \quad (11)$$

where $\tilde{T}_i(t)$ denotes the normalized version of the trace $T_i(t)$, μ_i is the mean, and $\sigma_i = \sqrt{\sigma_i^2}$ is the standard deviation of the i-th trace. The mean μ_i and variance σ_i^2 are calculated as

$$\mu_i = \frac{1}{d_t}\sum_{t=1}^{d_t} T_i(t), \sigma_i^2 = \frac{1}{d_t}\sum_{t=1}^{d_t}(T_i(t)-\mu_i)^2. \quad (12)$$

The scaling parameters (μ_i, σ_i^2) are saved for each trace. During testing, these parameters are used to invert the normalization of predictions back to the original scale, such that

$$\hat{T}_i(t) = \tilde{T}_i(t)\cdot\sigma_i + \mu_i, \quad (13)$$

where i indexes each trace to maintain the scaling parameters for each trace.

Bayesian Optimization with Generalization Objective

Optimizing hyperparameters is essential for improving model performance and generalization[6]. Due to the impracticality of traditional grid search in expansive hyperparameter spaces, Bayesian optimization [4] offers an efficient alternative by constructing a probabilistic model of the objective function and iteratively selecting the most promising hyperparameters.

Figure 4 illustrates Bayesian optimization, starting with initial observations (red dots) of the objective function. This simplified example reduces the hyperparameter space to prediction length d_y, though in practice it is multidimensional (hyperparameter vector $\mathbf{h}$). A model (blue line) predicts the objective function's mean and confidence interval (CI, shaded area). The next hyperparameter (star) is selected based on predicted high performance and uncertainty, balancing exploration and exploitation.

OmniFORE's custom objective function Obj($\mathbf{h}$) aims to maximize generalization across diverse container traces:

$$\text{Obj}(\mathbf{h}) = \frac{1}{N_{\text{val}}}\sum_{i=1}^{N_{\text{val}}} \text{Perf}\left(\mathbf{h}, T_{\text{val},i}\right), \quad (14)$$

where N_{val} is the number of validation traces, $T_{\text{val},i}$ is the i-th validation trace, and Perf$\left(\mathbf{h}, T_{\text{val},i}\right)$ represents the performance metric.

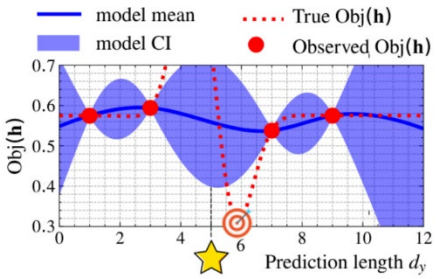


Figure 4. Bayesian optimization toy example. The star indicates the next value to try (d_y=5). The target signifies the goal of minimizing the objective function.

3.2.1.3. Performance evaluation

In this section, we present our evaluation results, demonstrating that the proposed model outperforms existing algorithms across all criteria. OmniFORE consistently shows lower error rates, better generalization on new data, and faster inference times.

Generalization Performance

Regular Training

Figure 5 displays the prediction performance of various models on test data from Google Cells A-F after regular training. OmniFORE demonstrates superior performance over AGCRN and LSTNet across all evaluation metrics.

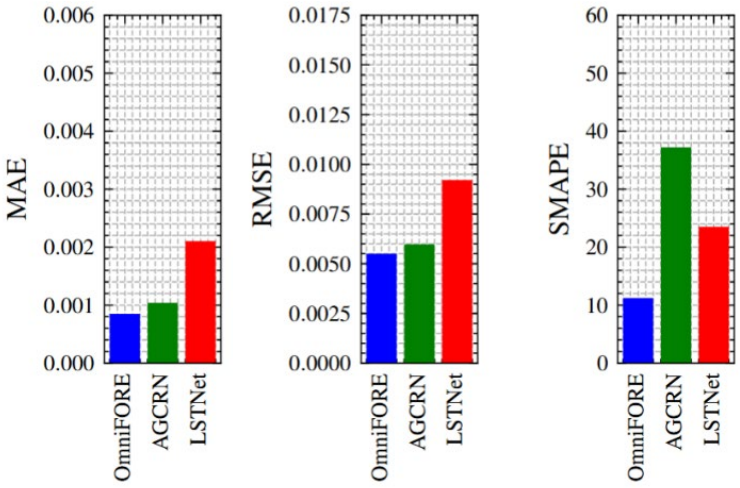


Figure 5: Generalization performance comparison of MAE, RMSE and SMAPE for each model for the regular training traces

OmniFORE achieves an MAE of $\mathbf{8.53 \times 10^{-4}}$, RMSE of $\mathbf{5.50 \times 10^{-3}}$, and SMAPE of 11.21%. In contrast, AGCRN records an MAE of $\mathbf{1.04 \times 10^{-3}}$, RMSE of $\mathbf{5.98 \times 10^{-3}}$, and SMAPE of 37.19%. LSTNet reports an MAE of $\mathbf{2.10 \times 10^{-3}}$, RMSE of $\mathbf{9.21 \times 10^{-3}}$, and SMAPE of 23.52%. OmniFORE's MAE is 18.21% lower than AGCRN and 59.42% lower than LSTNet. Its RMSE is 7.97% lower than AGCRN and 40.21% lower than LSTNet. OmniFORE's SMAPE is 69.86% lower than

AGCRN and 52.34% lower than LSTNet. This notable improvement is attributed to OmniFORE's advanced attention mechanism, which captures both long-term and short-term correlations among workloads.

Furthermore, OmniFORE achieves these results with an average inference time of 8.70 ms, compared to 131 ms for AGCRN and 10.6 ms for LSTNet. This represents a 17.92% reduction in inference time compared to the best-performing SoA scheme, highlighting OmniFORE's efficiency for real-time applications where quick orchestration decisions are required with minimal resource overhead. This efficiency is thanks to OmniFORE's use of generative inference, which predicts the entire sequence in a single step rather than sequentially.

Zero-Shot Performance

Figure 6 illustrates the prediction performance using zero-shot test data from Google Cells G and H. OmniFORE achieves an MAE of 6.34×10^{-4}, RMSE of 2.92×10^{-3}, and SMAPE of 6.31%. In comparison, AGCRN records an MAE of 3.93×10^{-3}, RMSE of 9.65×10^{-3}, and SMAPE of 55.27%, while LSTNet shows an MAE of 5.02×10^{-3}, RMSE of 1.49×10^{-2}, and SMAPE of 39.91%.

OmniFORE's superior zero-shot performance, surpassing even its regular training performance, stems from its advanced attention-based mechanism that captures complex patterns and dependencies, enabling robust generalization. In contrast, AGCRN and LSTNet perform worse on zero-shot data, indicating potential benefits from information leakage between training and test sets. Additionally, data variability and the exact nature of loaded traces can influence performance outcomes.

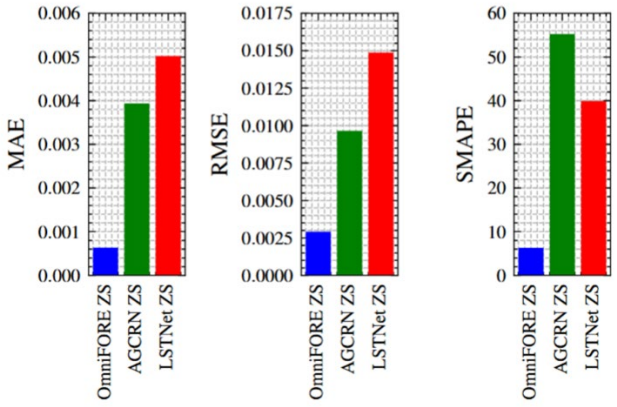


Figure 6: Generalization performance comparison of MAE, RMSE, and SMAPE for each model on the zero-shot traces

3.2.2. Adaptive Edge-cloud Resource Orchestration

In section 3.2.1 , we captured the AI algorithm based on attention mechanism that was developed in the project targeting to capture long-term

dependencies among workloads. However, the adopted algorithm achieves these with slightly increased complexity in terms of model parameters. Envisaging an edge environment with resource constrained nodes in terms of memory, the project developed an additional AI algorithm **AERO** (Adaptive Edge-cloud Resource Orchestration), a lightweight forecasting model with under 1,000 parameters, designed for dynamic workload prediction in resource-constrained edge environments. This algorithm in described in detail in [2].

In the subsequent section, we introduce the system model, design and evaluation of the proposed algorithm.

3.2.2.1. System Model

In this section, we formalize our cloud workload prediction framework through four key components: workload representation, workload prediction model, objective function, and edge-cloud network architecture. This formalization provides the mathematical foundation for comparing different architectural approaches.

Workload Representation

Let $\mathbf{W}_t = \left[W_t^{(1)}, W_t^{(2)}, \ldots, W_t^{(f)}\right]^\top \in \mathbb{R}^f$ represent the workload vector at discrete time step t, where $W_t^{(i)}$ denotes the i-th resource utilization feature (e.g., CPU, memory usage), f is the number of features, and $t \in \{1,2,\ldots,T\}$, with T being the total number of time steps in the trace. The complete workload trace forms a multivariate time series:

$$\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_T]^\top \in \mathbb{R}^{T\times f}. \qquad (16)$$

Workload Prediction Model

Our objective is to forecast future workload vectors based on historical observations. We define a prediction function $\Psi_{\boldsymbol{\phi}}: \mathbb{R}^{\kappa\times f} \to \mathbb{R}^{\lambda\times f}$, parameterized by $\boldsymbol{\phi} \in \mathbb{R}^p$, which represents the trainable parameters of the prediction model (e.g., weights and biases of a neural network). The function $\Psi_{\boldsymbol{\phi}}$ estimates the workload over a prediction horizon of length λ, where $\lambda \in \mathbb{N}$ and $\lambda \geq 1$:

$$\widehat{\mathbf{W}}_{t+1:t+\lambda} = \Psi_{\boldsymbol{\phi}}(\mathbf{W}_{t-\kappa+1:t}), \qquad (17)$$

where $\kappa \in \mathbb{N}, \kappa \geq 1$, is the length of the historical context window, and $\mathbf{W}_{t-\kappa+1:t} \in \mathbb{R}^{\kappa\times f}$ denotes the sequence of past workload vectors used for prediction. The model architecture and training process are determined by hyperparameters $\boldsymbol{\theta} \in \Theta$, where Θ defines the search space of valid configuration[6].

Objective Function

To learn the model parameters $\boldsymbol{\phi}$, we minimize a loss function that measures the prediction error between the forecasted workload $\widehat{\mathbf{W}}_{t+1:t+\lambda}$ and the ground

truth workload $\mathbf{W}_{t+1:t+\lambda}$ across all resource dimensions. We employ the Mean Squared Error (MSE) as our optimization objective, which penalizes larger prediction errors quadratically:

$$\mathcal{L}(\boldsymbol{\phi}) = \frac{1}{\lambda}\sum_{i=1}^{\lambda}\|\mathbf{W}_{t+i} - \widehat{\mathbf{W}}_{t+i}\|_2^2, \qquad (18)$$

where $\|\cdot\|_2$ denotes the Euclidean norm in $\mathbb{R}^f$ and $t \in \{1,2,\ldots,T-\lambda\}$. This loss function computes the average squared distance between predicted and actual resource utilization vectors across the prediction horizon λ.

Edge-cloud Network Architecture

Our simulations utilize a heterogeneous edge-cloud network comprising edge devices, fog nodes, and cloud servers. Let $\mathcal{H} = \{h_1, h_2, \ldots, h_N\}$ be the set of computational nodes. Each node $h_i \in \mathcal{H}$ possesses specific resource capacities defined as a vector $\mathbf{c}_i = \left[c_i^{\text{cpu}}, c_i^{\text{memory}}, c_i^{\text{disk}}, c_i^{\text{network}}\right]^{\mathsf{T}}$, where c_i^{cpu} represents CPU capacity in MHz, c_i^{memory} represents memory capacity in KB, c_i^{disk} represents storage capacity with read/write speeds in KB/s, and c_i^{network} represents network bandwidth in Gb/s.

Let $\mathcal{C}$ denote the set of containers, where each container is decomposed into a set of tasks $\mathcal{J}_c$ for container $c \in \mathcal{C}$. Each task $j \in \mathcal{J}_c$ has resource requirements $\mathbf{r}_j = \left[r_j^{\text{cpu}}, r_j^{\text{memory}}, r_j^{\text{disk}}, r_j^{\text{network}}\right]^{\mathsf{T}}$ and arrival time a_j, with measurements collected at regular intervals.

Container-level workload predictions inform task scheduling across nodes to optimize resource utilization and meet Quality of Service (QoS) requirements. The scheduler employs these predictions to make proactive allocation decisions aiming to minimize metrics such as energy consumption and Service Level Agreement (SLA) violations [15].

3.2.2.2. AERO Architecture

Edge-cloud environments pose unique challenges for resource orchestration due to their heterogeneity, resource constraints, and dynamic workload patterns. **AERO** (**A**daptive **E**dge-cloud **R**esource **O**rchestration), a lightweight forecasting model specifically tailored for adaptive workload prediction in edge-cloud settings. AERO is designed to operate efficiently on resource-constrained edge devices while maintaining the high prediction accuracy essential for effective scheduling decisions.

Model Overview

The AERO model operates on the historical workload data $\mathbf{W}_{t-\kappa+1:t} \in \mathbb{R}^{\kappa \times f}$, where κ is the context window length and f is the number of features, as defined in Section 5.1. The model serves as the prediction function $\Psi_{\boldsymbol{\phi}}$ parameterized by $\boldsymbol{\phi} \in \mathbb{R}^p$, mapping historical workloads to future predictions over a horizon λ:

$$\widehat{\mathbf{W}}_{t+1:t+\lambda} = \Psi_{\boldsymbol{\phi}}(\mathbf{W}_{t-\kappa+1:t}). \quad (19)$$

The AERO architecture consists of three main components:

A. Adaptive Period Detection Module: Identifies and adapts to the dominant period τ within the historical workload data by analyzing temporal patterns.
B. Adaptive Forecasting Module: Adjusts the detected period and performs forecasting using the SparseTSF model. This module handles the core prediction task, leveraging the strengths of SparseTSF for efficient time series forecasting.

C. Adaptive Layer Caching: Maintains a cache of neural network layers optimized for different periodicities. When similar workload patterns recur, the model reuses these cached layers instead of recomputing them, significantly reducing computational overhead.

In the following sections, we delve into the detailed workings of each component of the AERO model.

Adaptive Period Detection Module

Detecting the dominant period τ is critical, as it directly determines the model's configuration for optimal performance. The parameter τ dictates how the model segments and processes the time series data, influencing its ability to accurately capture and leverage periodic patterns. By precisely identifying τ, we ensure that the model aligns with the inherent periodicity of the data, effectively separating periodic components from trends and increasing forecasting accuracy.

The Adaptive Period Detection Module aims to identify the dominant period τ within the historical workload data sequence $\mathbf{W}_{t-\kappa+1:t}$. First, we compute the average across the feature dimension to obtain a univariate time series $\bar{\mathbf{W}}_{t-\kappa+1:t} \in \mathbb{R}^{\kappa}$:

$$\bar{\mathbf{W}}_{t-\kappa+1:t} = \frac{1}{f}\sum_{i=1}^{f} \mathbf{W}_{t-\kappa+1:t}^{(i)}, \quad (20)$$

where $\mathbf{W}_{t-\kappa+1:t}^{(i)}$ denotes the i-th feature of the workload data sequence $\mathbf{W}_{t-\kappa+1:t}$.

Next, we subtract the mean to focus on the periodic components, yielding the mean-centered series $\widetilde{\mathbf{W}}_{t-\kappa+1:t}$:

$$\widetilde{\mathbf{W}}_{t-\kappa+1:t} = \bar{\mathbf{W}}_{t-\kappa+1:t} - \boldsymbol{\mu}_{t-\kappa+1:t}, \quad (21)$$

where $\boldsymbol{\mu}_{t-\kappa+1:t} = \frac{1}{\kappa}\sum_{s=t-\kappa+1}^{t} \bar{\mathbf{W}}_s$ is the mean of the averaged workload sequence.

To efficiently identify the dominant period τ, we begin by computing the autocorrelation function (ACF) of the mean-centered workload sequence $\widetilde{\mathbf{W}}_{t-\kappa+1:t}$. The autocorrelation function measures the similarity between the

time series and its lagged versions, which is essential for detecting repeating patterns and periodicities in the data.

The autocorrelation function is computed using convolution[16] :

$$\text{ACF}(\ell) = \frac{(\tilde{\mathbf{W}}_{t-\kappa+1:t} * \tilde{\mathbf{W}}^{\text{rev}}_{t-\kappa+1:t})[\ell]}{\text{ACF}(0)+\epsilon}, \quad \epsilon = 10^{-8}, \qquad (22)$$

where $\ell \in \{0,1,2,\dots,\kappa-1\}$ is the lag or time shift, $\tilde{\mathbf{W}}^{\text{rev}}_{t-\kappa+1:t}$ is the time-reversed version of $\tilde{\mathbf{W}}_{t-\kappa+1:t}$, $*$ denotes the convolution operation, and $\text{ACF}(0)$ is the variance of $\tilde{\mathbf{W}}_{t-\kappa+1:t}$.

After obtaining the ACF, we identify peaks, which are points where the autocorrelation is higher than its immediate neighbors:

$$\text{peaks} = \{\ell \mid \text{ACF}(\ell-1) < \text{ACF}(\ell) > \text{ACF}(\ell+1)\}. \qquad (23)$$

These peaks correspond to potential periods in the data.

Candidate periods $\mathcal{K}$ are selected from the peaks within the desired range $[\tau_{\min}, \tau_{\max}]$ (model hyperparameters):

$$\mathcal{K} = \{\ell \in \text{peaks} \mid \tau_{\min} \le \ell \le \tau_{\max}\}. \qquad (24)$$

Finally, the dominant period τ is assigned based on the candidate periods:

$$\tau = \begin{cases} \left\lfloor \frac{\tau_{\max}}{2} \right\rfloor, & \text{if } \mathcal{K} = \emptyset, \\ \underset{\ell \in \mathcal{K}}{\text{argmax}} \text{ACF}(\ell), & \text{otherwise.} \end{cases} \qquad (25)$$

When no candidate periods are found, we default to $\tau = \left\lfloor \frac{\tau_{\max}}{2} \right\rfloor$. This choice ensures at least two complete segments within the maximum period range (a hyperparameter), providing the minimum basis for temporal comparison in the absence of periodic patterns. A larger τ would yield insufficient segments for comparison, while a smaller τ risks over-segmentation. This balanced approach maintains the model's ability to capture potential temporal relationships even without explicit periodicity.

Adaptive Forecasting Module

After detecting the dominant period τ from the historical data, the Adaptive Forecasting Module adjusts this period to ensure compatibility with the model architecture and proceeds to predict future workloads $\hat{\mathbf{W}}_{t+1:t+\lambda} \in \mathbb{R}^{\lambda \times f}$, where λ is the prediction horizon.

We first clamp the detected period τ to a valid range and make it even to satisfy the model's requirements:

$$\tau' = \text{clamp}\left(\tau,\ 2,\ \left\lfloor \frac{\kappa}{2} \right\rfloor\right), \quad \tau' = \tau' + (\tau' \bmod 2). \qquad (26)$$

Clamping τ ensures that the period is within a feasible range based on the context window length κ, and making it even simplifies subsequent segmentation and reshaping operations.

Next, we adjust the clamped period τ' to the closest common divisor of κ and λ:

$$\tau^* = \underset{d \in D}{\operatorname{argmin}} |d - \tau'|, \qquad (27)$$

where D is the set of common divisors of κ and λ. Adjusting τ' to τ^* ensures that both the input sequence and the prediction horizon can be evenly divided into segments of length τ^*, facilitating efficient downsampling and upsampling in the model.

After this point, the forecasting process follows the standard SparseTSF model[14] . For completeness, we outline their forecasting process below.

First, the input data is normalized by subtracting the mean vector:

$$\widetilde{\mathbf{W}}_{t-\kappa+1:t} = \mathbf{W}_{t-\kappa+1:t} - \mathbf{1}\boldsymbol{\mu}_{t-\kappa+1:t}^{\top} \in \mathbb{R}^{\kappa \times f}, \qquad (28)$$

where $\mathbf{1} \in \mathbb{R}^{\kappa}$ is a column vector of ones, and $\boldsymbol{\mu}_{t-\kappa+1:t} = \frac{1}{\kappa}\sum_{s=t-\kappa+1}^{t} \mathbf{W}_s \in \mathbb{R}^{1 \times f}$.

The normalized data is then transposed to align dimensions for convolution:

$$\widetilde{\mathbf{W}} = \widetilde{\mathbf{W}}^{\top} \in \mathbb{R}^{f \times \kappa}. \qquad (14)$$

A one-dimensional convolution is applied with residual connection:

$$\mathbf{Z} = \text{Conv1D}\left(\widetilde{\mathbf{W}}; k, p\right) + \widetilde{\mathbf{W}} \in \mathbb{R}^{f \times \kappa}, \qquad (29)$$

where the kernel size $k = 1 + 2(\tau^*/2)$ and padding $p = \tau^*/2$.

The data is then downsampled through reshaping:

$$\mathbf{Z} = \text{reshape}(\mathbf{Z}, [f \cdot \tau^*, n_x]) \in \mathbb{R}^{(f \cdot \tau^*) \times n_x}, \qquad (30)$$

where $n_x = \kappa/\tau^*$ represents the number of segments in the input sequence.

A linear transformation maps the downsampled segments to prediction segments:

$$\mathbf{Y} = \text{Linear}(\mathbf{Z}) \in \mathbb{R}^{(f \cdot \tau^*) \times n_y}, \qquad (31)$$

where $n_y = \lambda/\tau^*$ represents the number of segments in the prediction sequence.

The predictions are then upsampled through reshaping:

$$\mathbf{Y} = \text{reshape}(\mathbf{Y}, [f, \lambda]) \in \mathbb{R}^{f \times \lambda}. \qquad (32)$$

Finally, the predictions are denormalized:

$$\widehat{\mathbf{W}}_{t+1:t+\lambda} = \mathbf{Y}^{\top} + \mathbf{1}\boldsymbol{\mu}^{\top} \in \mathbb{R}^{\lambda \times f}. \qquad (33)$$

This sequence of operations efficiently processes the time series data while maintaining its periodic structure. In our implementation, we leverage this architecture with our adaptively detected period τ^*.

Adaptive Layer Caching Mechanism

In dynamic edge-cloud environments, workloads exhibit varying periodicities. To efficiently adapt without significant overhead, we introduce an Adaptive Layer Caching Mechanism in the AERO model. For each feasible period τ within a practical range $[\tau_{\min}, \tau_{\max}]$, we cache pre-initialized neural network layers configured specifically for that τ. These layers include convolutional layers with kernel size $k = 1 + 2\left(\frac{\tau}{2}\right)$ and padding $\frac{\tau}{2}$, and linear layers mapping input segments of size $n_x = \kappa/\tau$ to output segments of size $n_y = \lambda/\tau$.

When the dominant period τ is detected, it may not exactly match a cached period. In such cases, we adjust τ to the nearest cached value. This adjustment is acceptable because small variations in period length often correspond to similar temporal patterns in the workload data.

3.2.2.3. Evaluation

In this section, we present the evaluation of our proposed AERO model against state-of-the-art approaches using metrics including prediction accuracy, computational efficiency, and model complexity.

Dataset

For performance evaluation against the SoA algorithms, we use the Alibaba 2022 microservices trace dataset [16], focusing on pod MS_11349. The dataset provides $T = 18,720$ samples collected at 1-minute intervals over 13 days. Each workload vector $\mathbf{W}_t \in \mathbb{R}^f$ consists of $f = 2$ features: normalized CPU utilization ($W_t^{(1)}$) and memory utilization ($W_t^{(2)}$), aligning with the workload representation in Section 4.2.1. This forms a workload trace $\mathbf{W} \in \mathbb{R}^{T \times f}$ exhibiting both deterministic trends $\boldsymbol{\mu}_t$ (e.g., daily patterns) and stochastic fluctuations $\boldsymbol{\epsilon}_t$ (e.g., demand spikes), making it ideal for evaluating prediction models with different historical context windows κ and forecast horizons λ.

Benchmark algorithms

We compare AERO with six state-of-the-art models, each representing a distinct architecture:

- **FourierGNN** [18] combines Graph Neural Networks (GNNs) with Fourier transforms to capture complex spatio-temporal dependencies.
- **ModernTCN** [8]employs Temporal Convolutional Networks (TCNs) with advanced dilation to model long-range dependencies efficiently.

- **WGAN [10]** is a transformer-based model utilizing a generative adversarial approach, representing the state-of-the-art in cloud-native workload prediction.
- **Pathformer[11]**, designed by Alibaba Group, optimizes the Transformer architecture with efficient attention mechanisms.
- **DLinear** [19] is a lightweight model that decomposes time series into trend and seasonal components using moving averages, processes each with separate linear layers, and combines them for prediction.
- **SparseTSF** [14] is a lightweight framework based on sparse MLPs, designed for fast inference and minimal resource consumption.

Experimental Settings

All experiments were conducted on a server with NVIDIA A100 GPU (40 GB), 30 CPU cores, 200 GiB RAM, running Ubuntu 20.04.2 LTS. Models were implemented in PyTorch, and MLFlow tracked all experiments. To ensure fair comparisons, we standardized the experimental setup. Our hyperparameter tuning method used Bayesian optimization (BoTorch)[20], GPyTorch[21] with ranges from original authors. Each model had 20 random hyperparameter evaluations, followed by 30 optimization runs (totaling 50 training runs per model), optimizing for validation loss. All parameters were logged to MLFlow[22]. Per individual training run, models trained up to 100 epochs with early stopping (patience 10), optimizing validation loss. We used identical learning rate schedulers. Data were identically partitioned, and input was uniformly normalized and rescaled. Preprocessing steps ensured identical inputs. We used the same random seed for reproducibility. By aligning hyperparameter tuning, training protocols, computational resources, data handling, and evaluation metrics, we minimized biases, ensuring fair comparisons and strengthening our conclusions.

We assess model performance using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), the two most commonly used metrics in time-series forecasting:

$$\text{MAE} = \frac{1}{\lambda}\sum_{i=1}^{\lambda}\|\mathbf{W}_{t+i} - \widehat{\mathbf{W}}_{t+i}\|_1, \quad (34)$$

$$\text{RMSE} = \sqrt{\frac{1}{\lambda}\sum_{i=1}^{\lambda}\|\mathbf{W}_{t+i} - \widehat{\mathbf{W}}_{t+i}\|_2^2}, \quad (35)$$

where $\|\cdot\|_1$ and $\|\cdot\|_2$ denote the ℓ_1 and Euclidean norms, respectively. In addition to prediction accuracy, we evaluate practical deployment considerations including inference time, convergence speed, parameter count, and memory usage.

Experimental Results

Prediction Accuracy

Accurate workload prediction is essential for efficient resource allocation and maintaining Quality of Service in cloud-native scenarios. All models demonstrate predictive capabilities, but key differences emerge (see Figure 7).

For memory utilization, Pathformer achieves the best predictive performance with an RMSE of 2.23×10^{-4}, while ModernTCN, AERO, and SparseTSF form a competitive cluster with RMSE values of 3.58×10^{-4}, 3.78×10^{-4}, and 3.99×10^{-4}, respectively. DLinear achieves moderate memory prediction performance with an RMSE of 7.38×10^{-4}, demonstrating the trade-offs of its linear decomposition approach. This clustering suggests these architectures effectively capture the underlying patterns in memory usage, though FourierGNN and WGAN lag significantly behind with higher RMSE values, indicating potential limitations in their ability to model memory consumption patterns.

In CPU utilization prediction, we observe similar performance across all models. The RMSE values cluster tightly around 2.8×10^{-3}, indicating that CPU patterns may be more challenging to predict precisely due to their stochastic nature.

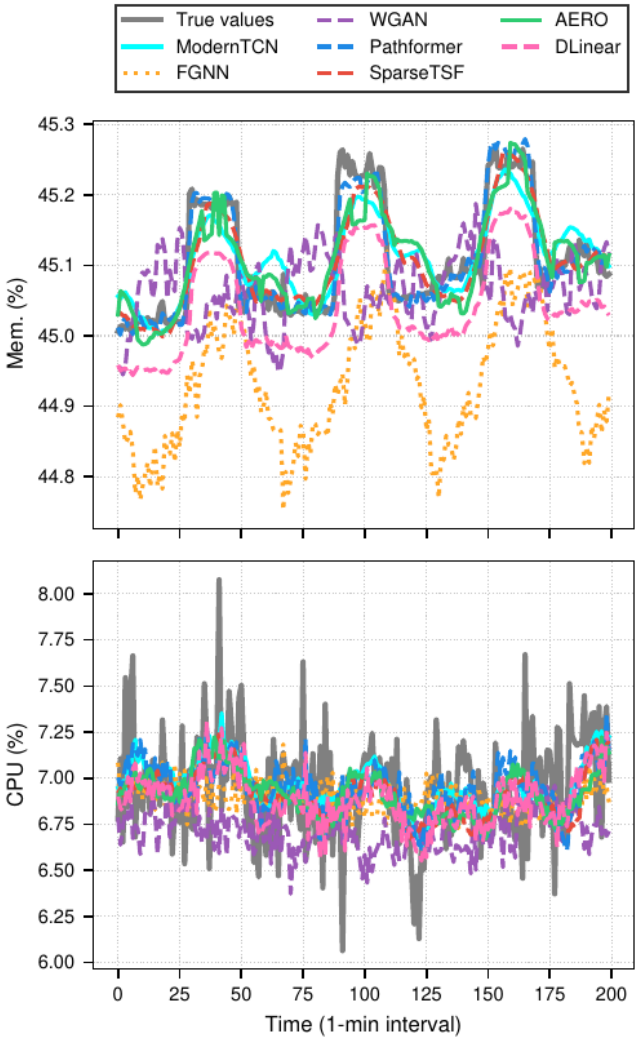


Figure 7: predicted vs true values for memory (top) and CPU (bottom)

Model Complexity and Size

Figure 8 presents a logarithmic comparison of model complexity, highlighting the lightweight nature of AERO relative to more complex architectures. Resource efficiency is critical in cloud-native deployments, particularly at the edge. SparseTSF stands out with only 35 parameters and minimal memory usage (140 bytes), ideal for resource-constrained devices. AERO has 599 parameters and a memory footprint of 2,236 bytes, balancing lightweight design with enhanced functionality. DLinear occupies a middle ground with 4128 parameters, offering a practical balance between model complexity and predictive power. While larger than SparseTSF, AERO maintains a sub-1K parameter count while offering adaptability to varying periodicities.

ModernTCN and FourierGNN offer a balance between complexity and performance, suitable for environments with moderate resources. Pathformer and WGAN, with millions of parameters and higher memory requirements, are more appropriate for powerful edge nodes or cloud servers.

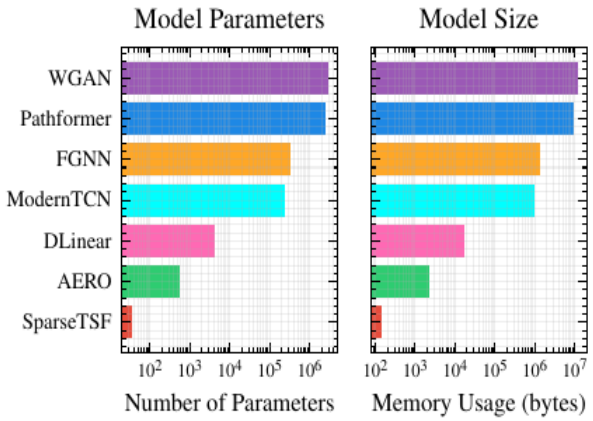


Figure 8: Logarithmic comparison of model complexity: number of parameters and maximum memory usage

3.3. Software components, frameworks and demos

3.3.1. NearbyOne orchestration solution

Deliverable *D4.1: Planificación del laboratorio y de los pilotos de las plataformas 5G IoT-Edge-Cloud"* (Activity 8) of the ADV5G-TWINS-ORCHESTRATION project documented the proposed architecture for orchestration of the digital twin laboratory. This is also shown in Figure 9 for reference.

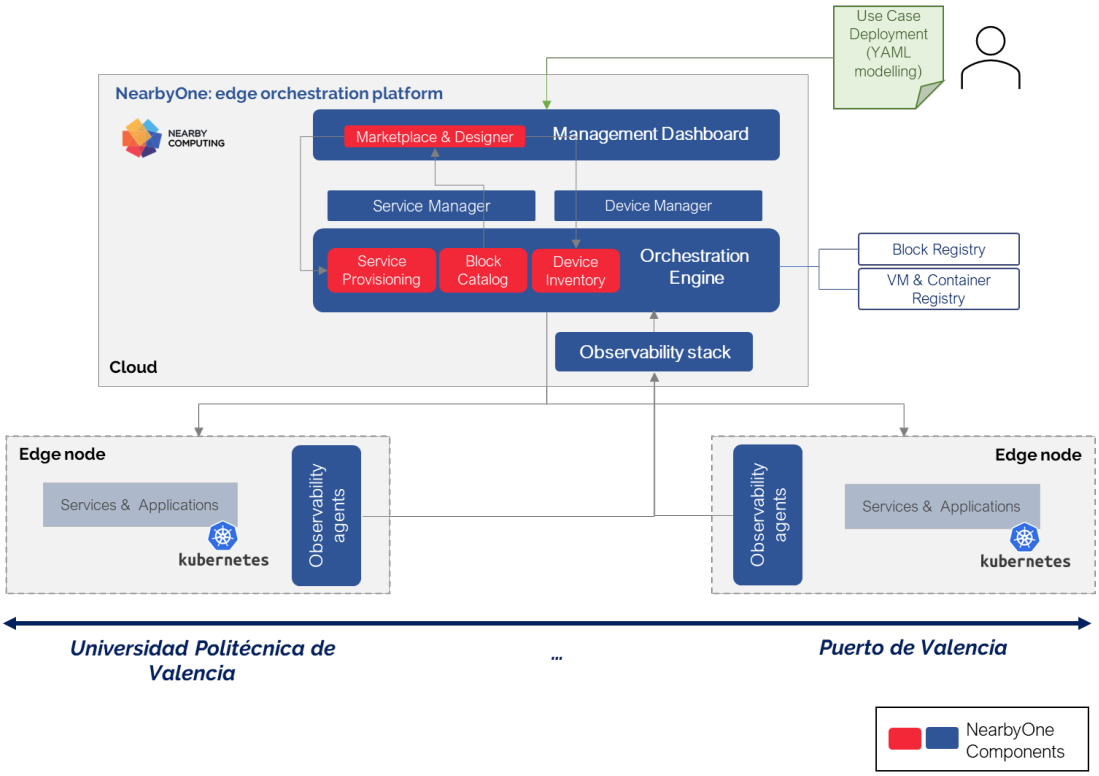


Figure 9. ADV5G-TWINS orchestration architecture.

The NearbyOne orchestrator is responsible for the service onboarding and life-cycle management (LCM) of cloud-native applications and infrastructure at a global scale, across the edge cloud compute continuum. NearbyOne Orchestrator plays the role of the multi-cluster orchestration engine in ADV5G-TWINS. NearbyOne solution and provides mechanisms to automate and orchestrate the edge infrastructure located in Valencia, and the deployment of components, such as the observability stack, or the use case applications. In particular, NearbyOne will manage the two sites located in the UPV and the Port of Valencia. Each site corresponds to a dedicated edge server, Lenovo SE450 that has been provisioned with a Kubernetes system to manage containerized applications. The NearbyOne controller, a cloud native service by itself with more than ~30 different microservices and pods, has been deployed in a public cloud, in Amazon Web Services (AWS).

The NearbyOne orchestrator management dashboard is accessible in the following URL:

https://nearbyone.innovationlab.nearbycomputing.com/

The access to the NearbyOne UI requires authentication via user and password. The users shown in Figure 10 have been granted access to the orchestrator platform. More users can be created upon demand to the Nearby Computing team.

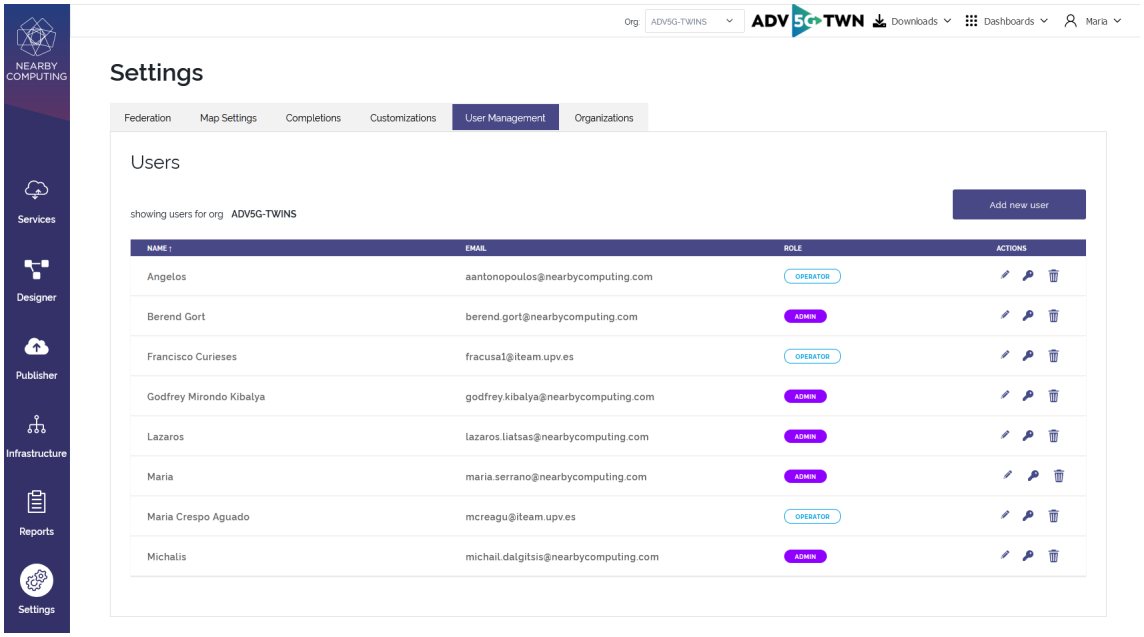


Figure 10. ADV5G-TWINS NearbyOne management dashboard users.

3.3.1.1. NearbyOne architecture

NearbyOne Orchestrator offers different flavors of Edge Computing on a Single Pane of Glass, including lifecycle management of the infrastructure, and advance service placement and lifecycle management. NearbyOne addresses all orchestration issues in a holistic and intertwined way, not as an aggregation of separate problems. It targets multi-cloud and multi-cluster environments, allowing for hybrid and unified management of VM resources and container resources from a single pane of glass. NearbyOne offers a full 360 view of the edge, addressing the main challenges of application, network functions, and infrastructure management through a single pane of glass and an intuitive GUI (NearbyOne ***Management Dashboard***). NearbyOne not only provides mechanisms to automate and orchestrate provisioning of services but also offers a ***Marketplace*** that facilitates the choice and chained deployment of services and applications with only one click. The *Marketplace*, easily extendible, gives access to the catalog of solutions available for rapid service deployment through the NearbyOne Dashboard (see Figure 11). The ***Designer*** module of the NearbyOne Dashboard is used to define service deployment chains and establish the relations among them. Deployments, in the form of YAML manifests, contain not only the application deployment information, but also its requirements and placement/configuration policies.

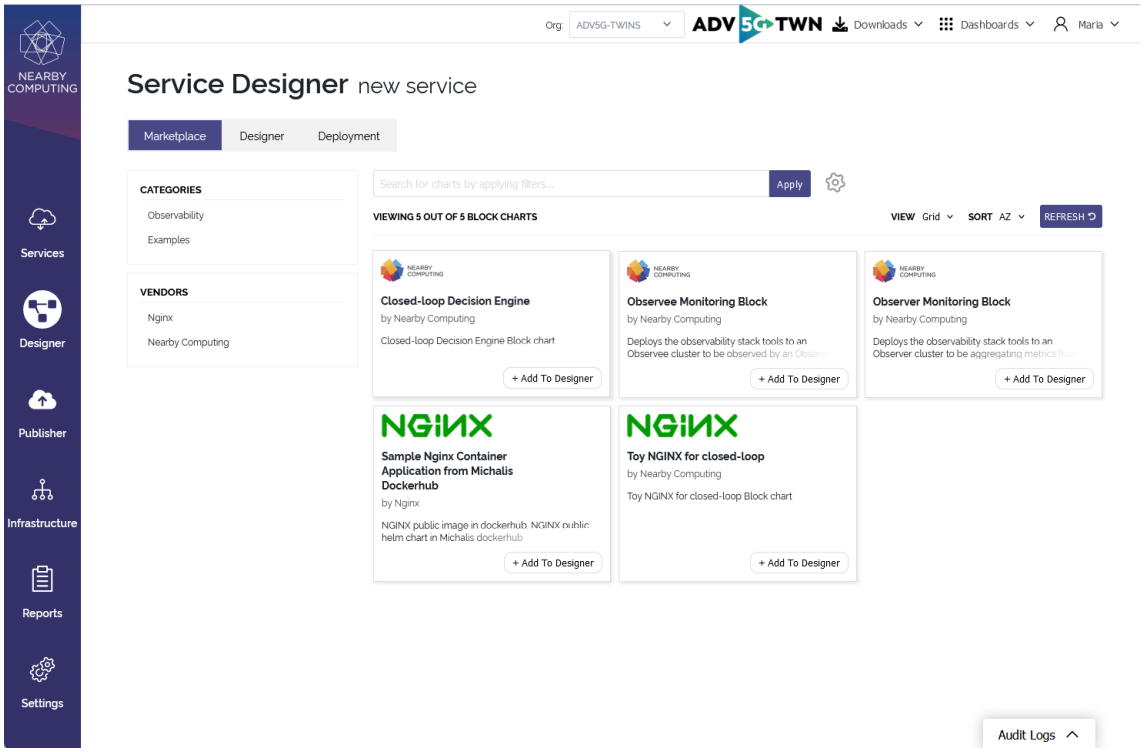


Figure 11. ADV5G-TWINS NearbyOne dashboard Marketplace.

At the core of the orchestrator, the ***Device manager*** is responsible for the operations related to edge sites, while the ***Service manager*** is responsible for managing high-level CRUD (create, read, update, and delete) operations for service chain deployments and other components related to them. The ***Orchestration engine*** is the main operational component responsible for managing complex and distributed operations, enforcing eventual consistency rather than transactions. It follows the pattern of self-healing reconciliation loops. It also includes components that manage the policies and other dynamic behavior related to services and devices. The ***Block Catalog*** is responsible for keeping track of the service definitions and their associated images and binary file dependencies. The ***Service Provisioning*** manages service access control as well as service level definitions and policies associated to each service instance. The ***Device Inventory*** is responsible for maintaining the catalog of orchestrated devices tracking information such as MAC addresses, IP addresses, hardware capabilities, or provisioned flavor for each device, while also managing their provisioning and providing resource access control enforcement.

The use case applications and services are the central component of the use case, and they require the seamless integration of components provided by third-party software developers or vendors. Application developers are not required to make any changes to their application code for proper integration with the orchestration platform. All configurations are made using declarative languages (YAML).

3.3.1.2. Infrastructure provisioning

Infrastructure provisioning refers to the process of setting up and managing the necessary computing resources, such as servers, storage, networking equipment, and software, to support the ADV5G-TWINS project. This section describes the process to get the remote edge infrastructure in the UPV, and the port of Valencia provisioned and ready to deploy services.

Modern edge infrastructure is built on several layers, as shown in Figure 12, starting from the physical or bare metal layer to the applications or business logic. The bare metal layer provides the raw computing power and connectivity; the Operating System (OS) manages the hardware resources and provides a stable environment for container runtime; the containerization layer deploys and manages containerized applications; and, finally, the applications, in the form of cloud-native workloads, deliver the end-user functionalities and services, designed to be scalable and resilient.

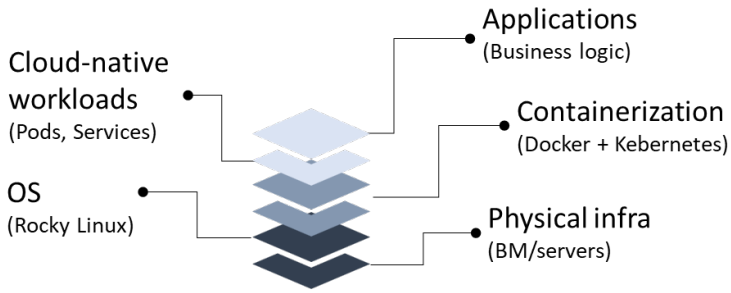


Figure 12. Modern Edge infrastructure layers.

This multi-layered approach allows for efficient resource utilization, rapid deployment of new services, and robust management of complex workloads, making it ideal for edge computing scenarios.

UPV Edge server

After gaining access to the physical UPV edge server, where the hardware and OS Rocky Linux are configured, the container provisioning process starts. This process leads all the way up to set up a Kubernetes cluster to be onboarded in NearbyOne as part of the orchestrated infrastructure. The details of the UPV edge server are the following:

- Hardware: Lenovo SE450
- *IP address: 172.22.0.3*
- *ssh access credentials:*
    - *Sudo user: nearby-computing*
    - *Password:* ****
- OS: Rocky Linux 9.4 (Blue Onyx)

The container provisioning is performed using **Infrastructure as Code** (IaC), a modern approach that uses code instead of manual processes and settings. This approach uses declarative configuration files to define the desired state of the infrastructure and applications, enabling consistent and repeatable deployments.

In particular, we use Ansible[1], an open-source automation tool that uses a simple, human-readable language (YAML) to describe automation jobs, making it easy to write and understand. Ansible's scripts, called *playbooks*, describe each step, ensuring that every time they're run, they produce the same environment, reducing errors and inconsistencies.

A summary of the actions taken to provision the UPV edge server are:

- OS utilities, dependencies and other configurations: installation of some dependencies, tools and OS utilities, and other OS settings configurations.
- Ansible playbook: different playbooks are applied to install dependencies, the *Rancher Kubernetes Engine* (*RKE*) Kubernetes distribution, and Longhorn[2] that provides persistent storage for Kubernetes.
- NearbyOne Agents: Install and configure a service for facilitating communication between the NearbyOne orchestrator and the Kubernetes cluster, where the orchestrated application will run.

Finally, the new Kubernetes cluster is onboarded in NearbyOne , as part of the orchestrated infrastructure, to deploy and manage the lifecycle of applications and services.

Preliminary dependencies and OS configurations

The preliminary manual installations include the following dependencies and tools:

- `pip 24.0`, the package installer for Python.
- `PyYAML 6.0.1`, a YAML parser and emitter for Python.
- `kubernetes 29.0.0`, Python client for Kubernetes.
- `git 2.43.5`, a distributed version control system.
- `Docker Engine 27.0.1`, an open-source containerization technology for building and containerizing applications.
- `iscsi-initiator-utils 6.2.1.9`, iSCSI (Internet Small Computer System Interface), a protocol for distributed storage.

[1] https://www.ansible.com/
[2] https://longhorn.io/

- `nfs-utils 1:2.5.4-25.el9`, NFS (Network File System) , a distributed file system protocol that allows you to share files and directories between systems.

Other manual configurations:

- Stop the `firewalld` service
- Create an `SSH key`

Ansible playbooks

The following Ansible playbooks have been executed:

- `Kernel` playbook: Disable swap memory and install OS dependencies (`epel-release`, `iptables`, `pciutils`).
- `Cluster bootstrap` playbook: Install RKE K8s cluster (`rke_helm` version 3.13.0; `rke` version 1.4.5, and `rke_kubectl` version 1.28.0) with its specific configuration.
- `Longhorn` playbook: Longhorn installation into the RKE cluster, that will act as the default Storage Class in Kubernetes (persistent storage).

This automated process via Ansible playbooks results in a successful installation of a **single-node RKE Kubernetes cluster** in the UPV edge server, see Figure 13.

```
[nearby-computing@localhost ~]$ kubectl cluster-info
Kubernetes control plane is running at https://172.22.0.3:6443
CoreDNS is running at https://172.22.0.3:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
[nearby-computing@localhost ~]$
```

Figure 13. UPV edge server Kubernets cluster info.

NearbyOne agent

Skupper[3] is an open-source tool that enables secure multicloud communication across Kubernetes clusters with no VPNs or special firewall rules. In particular, we use Skupper as a NearbyOne *agen*t to enable communication between the edge server on the infrastructure and the NearbyOne orchestrator. Skupper solves multi-cluster communication challenges through something called a Virtual Application Network (VAN). A VAN connects the applications and services in your hybrid cloud into a virtual network so that they can communicate with each other as if they were all running in the same site.

[3] https://skupper.io/

We use a helm chart to install Skupper on the k8s cluster of the UPV edge server, and expose its Kubernetes API service to the NearbyOne controller, see Figure 14.

```
[nearby-computing@localhost ~]$ kubectl get deployments -n nearbyone-agents
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
skupper-router               1/1     1            1           44d
skupper-service-controller   1/1     1            1           44d
skupper-site-controller      1/1     1            1           44d
```

Figure 14. NearbyOne Skupper agent in UPV Kubernetes cluster.

Onboarding site to NearbyOne

The NearbyOne orchestrator extensively supports the integration of Kubernetes-based edge sites as part of the infrastructure to deploy applications and services (container-based deployments). NearbyOne is capable of registering the existing clusters and monitoring the infrastructure. To do so, the NearbyOne dashboard offers a user-friendly approach by simply using the Kubernetes `.kubeconfig` file that contains the necessary information for the orchestrator to communicate with the API server of the target Kubernetes cluster.

The NearbyOne dashboard has an Infrastructure section, where users can create Sites. A Site is a logical space where clusters are added. Sites can have sub-sites, building a hierarchy. After creating a Site, users are able to register an existing Kubernetes cluster, as shown in the steps of Figure 15. This involves 1) selecting Add device in the Infrastructure view; 2) selecting the Site; 3) selecting the type of device (existing Kubernetes Cloudlet in this case); and 4) providing the device details (name, optional tags, `.kubeconfig` file, and position coordinates). With this, the UPV edge cluster is integrated into NearbyOne, ready for monitoring and deploying edge services.

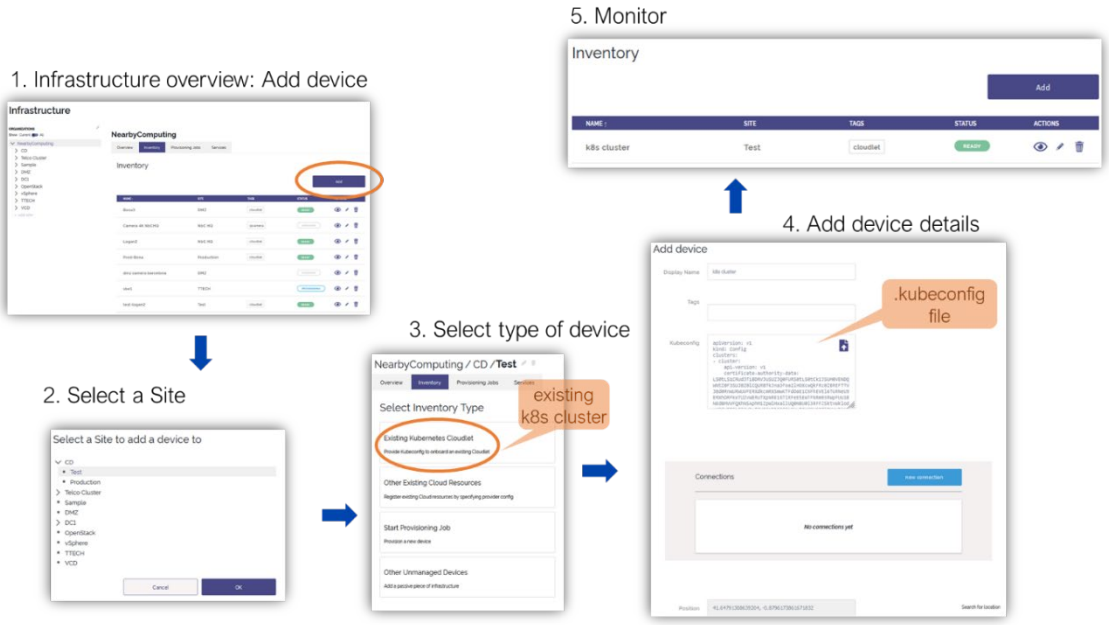


Figure 15. NearbyOne dashboard: interface to onboard an existing Kubernetes cluster.

3.3.1.3. Integration with 3rd party services

Once the NearbyOne orchestrator and the infrastructure to deploy services are ready, NearbyOne allows users to onboard 3rd party applications and services into the platform to be deployed and orchestrated into the infrastructure (see the marketplace in Section 3.3.1). This is done through the definition of a number of declarative files that define the behavior that the orchestration platform must follow to fulfil user-defined requirements.

Nearby blocks

In NearbyOne, third-party software components onboarded into the platform to operate services deployment are named Nearby Blocks or simply Blocks. Each Nearby Block contains references to the application logic (service containers or VMs), and they are encapsulated with a set of auxiliary components that provide the means for the application to be effectively managed. Nearby Blocks describe how to deploy applications and network functions, including different aspects such as:

- how to render their configurations
- where to place them across the registered clusters
- how many instances to deploy (statically or dynamically decided at runtime)
- how to monitor their KPIs
- how to use KPIs to manage the aspects defined above in this list

NearbyOne orchestration resources follow a similar syntax to Kubernetes manifests, and Nearby Blocks are packaged and follow a templating syntax similar to Helm charts[4].

Orchestration resources

The NearbyOne orchestration resources can be grouped based on their main functionalities:

- Container-based deployments (across managed MEC clusters):
  - ***ChartDeployment***: intent-based resource that represents a set of Helm Charts to be deployed across the container-based infrastructure. They can get associated policies for selecting target clusters to enable dynamic placement or workload co-location policies. This resource is responsible for generating *ChartRelease* resources.
    - ***ChartRelease***: one particular release of a Helm Chart to be deployed on a particular cluster available at the infrastructure level. This resource can be directly generated (being static) or can be automatically generated (being dynamic) by *ChartDeployment* resources that become responsible for the management of the *ChartRelease* resources according to policies associated to the ChartDeployment.
- Resource relations:
  - ***Connection***: a representation of information that can be shared and reused across Blocks. The information can be structured according to user needs and is represented as a text field in a resource document. The *Connection* resources can be selected by Blocks at runtime, and use internally the information as needed (i.e. to populate a configuration field).
  - ***ConnectionSet***: allows the automatic creation of *Connection* resources. This is useful to create a connection for each *ChartRelease* or Kubernetes Cluster that matches a specific criteria.

Figure 16 shows an example of a materialized helm chart deployment (nginx service), in a specific Kubernetes cluster (k8sClusterName). Notice that this is not the chart itself (as provided by the third-party vendor, `bitnami` in this case), but the actual definition of how this chart will be deployed in the MEC

[4] https://helm.sh/docs/topics/charts/

infrastructure (Kubernetes cluster) through NearbyOne. This includes the reference to the actual helm-chart as well as the definition of policies that need to be enforced associated to this deployment. The actual deployment can be relocated from one cluster to another (in a multi-cloud approach) through the ability of NearbyOne to federate the control plane of different clusters.

```yaml
apiVersion: blocks/v1beta1
kind: ChartDeployment
metadata:
  name: nginx-deployment
  namespace: 12345678-9012-3456-7890-123456789012
spec:
  template:
    metadata:
      labels:
        application: nginx-sample-application
  spec:
    k8sClusterName: e40990f8-8f55-4d02-90d0-462c48434286
    chart: nginx
    repo:
      url: https://charts.bitnami.com/bitnami
    version: 11.1.5
    values: |
      service:
        type: NodePort
        port: 80
        httpsPort: 443
        nodePorts:
          http: "31080"
          https: "31443"
```

Figure 16. NearbyOne Service deployment: ChartDeployment basic example.

Registries

Besides the Nearby blocks, other relevant external entities necessary for the operation are the registries, which are the centralized repositories used to store and access application or services essential components like orchestration artifacts and container images. NearbyOne interacts with various types of registries, each serving a unique purpose in the orchestration and deployment of applications. Figure 17 shows the NearbyOne orchestration resources (on the left) and the registries associated (on the right).

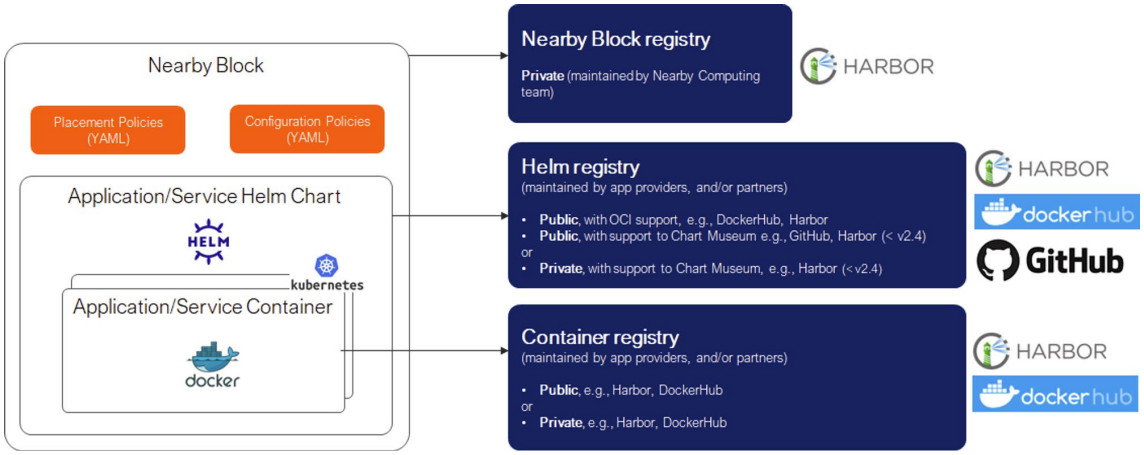


Figure 17. NearbyOne orchestration resources (on the left) and the registries associated (on the right).

- **Container Registry**: This is the repository for container Docker images. These images are essential building blocks for deploying services and applications. The registry ensures a reliable and secure distribution of images across different environments. NearbyOne pulls images from public container registries like Docker Hub[5] or private registries depending on the requirements and security considerations.
- **Helm Chart Registry**: Helm charts are used to define, install, and upgrade complex Kubernetes applications. The Helm Chart registry is where these charts are stored. The charts in this registry reference the images stored in the Container Registry. NearbyOne Helm Chart registry can be implemented under the Harbor[6] open-source project, but it can also interact with public registries that support OCI, such as Docker Hub.
- **Block Registry**: Similar to the Helm Chart Registry, the Block Registry is where the NearbyOne Blocks are stored, which are higher-level components encapsulating a service or application. Each Block references a Helm chart and, in turn, container images.

By leveraging these registries, NearbyOne ensures a smooth and efficient deployment process, enabling rapid scaling and updating of its services and applications. It is important to note that appropriate access controls and security measures are implemented to protect the integrity and confidentiality of the data in these registries.

We have set up two projects in the Nearby Computing private Harbor registry for the ADV5G-TWINS project: one Helm Chart Registry `adv5g-twins-apps`,

[5] https://hub.docker.com/
[6] https://goharbor.io/

and one Block registry `adv5g-twins-blocks`. Users in the Nearby Computing team have been granted access to the two registries to be able to push/pull Helm Charts. Moreover, a Harbor robot account has been created (`robot$adv5g-twins`) to run automated operations by using the Docker and Helm CLIs. Figure 18 shows the two ADV5G-TWINS Harbor projects created.

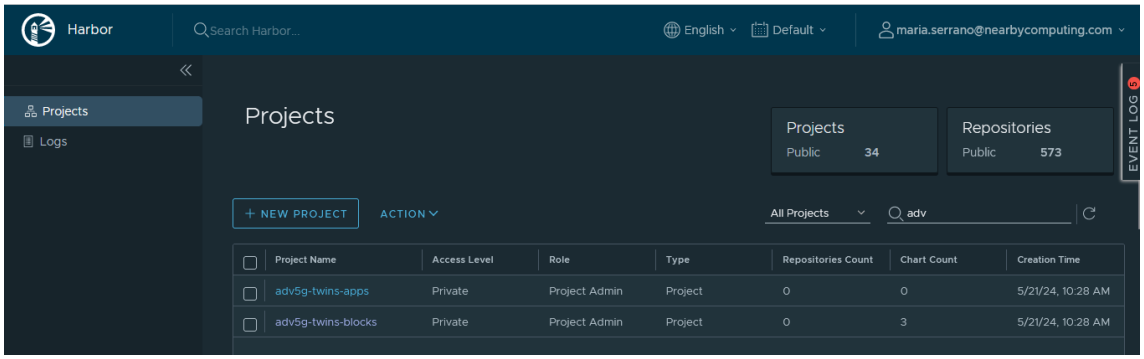


Figure 18. ADV5G-TWINS Harbor projects.

3.3.1.4. Nearby block deployment

Once the Nearby block has been implemented and pushed to the corresponding registry, the NearbyOne dashboard serves as a user-friendly interface, also offering a graphical platform for users to configure Nearby Blocks deployments effortlessly. As described in Section 3.3.1, the dashboard is accessible through a web URL, and it empowers users to interact with the orchestrator, enabling the onboarding, design, and parameterization of their intents through YAML documents. This intuitive interface enhances the user experience, providing a visual and straightforward means of orchestrating services.

As an example, Figure 19 shows the configuration values of a Nginx service, prior to deployment, in the Designer view of the NearbyOne dashboard.

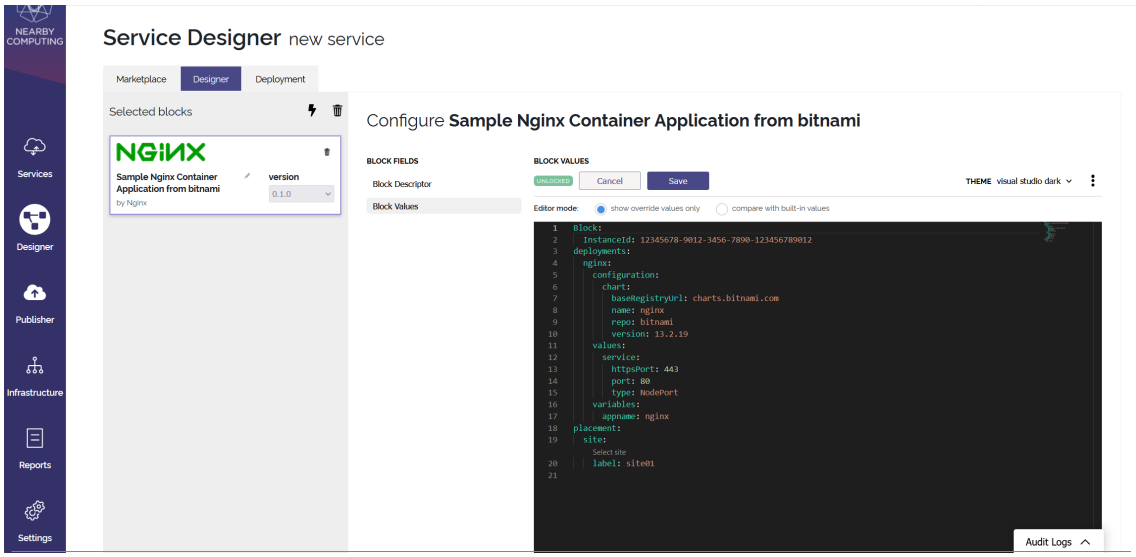


Figure 19. NearbyOne dashboard: Block deployment.

Future implementation of Nearby blocks are expected to enable the lifecycle management and orchestration of real use cases of the ADV5G-TWINS project.

4. Final demo Integration

The project also showed demonstration activities for the two activities showcasing the performance gains of AI driven orchestration in terms of energy efficiency and orchestration performance. The demonstration for the two activities are discussed below.

4.1. Activity 8: Orchestration performance

Within the ADV5G-TWINS project, one of the primary objectives is to enhance the performance of applications and services deployed over distributed infrastructure. This requires not only guaranteeing Quality of Service (QoS) but also ensuring that computational and network resources are used efficiently, given the multitude of services envisaged to be supported by future networks including 5G and 6G.

However, traditional resource management approaches often suffer from inefficiencies caused by static provisioning policies, which may lead to either over-provisioning through excessive resource allocation or under-provisioning through insufficient resource allocation. Both scenarios have detrimental effects: the former increases costs and energy consumption, while the latter compromises application performance and service reliability.

To address this challenge, ADV5G-TWINS project leverages AI-driven algorithm to achieve proactive service orchestration to ensure that the QoS of deployed applications is satisfied without resource overprovisioning. The motivation behind this solution is to enable intelligent, adaptive, and real-time decision-making regarding resource allocation. By forecasting the resource demands on a given node in advance, the orchestration framework can anticipate application needs, dynamically scale resources, and maintain the required QoS without unnecessary overuse of infrastructure resources. To achieve this with minimal human intervention, the ADV5G-TWINS Orchestration project developed a zero-touch closed-loop framework for service orchestration.

4.1.1. Demo architecture for activity A8

The Demo setup is built around the closed-loop framework architecture shown in Figure 20.

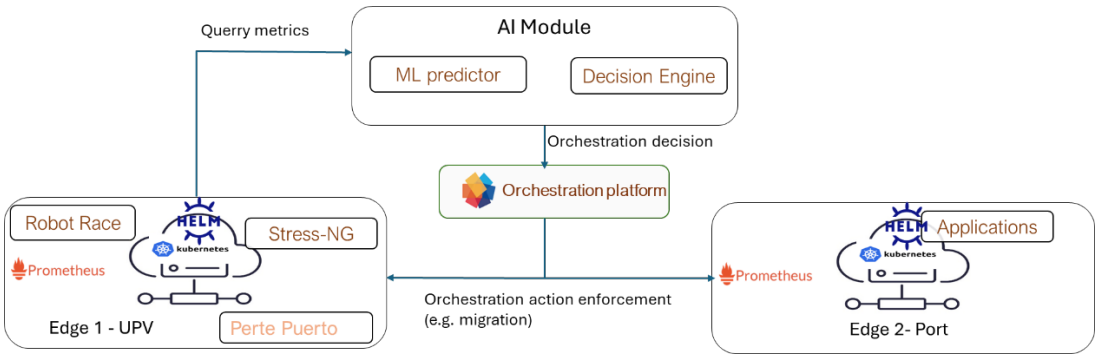


Figure 20: Demonstration high level architecture

The architecture is composed of a number of components interconnected through enabling interfaces including the following the zero-touch closed loop service orchestration framework discussed in previous deliverable D2.2.

The core components of the demonstration architecture are the following

4.1.1.1. Infrastructure

The demo is executed over the edge infrastructure located in Valencia, where the 5G IoT-Edge-Cloud architecture for digital twins in industrial and logistics environments was designed and deployed. The infrastructure is composed on 2 cloudlets/clusters thus, Edge 1-UPV at UPV and Edge 2 -Port at the port of Valencia. More concretely, the demo setup is deployed in the computing infrastructure, based on Kubernetes, provisioned at the UPV and port edge servers. The infrastructure is responsible for hosting the two applications provided by Tyris (Robot Race and Perte Puerto) and other requisite

components such as the observability agents for aggregating system metrics and 3rd party applications and ML models.

4.1.1.2. Orchestration platform

The ADV5G-TWINS edge orchestration platform, NearbyOne is the high-level service orchestration platform, acting as a single-pane-of-glass for all underlying operations, such as the lifecycle management of services and infrastructure. The orchestrator is responsible for enforcing all orchestration actions such as service scaling, migration, placement and termination, among others.

4.1.1.3. Prediction Analytics Engine

The *Prediction Analytics Engine as part of the AI module* leverages a machine learning model for forecasting system state based on the historical data aggregated in the observability stack. These ML models managed via mlflow are used to analyze historical and real-time metrics to predict future system requirements, enabling proactive resource allocation.

4.1.1.4. Decision Engine

The decision engine as part of the AI module leverages the results of the analytics and the current system state and user required QoS to derive orchestration actions that can satisfy the user QoS given the real-time system state.

4.1.2. Demo scenario description

The proposed AI -driven service orchestration operates within a closed-loop control framework, continuously monitoring real-time resource consumption patterns, with particular emphasis on CPU utilization across active applications. By applying advanced predictive analytics developed in the project, the framework accurately forecasts future workload behavior, enabling proactive decisions on computational resource distribution and application placement when desired.

These predictive insights play a vital role in maintaining optimal resource allocation aligned with application demands. By anticipating workload fluctuations, the system can dynamically adjust resource levels to prevent performance degradation or inefficient utilization. Guided by AI-driven forecasts, the orchestration platform can autonomously scale application resources up or down in real time. This adaptive management approach ensures that applications receive sufficient computational power while avoiding unnecessary allocations, effectively mitigating the risks of both over-provisioning and under-provisioning. Over-provisioning involves allocating more resources than required leading to wasted capacity and increased operational costs. In contrast, under-provisioning can degrade performance through higher latency and possible service interruptions.

However, in practice, a single server supports multiple applications. Therefore, the available computational resources on a physical server may reach their upper limit, preventing further scaling of applications. This can occur due to hardware constraints, such as limited CPU cores, memory, or storage capacity, or due to power consumption restrictions that limit the ability to allocate additional resources. In such cases, simply increasing resource allocations is not a viable solution, as it may lead to system instability or degraded performance and service disruptions.

Therefore, to tackle this challenge, thisADV5G-TWINS-ORCHESTRATION orchestration scenario leverages an intelligent migration strategy. Instead of allowing critical and other applications to suffer from resource shortages of a single resource constrained server, the system identifies and prioritizes workloads based on their importance. Non-essential or lower-priority applications are seamlessly relocated to alternative nodes within the network infrastructure, freeing up essential computing resources for high-priority applications.

Moreover, this migration process is executed with minimal service disruption, ensuring that all applications continue functioning efficiently while optimizing overall resource utilization. To enhance the efficiency of this migration strategy, the system leverages predictive analytics to anticipate resource bottlenecks before they occur. In addition, by proactively identifying potential scaling limitations, the orchestration mechanism can initiate early migration of non-critical applications, thereby preventing sudden performance degradation.

Specifically, as depicted in Figure 21 below, the scenario starts with UPV Edge server 1 hosting the Robot Race application and a stress-NG application designed with flexibility to stress the server in terms of computational resources. In practice, this could be any other 3rd party application contending for resources on the same server with the Robot Race application. As the CPU requirement of the stress-NG App is gradually increased, the overall CPU utilization of Edge 1 server increases, resulting in an increase in response time of the Robot race App. This CPU utilization is continuously monitored by the observability agents and future forecasts derived by the ML prediction analytics. Based on these prediction results together with the acceptable QoS thresholds, the DE triggers migration of the stress-NG App to the Port Edge 2 whenever the anticipated CPU utilization exceeds 70%. This migration frees up space for the Robot race App and enabling up scaling of the Stress-NG application without violating QoS of the Robot Race App.

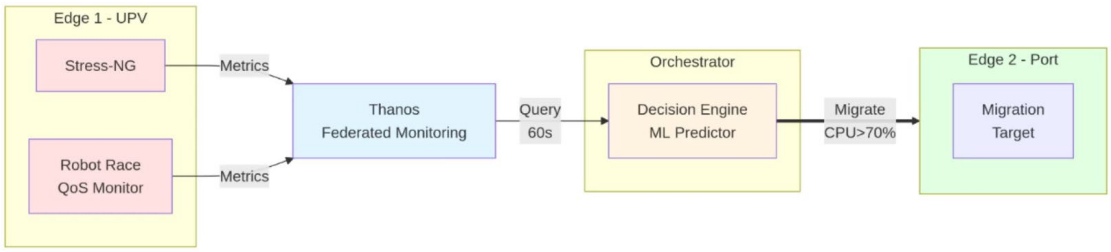


Figure 21: Demo scenario

The developed AI-based service orchestration leverages a closed-loop orchestration framework introduced in Deliverables D2.1 and highlighted in section 2.1 of this deliverable where feedback from the underlying infrastructure continuously informs the decision-making process regarding the application lifecycle management. The scenario considered involves a closed loop mechanism as shown in Figure 22 involving

- Continuous real-time monitoring of the resource consumption patterns on the server nodes with focus on CPU utilization of running applications. This is enabled through observability monitoring agents that were deployed on the two clusters of the infrastructure
- Continuous forecasting of the CPU needs of the applications based on the current resource consumption. This is achieved using the developed attention-based informer model for workload forecasting.
- Proactive decision making regarding whether to scale resources up or down and dynamically reassign or migrate workloads if needed. These adjustments are made in real-time ensuring that the applications always receive the required computational resources without over-allocating resources.

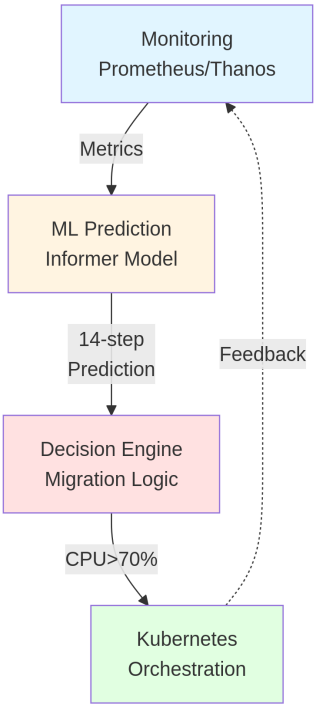


Figure 22: Scenario closed loop mechanism

4.1.3. Demo setup procedure

To realize the demo, the following activities were undertaken.

4.1.3.1. Infrastructure setup and provisioning

To enable the deployment of the applications and various components of the closed-loop framework including the observability agents, decision engine, and the AI models, a compatible and well-configured infrastructure is needed to provide the requisite foundational environment. The ADV5G-TWINS project and demonstration activities are executed on a robust hardware and software foundation where Linux OS is configured, and the container provisioning process is initiated. This process culminates in setting up Kubernetes clusters, which are onboarded into the NearbyOne Orchestrator as part of the orchestrated infrastructure.

Accordingly, the following steps have been implemented to support the two activities of the project, thus A7 and A8:

- **Virtual Machine (VM) Provisioning**: Two virtual machines have been provisioned with the Rocky Linux operating system. The two virtual machines are targeted to emulate a distributed network infrastructure and support application migration use cases.
- **Containerization Layer**: Docker has been installed to serve as the containerization layer, enabling the encapsulation and efficient management of application components within containers.

- **Container Orchestration Platform**: Two RKE (Rancher Kubernetes Engine) Kubernetes clusters have been deployed on the two virtual machines. Each cluster functions as the container orchestration platform, managing the lifecycle of containerized workloads. It also acts as the target Kubernetes cluster where the NearbyOne Orchestrator will deploy orchestrated services. This Kubernetes cluster is hence forth referred to as the *Cloudlet*.
- **Communication Agent**: To establish seamless communication between the NearbyOne Orchestrator and the Cloudlet, a communication agent has been installed within each Kubernetes cluster. This agent ensures efficient coordination and data exchange between the orchestrator and the Cloudlet, enabling the automated deployment and management of services.
- **Infrastructure Onboarding** : Once the deployed infrastructure has been onboarded on the NearbyOne orchestrator, this becomes visible under the infrastructure menu of the NearbyOne dashboard. This demonstrates that the orchestrator has visibility to this infrastructure irrespective of infrastructure`s geographical location, hence is able to communicate with and reconfigure applications deployable on this infrastructure. The two provisioned cloudlets can be seen under the ADV5G-TWINS project organization of the NearbyOne dashboard as shown in Figure 23 under the *infrastructure* tab of the dashboard.

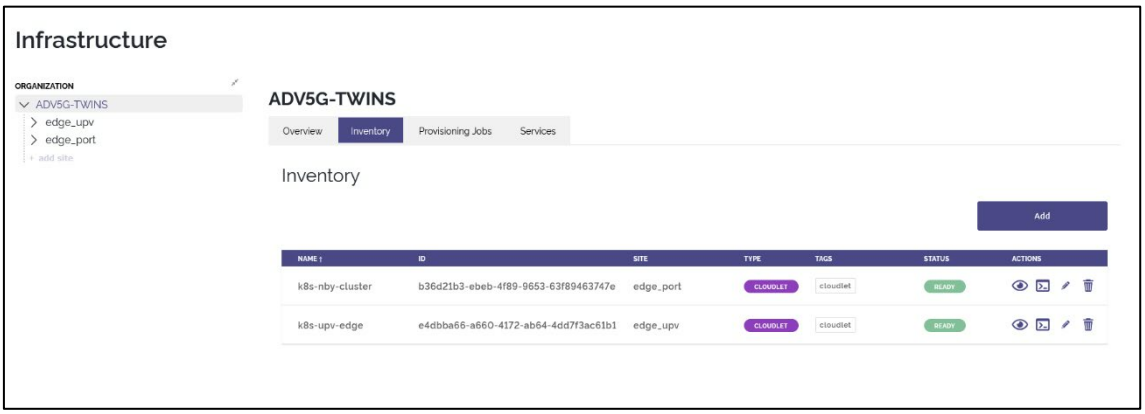


Figure 23: The two provisioned cloudlets

4.1.3.2. Deployment of orchestration platform

The ADV5G-TWINS project's NearbyOne Orchestrator functions as the central service orchestration platform, offering a unified "single-pane-of-glass" interface for overseeing all underlying operations. These operations encompass the lifecycle management of both services and their supporting infrastructure. To enable this capability, the following steps have been implemented:

- Deployment of the orchestration platform: The ADV5G-TWINS orchestration platform has been successfully deployed on an AWS Elastic Kubernetes Service (EKS) cluster. This ensures a robust, scalable, and cloud-based foundation for orchestration operations.
- Cloudlet integration: The two cloudlets, established in the prior setup phase, have been onboarded into the ADV5G-TWINS orchestrator. This integration allows the orchestrator to manage and deploy services seamlessly within these cloudlets' environments.
- Creation and onboarding of Nearby blocks: Several key components, referred to as *Nearby blocks*, have been created and integrated into the ADV5G-TWINS Orchestrator. The Nearby blocks describe how to deploy applications and network functions, including different aspects such as how to render their configurations, where to place them across the registered clusters, how many instances to deploy (statically or dynamically decided at runtime) and how to monitor their KPIs, among others. The blocks include the deployed applications and decision engine among others.

Once the infrastructure is deployed and onboarded on the orchestration platform, this infrastructure is visible from the dashboard of the nearby orchestration platform.

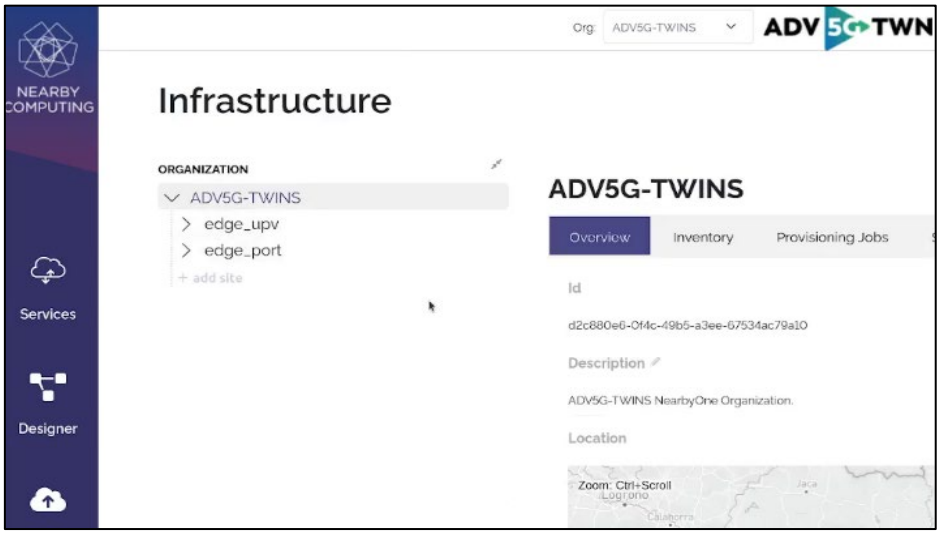


Figure 24: the two distributed clusters as seen from the GUI of Nearbyone Orchestrator

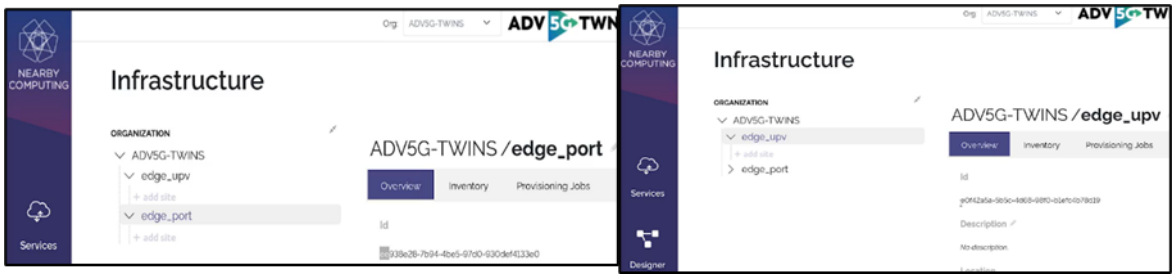


Figure 25: Unique IDs of the two clusters as seen from the NearbyOne GUI

4.1.3.3. Deployment of observability stack

The observability stack components (see deliverable *D2.1: Evolución de la arquitectura de las plataformas IoT-Edge-Cloud y soluciones de AI/ML para gemelos digitales*) have been deployed in the UPV edge server via the NearbyOne GUI to collect telemetry and monitoring data. Figure 26 displays, highlighted in yellow, the running instance of the Observer block in the UPV edge server.

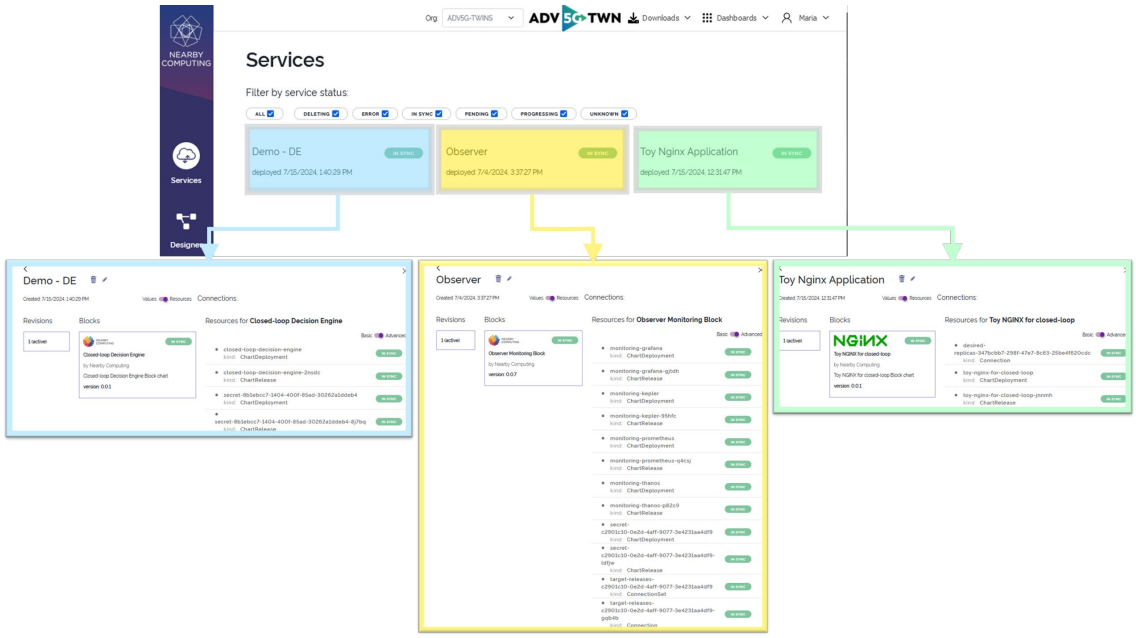


Figure 26. some of demo components deployed via the NearbyOne dashboard.

To enhance monitoring and performance analysis, a custom plugin was developed to report application metrics to the orchestrator's observability stack. This plugin includes a Prometheus Exporter that initializes alongside the application and exposes task-specific processing metrics to Thanos (see Figure 27), the orchestrator's centralized monitoring hub. For each application task (e.g., Lane Visualization, Radar Visualization), the following metrics are made available via Prometheus queries :
- node_avg_durations_ms: Average processing time

- node_curr_durations_ms: Current processing time
- node_div_durations_ms: Standard deviation of processing time
- node_min_durations_ms: Minimum processing time
- node_max_durations_ms: Maximum processing time

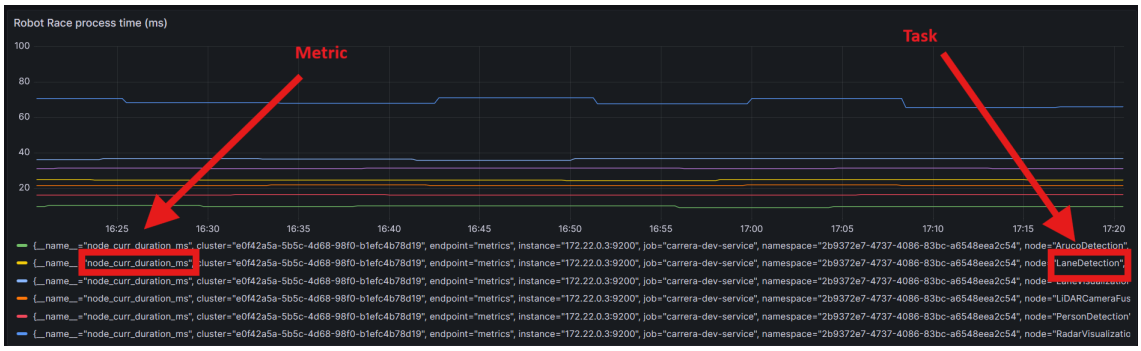


Figure 27: Monitoring of Robot Race performance KPIs through the Grafana dashboard.

4.1.3.4. Application deployment

As part of this project, two applications were successfully integrated into the NearbyOne Orchestrator by implementing their respective Blocks: Perte Puerto and Robot Race. These are shown in Figure 28.

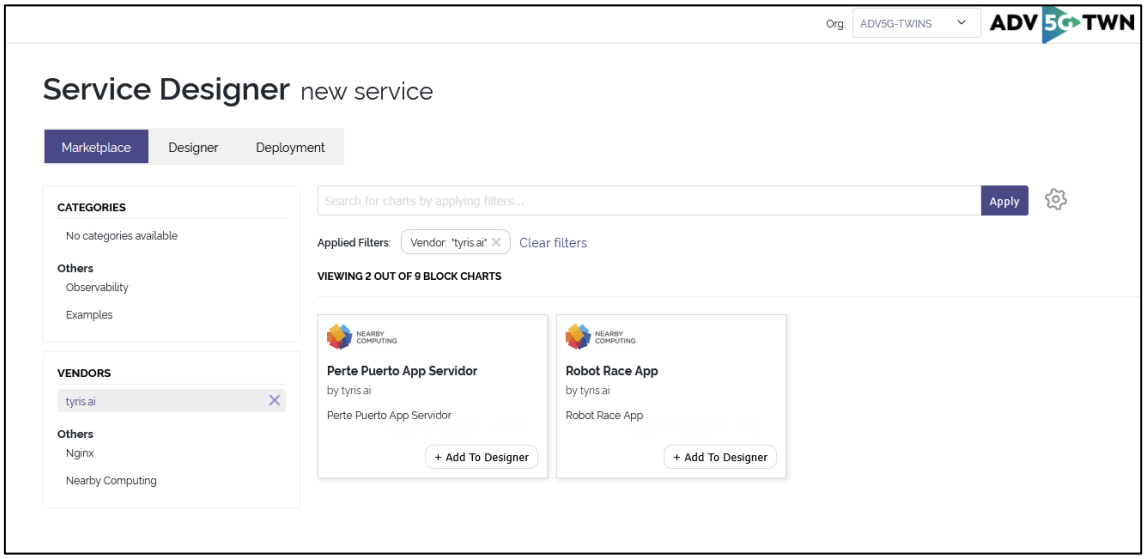


Figure 28: The two Tyris applications as seen from the GUI of NearbyOne

Perte Puerto Application

The Perte Puerto application was already available as a Helm Chart, which streamlined the process of wrapping it into a NearbyOne Block. The integration focused on exposing key configuration parameters to the orchestrator (see Figure 29), including:

- Number of replicas
- Server port
- Container image version

This allowed for dynamic and seamless configuration directly from the orchestrator interface.

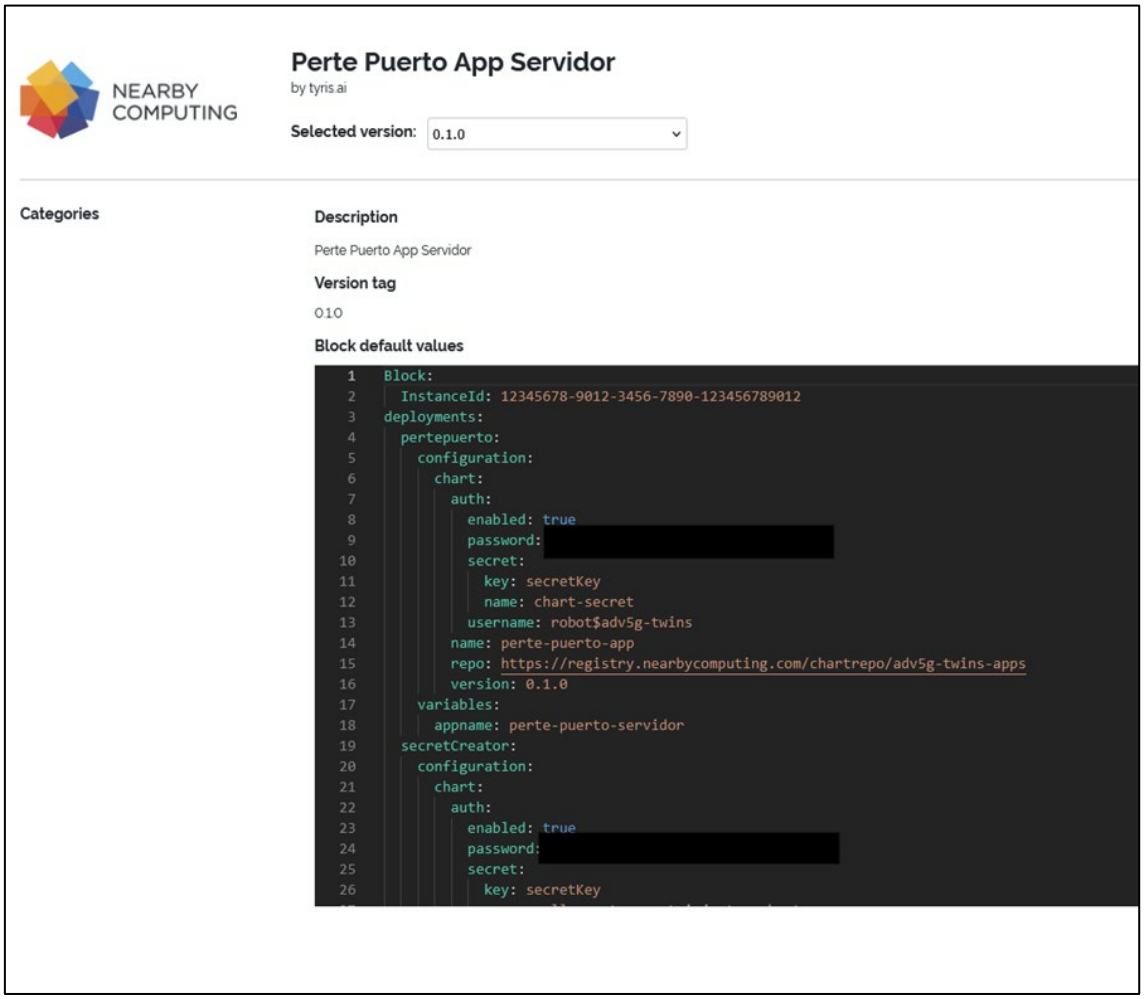


Figure 29: Configurations for the Perte-Puerto App

Robot Race Application

Unlike Perte Puerto, the Robot Race application was only available as a container image. Therefore, the first step was to develop a Helm Chart to facilitate its integration into the NearbyOne ecosystem. The configurations for the application are shown in Figure 30. .

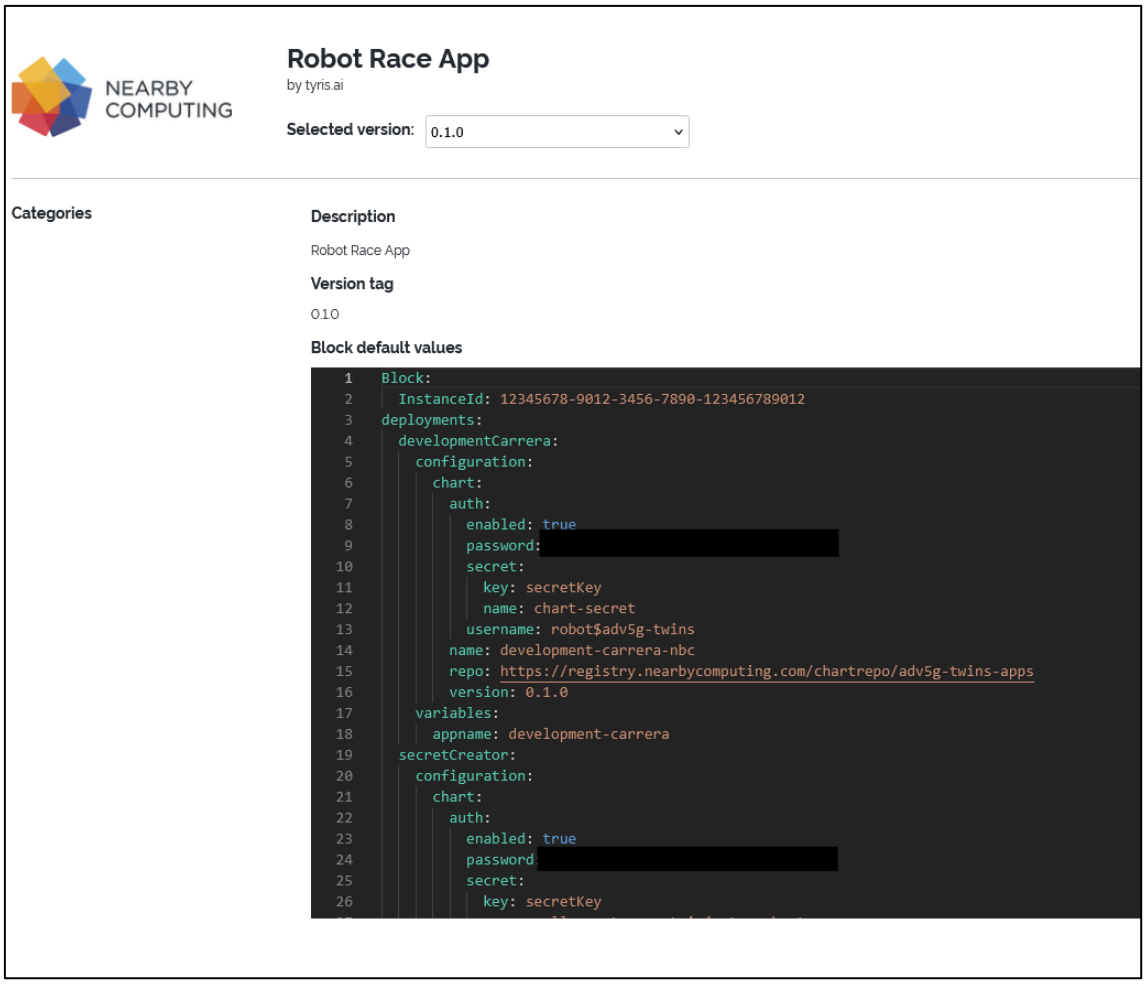


Figure 30: Configuration for the Robot Race App

4.1.3.5. Decision Engine Integration

The Decision Engine (DE) takes as input the prediction results from the analytics and the current system state to generate orchestration actions to be enforced to the underlying infrastructure by the Orchestrator. In the current implementation, the DE logic is implemented as a heuristic algorithm based on a Python script to determine when to trigger the orchestration action and which action to trigger based on the prediction results. The DE has been provisioned to the edge cluster through the NearbyOne Orchestrator and deployed as a containerized application as seen from Figure 31 showing all components onboarded onto the NearbyOne orchestration platform.

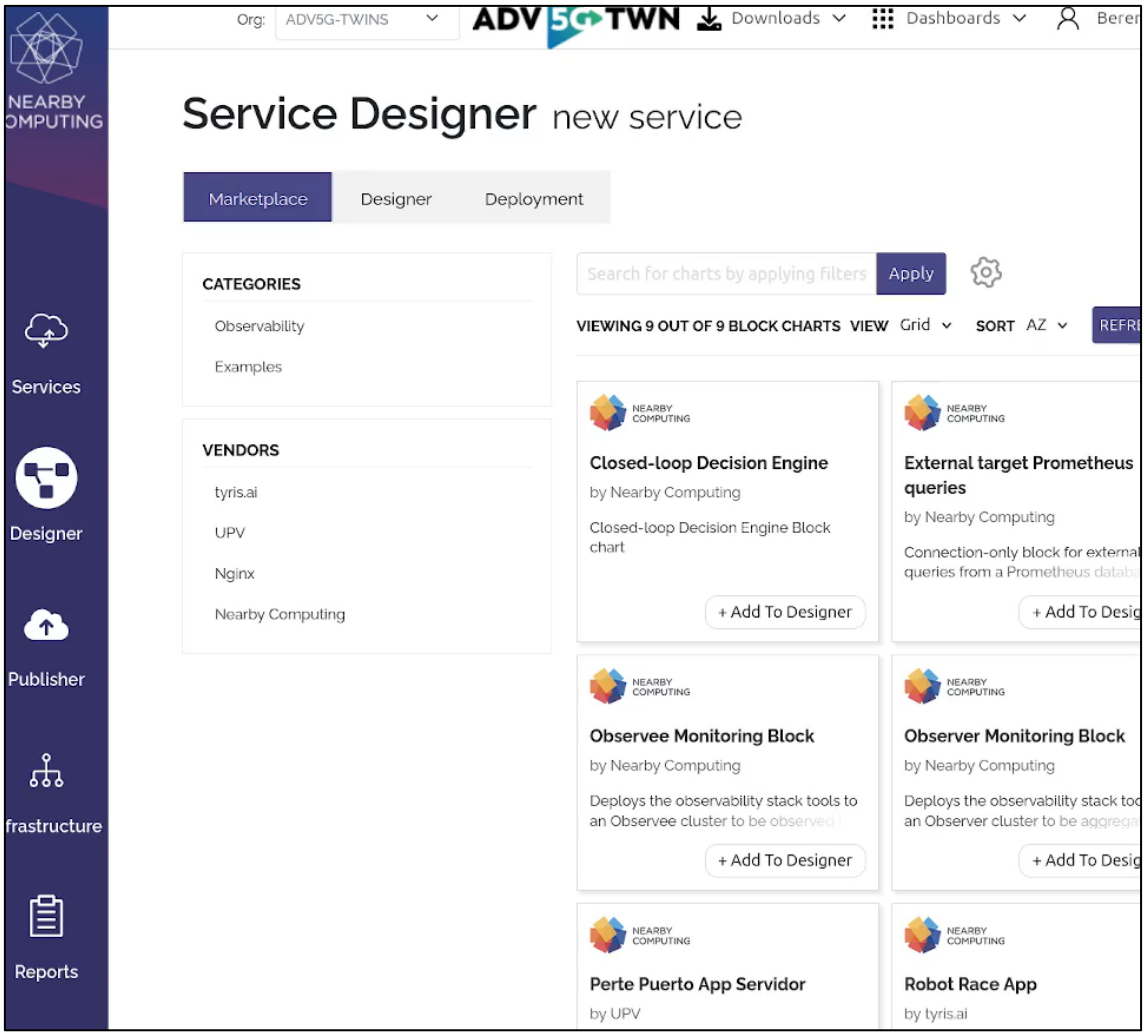


Figure 31: Deployed components including the DE as seen from the NearbyOne GUI

4.1.3.6. ML-based predictor Integration

The ML-based predictor system employs a modular architecture with five primary components working in coordination:

- The Main Orchestrator manages the closed-loop process through YAML configuration files, supports multiple predictor algorithms, operates on 60-second cycles, and maintains comprehensive logging capabilities.
- The Data Querier interfaces with Thanos for historical metrics retrieval, executes Pod CPU usage queries with configurable time windows, monitors replica counts, and performs multi-pod aggregation.
- The Prediction Engine provides three model implementations: ARIMA for statistical forecasting, LSTM for deep learning sequence prediction, and Informer for advanced transformer-based forecasting. All implementations feature standardized interfaces, power consumption estimation, and MLflow integration.

- The Scaling Logic implements rule-based decision making with threshold-based scaling, binary scaling strategies, and CPU percentage-based algorithms.
- The data collection framework uses CPU utilization metrics as core input, pod replica counts for scaling state representation, CPU resource limits for scaling constraints, and temporal information for time series analysis.

Data preprocessing

Data preprocessing implements time series standardization using StandardScaler, CPU utilization normalization to 0-1 range, and temporal feature engineering with multiple granularities including minute, hour, day, week, month, and seasonal patterns.

The system uses the Alibaba Cluster Trace for Microservices version 2022 for Informer model training. This dataset contains metrics from nearly 20,000 microservices across 10,000+ bare-metal nodes over 13 days, providing over 470,000 container records and 20+ million call graphs for realistic workload pattern training.

Model training

The training infrastructure leverages MLflow for experiment tracking, model versioning, and artifact management with MinIO S3 storage. The Informer training employs the OmniFORE framework with seven stages as discussed in deliverable D2.3: spatio-temporal feature extraction, latent space clustering, trace labeling, representative training selection, data scaling, attention mechanism implementation, and Bayesian optimization for generalization.

The process transforms container traces into latent space using encoder-decoder CNN architecture, groups similar patterns through unsupervised clustering, ensures training diversity through proportional cluster sampling, and implements individualized trace normalization.

Performance benchmarking shows Informer consistently outperforming baseline methods. For 24-step predictions, Informer achieves MSE 0.577 vs LSTM's 0.650. Extended 720-step predictions demonstrate 38% improvement with MSE 1.215 vs LSTM's 1.960.

Validation employs the predictor_informer_test.py framework with systematic hypothesis testing including high utilization scenarios (60-100% CPU), realistic low utilization validation (3.8-15% CPU) using actual container trace data, GitHub performance threshold testing at documented degradation points (12%, 20%, 30%, 61%), and comprehensive CPU/GPU model comparison across identical test scenarios.

AI Model Integration Architecture

The integration pipeline connects machine learning predictions with orchestration actions through sophisticated real-time inference capabilities.

- Model serving utilizes MLflow lifecycle management with Triton Inference Server for high-performance inference across CPU and GPU environments. Containerized deployment supports flexible hardware configuration with optimized model artifacts.
- Data flow integration standardizes interfaces between monitoring infrastructure and ML models through Thanos queries, consistent preprocessing pipelines, and temporal feature extraction across multiple granularities.
- Prediction integration implements unified interfaces abstracting algorithm complexity, supporting dynamic model selection, and providing structured results with confidence intervals and metadata.
- Decision integration translates predictions into orchestration actions through rule-based logic and threshold-based evaluation.

Model forecasting

The algorithmic design centers on CPU utilization as the primary quality-of-service indicator. CPU consumption serves as the dominant predictor of system performance and energy consumption, accounting for 40-60% of total system energy with strong correlation to overall performance metrics. The prediction algorithm operates on six-step forecasting horizons, generating CPU utilization predictions for the next six minutes using historical sequences of 30-time steps. The forecasting combines historical utilization with temporal features including hour-of-day, day-of-week, and seasonal patterns, maintaining prediction accuracy within 2% tolerance for utilization levels below 70%.

4.1.4. Demo Results

In this section, we present the results of the experimental setup that demonstrate the scenario described in Section 2.2. Specifically, the demo execution proceeds through the following sequential steps:

- Deployment of all necessary components, including infrastructure, applications, the decision engine, AI models, and observability modules as discussed in the setup procedure.
- Deployment of the Stress-NG and Robot Race applications on the Edge1-UPV server, which acts as the primary cluster for executing both workloads.
- Gradual variation of the traffic load directed to the Stress-NG application.
- Continuous monitoring of CPU utilization on the primary cluster.

- When the forecasted CPU utilization exceeds the 70% threshold, thus indicating a potential QoS violation for the Robot Race application, the Stress-NG application is proactively migrated to the Edge2-Port cloudlet. This migration alleviates load on the primary cluster, ensuring continued QoS compliance for the Robot Race application.
- The Robot Race application consistently maintains response times within the acceptable threshold of less than 100 ms.

The key extracted snapshots of the Demo are described below.

4.1.4.1. Initialization

At initialization, the different components of the Demo are instantiated as shown in Figure 32 of the NearbyOne orchestrator GUI.

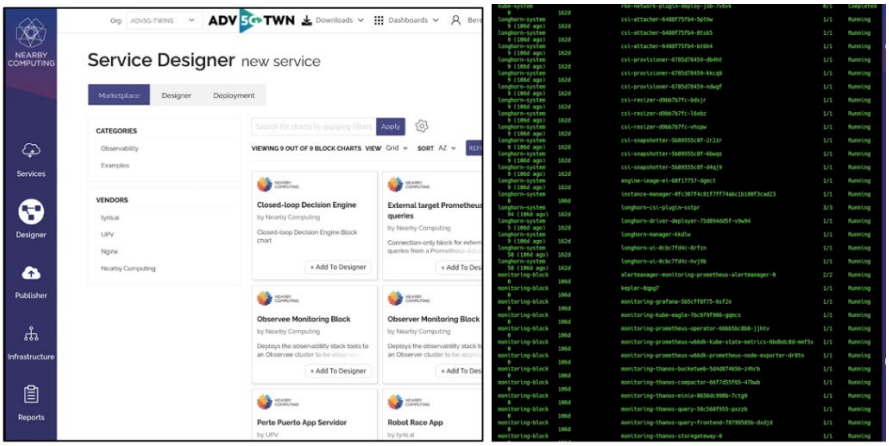


Figure 32: Instantiated components (left) and running state (right)

4.1.4.2. Continuous monitoring

The system continuously collects performance metrics related to the CPU utilization of the cluster Infrastructure and response time of the Robot Race App. This is shown in Figure 33. The response time of the App is directly affected by the utilization levels of the node with the 100ms violation occurring at 70% CPU utilization threshold.

Figure 33: Cloudlet CPU utilization (bottom) and the corresponding response time of the Robot Race App (top)

4.1.4.3. Migration trigger with QoS violation

As the CPU load increases, the 70% utilization threshold violation is predicted by the AI module. In this way, the DE triggers proactive migration of the stress-NG application to the new cloudlet Edge-2 port as seen from the site ID in Figure 34.

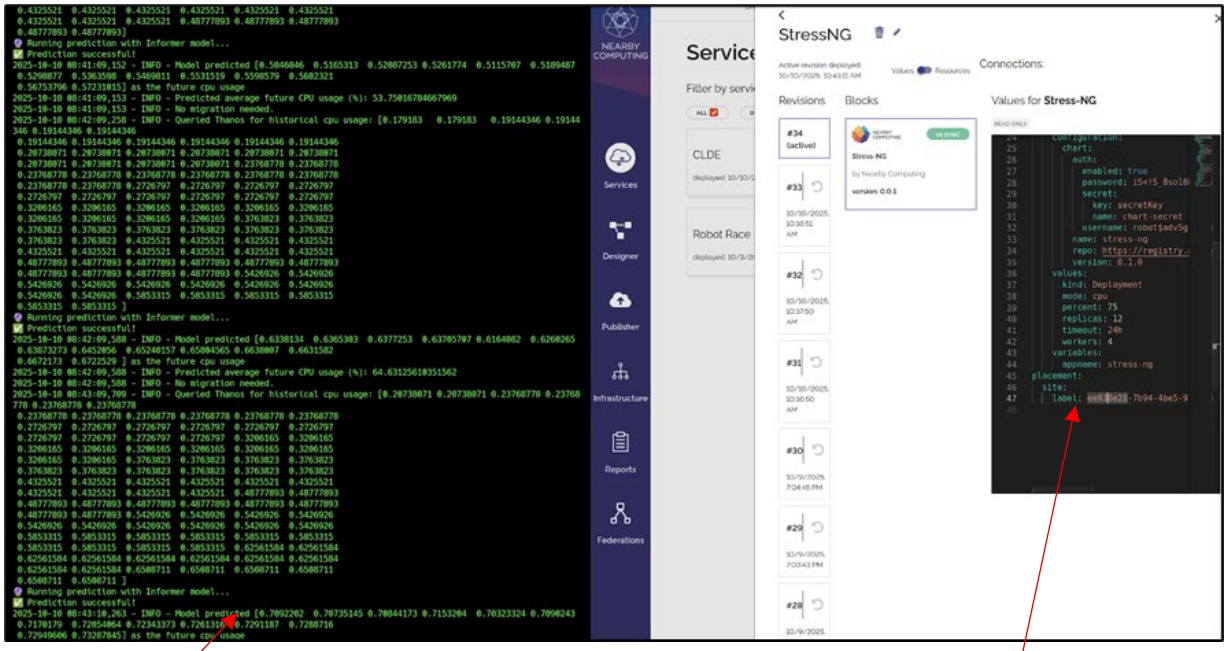


Figure 34: 70% threshold violated migration triggered

4.1.4.4. Compliance with QoS

As seen from Figure 35, after migration of the Stress-NG app, the CPU utilization of the cloudlet drops keeping the response time of the Robot Race application confined within the 100 ms.

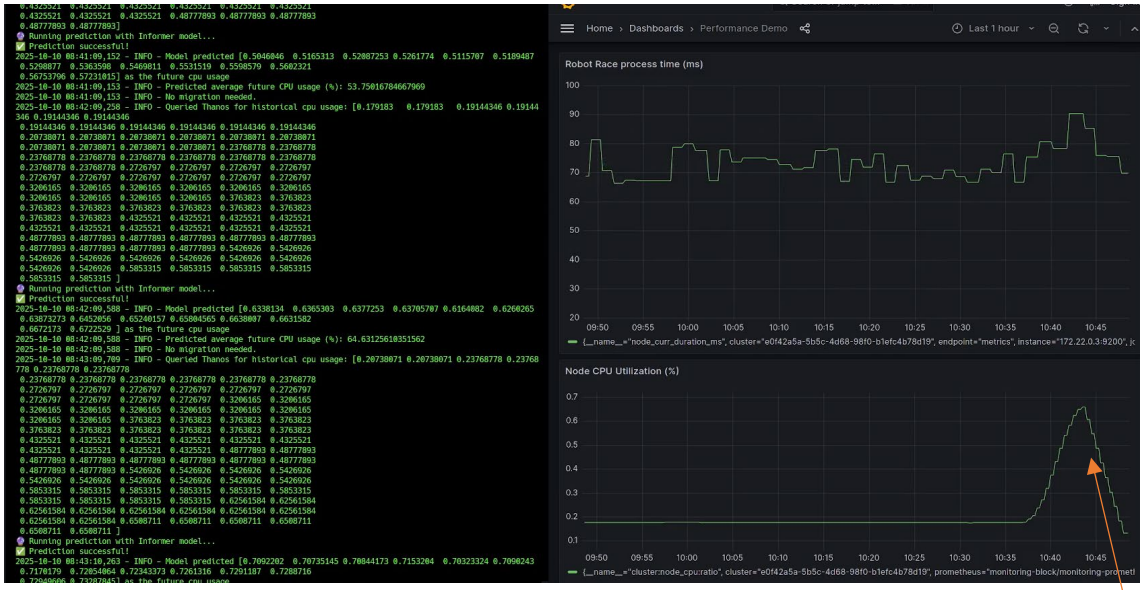


Figure 35: CPU utilization drops after migration, but processing time remains stable

The results of the experimental setup served have validated several key capabilities of the ADV5G-TWINS orchestration platform. The demonstration showcased the platform's ability to operate effectively across heterogeneous distributed network environments, while leveraging cloud-native infrastructure principles. The following capabilities were highlighted:

- Cross-layer visibility and orchestration: The platform demonstrated end-to-end visibility across distributed clusters, enabling informed orchestration decisions that account for the diverse characteristics of different infrastructure within a unified cloud-native framework.
- Dynamic resource utilization: The developed mechanisms facilitated adaptive and efficient allocation of resources, ensuring optimal utilization under varying network and application conditions typical of cloud-native edge-cloud domains.
- Comprehensive application life-cycle management: The platform successfully enforced full life-cycle operations of applications including

instantiation and migration which are essential for maintaining service continuity and performance.

- Scalability and extensibility: The orchestration framework exhibited scalability through seamless integration and onboarding of third-party logic components, implemented as Nearby blocks, thereby enabling flexible system expansion.
- Interoperability with external decision systems: The platform proved capable of receiving orchestration inputs from external control loops and third-party decision engines, ensuring interoperability and collaborative orchestration.
- Distributed infrastructure management: The orchestration layer effectively accessed and reconfigured computing clusters distributed across multiple cloudlets including migration of applications across cloudlets, highlighting its capability in managing geographically distributed resources and infrastructure.
- Real-time QoS compliance: The platform instantiated and migrated applications dynamically, responding in real time to network conditions to maintain QoS targets while avoiding unnecessary resource overprovisioning.

4.2. Activity 7: Energy efficiency

This section discusses the demonstration for Activity A7 including the architecture and the scenario for this activity, demo setup and results.

4.2.1. Demo architecture

The demonstration scenario leverages architecture similar to that described in D4.3 related to activity A8 of the ADV5G-TWINS project. The architecture leverages the closed-loop framework described in deliverable D2.2 composed of the cloud-native infrastructure in Valencia, the AI module constituted of the analytics and decision engine and the orchestration platform levering the NearbyOne orchestrator. As shown in *Figure 37* , the architecture follows a closed loop feedback mechanism where the system energy state is continuously monitored and forecasted to derive real time energy efficient orchestration actions.

The flow involves continuous monitoring of the power consumption of the compute nodes through energy measurements components including Kepler that was installed on the server and the current CPU consumption of the server. These power readings (in watts) are collected by Prometheus, which acts as the monitoring backbone of the system making these metrics available to other components in near real time.

The Informer ML model then consumes these metrics to predict future CPU utilization for the upcoming time steps (e.g., 14 intervals ahead). This prediction anticipates how workloads will evolve, allowing proactive decision-making rather than reactive scaling. Based on these CPU states, the Power Model translates the predicted CPU utilization into expected power consumption (watts). This allows the system to forecast how much energy will be used if current trends continue in terms of CPU usage. Different from the AI logic presented in activity A8, the AI module in this activity predicts both CPU and energy Consumption.

```
INFO - Decision Engine is running
INFO - Queried Thanos for historical cpu usage: [0.25, 0.26, 0.28, ...]
INFO - Model predicted [120.5, 125.3, 128.7, 132.1, 135.8, 140.2] as future power consumption
INFO - Predicted average future power consumption W 130.4
INFO - No scaling...
```

Figure 36: Joint CPU and Power prediction from the AI module

Then, Decision Engine (DE) continuously evaluates these predicted power levels. If the forecast indicates that consumption will exceed a defined threshold (350W in this case), it triggers scaling actions through Kubernetes. Kubernetes then adjusts the active resources by scaling down, workloads, or spinning up new containers to optimize both performance and energy efficiency. The scenario for workload migration was demonstrated in Activity A8. In this demo, we tame energy consumption violations through proactive service scaling.

After scaling, the system's new operational state produces updated metrics (reflecting the new CPU load and power consumption). These are again captured by Prometheus, closing the loop. The updated data feeds back into the Informer ML model, which it can use to refine its predictions and trigger the next cycle.

Because this entire sequence repeats automatically and continuously, the system forms an adaptive feedback loop. It learns from real-time data, predicts upcoming resource needs, adjusts accordingly, and verifies the effects all with the goal of maintaining optimal energy efficiency and QoS in a cloud-native IoT-edge-cloud environment.

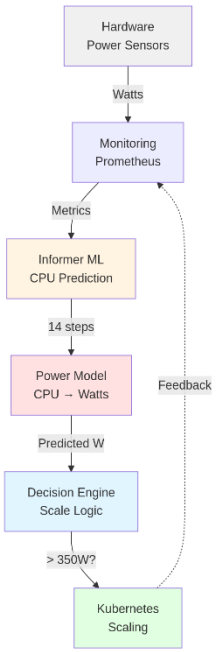


Figure 37: Closed loop flow diagram

4.2.2. Demo scenario

Within the ADV5G-TWINS project, a key objective is to enhance the energy efficiency and performance of applications and services deployed across distributed, cloud-native IoT-edge-cloud infrastructures. Achieving this goal requires not only maintaining the desired Quality of Service (QoS) but also optimizing the use of computational and network resources to minimize energy consumption, an increasingly critical factor in large-scale 5G and 6G environments. Traditional static resource management approaches often result in inefficiencies, where over-provisioning leads to unnecessary power usage and operational costs, while under-provisioning degrades application performance and reliability. To overcome these limitations, the ADV5G-TWINS project employs AI-driven orchestration mechanisms that enable intelligent, proactive, and adaptive resource allocation and service management. By forecasting workload and resource demands in real time, the framework dynamically scales and reallocates resources nodes, ensuring sustained QoS while reducing energy waste and promoting sustainable and efficient operation of future cloud-native networks.

4.2.3. Demo flow procedure

The Demo execution procedure is shown in the flow diagram of Figure 38 with the following flow:

- A Stress-NG app is installed along with the Tyris application at the UPV server operating at 350w power budget as the acceptable maximum power threshold. The Stress-NG app is chosen due to the ease of manually varying the resource requirements on the server as opposed to the realistic Tyris applications. In this way, the replicas can be varied from 1 -4-8 to represent increasing computational demand from the UPV server node
- Keppler monitors power consumption in real time, and these metrics are collected by Prometheus
- The DE in energy mode queries Prometheus every 60 seconds for the aggregated metrics. If the predicted power exceeds the 350w budget threshold, then a scale down trigger is activated decreasing the number of replicas for the Stress-NG application.

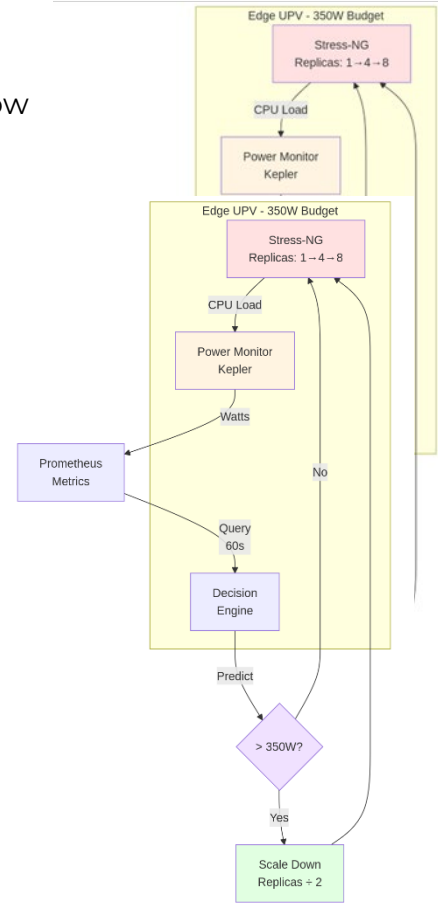


Figure 38: Demo execution flow

4.2.4. Demo setup

To realize the Demo, a number of steps were executed. Most of the steps executed for the demonstration of activity A7 are similar to these documented in D4.3 for activity A8. These steps include

- Deployment of requisite Infrastructure involving provisioning of the virtual machines which are shared by the two activities and installation of the auxiliary components such as observability agents.
- Deployment of orchestration platform
- Deployment of applications provided by Tyris and Stress-NG with capability to stress the server in terms of computation resource requirements

However, the two activities differ in terms of the AI logic. Whereas A8 focused on forecasting CPU utilization to enhance orchestration performance, activity

A7 focusses on power consumption prediction to enhance energy efficiency. The AI logic implemented two models in this case.

- A CPU prediction model leveraging the Informer based ML model to predict CPU utilization
- A power model which converts CPU predictions to Watts using calibrated coefficients

The power model uses a polynomial function: Power = a + b*CPU + c*CPU² where coefficients are derived from empirical measurements of the UPV node based on the best fit energy prediction modelling framework discussed below

4.2.5. Best Fit Energy Prediction Modeling Framework Implementation

The energy efficiency framework implements Best Fit Energy Prediction Modeling (BFEPM) through the energy_fit_node.py measurement tool. The framework evaluates five power model types: Linear, Quadratic, Cubic, Power, and Gaussian using controlled CPU stress testing from 5% to 70% utilization. Model selection employs statistical criteria including R-squared, adjusted R-squared, and RMSE.

Real measurements from two server configurations validate the approach: UPV_node (Intel Xeon Gold 6338N, 64 cores) achieves best accuracy with cubic models (0.38% mean error), while PC_berend (Intel i9-9900K, 16 cores) shows optimal performance with cubic models (3.52% mean error). The PowerModelFitter class implements all five model fitting algorithms with automatic best model selection based on adjusted R-squared values.

The measurement methodology follows systematic temporal progression with controlled CPU stress levels. Target utilization increases incrementally from 5% to 70%, with each level maintained for sufficient duration to achieve equilibrium. The measurement timeline demonstrates precise control between target and actual CPU utilization, with corresponding power consumption measurements exhibiting clear correlation patterns.

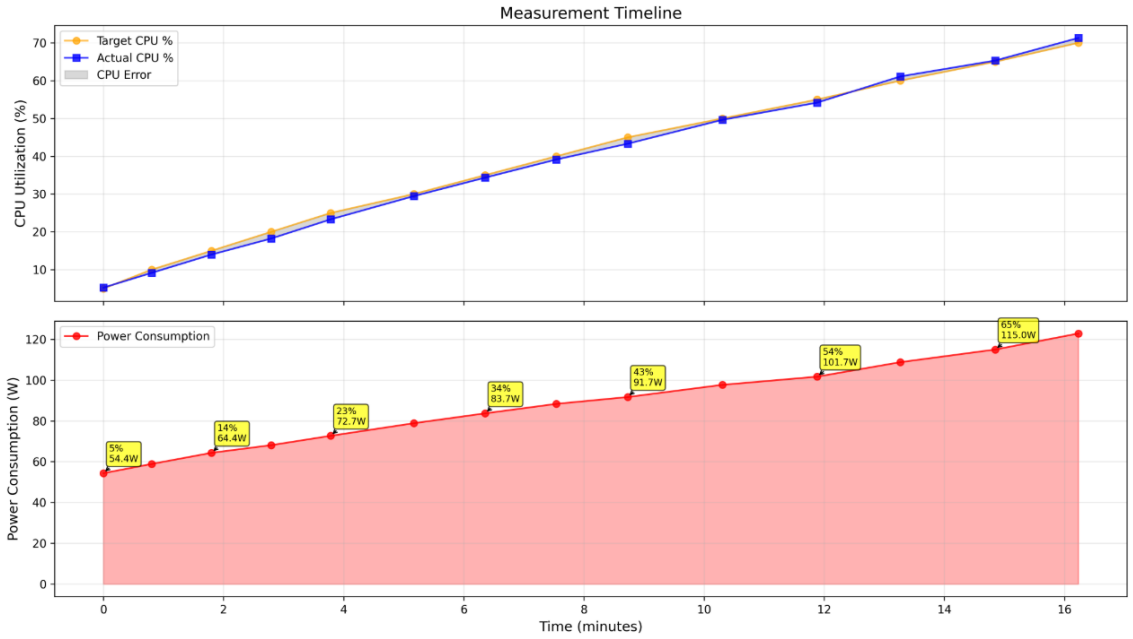


Figure 39: CPU and power consumption profile

The primary power model analysis reveals the optimal cubic polynomial relationship between CPU utilization and power consumption for the UPV server configuration. Measured data points show strong correlation with the fitted model, achieving high R-squared values and minimal prediction errors. The model accuracy comparison validates the CPU load control precision, with actual utilization closely tracking target values across all measurement points.

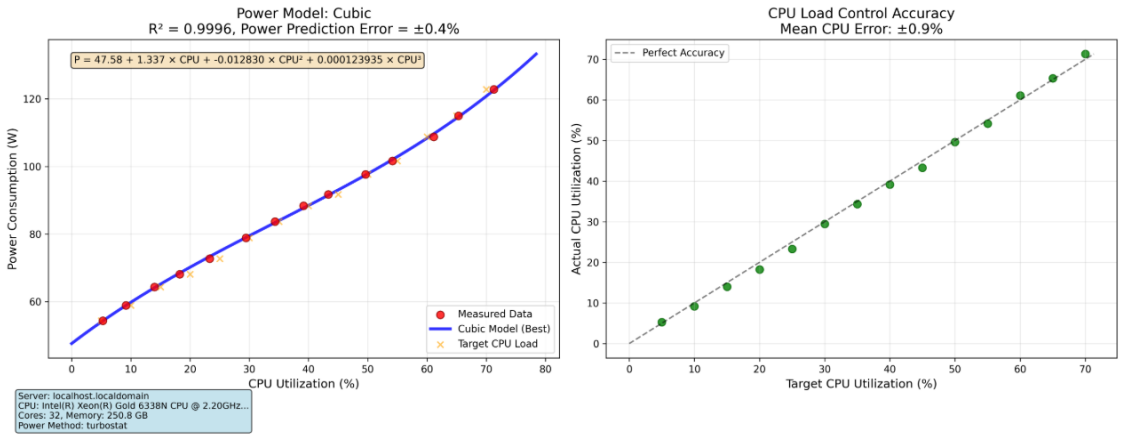


Figure 40: Cubic model performance

Comprehensive model comparison across all five power model types demonstrates the superiority of the cubic approach for this server configuration. Linear, quadratic, power, and Gaussian models show varying degrees of fit quality, with statistical analysis revealing the cubic model's

optimal balance between complexity and accuracy. Model residuals analysis confirms consistent prediction quality across the full utilization range.

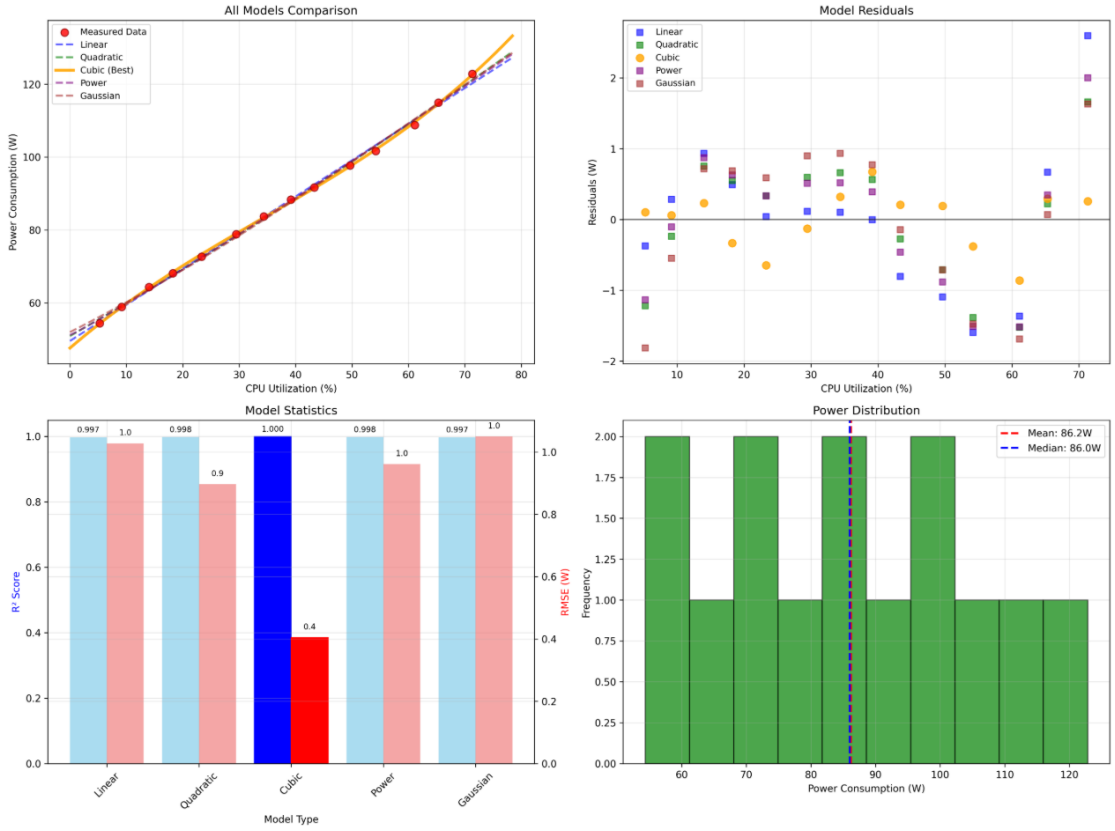


Figure 41: Performance of the different power models

4.2.6. Demo Results

The Demo showcases energy-aware orchestration. Instead of reacting to CPU utilization, we'll predict power consumption and scale workloads down to optimize energy efficiency. This is crucial for sustainable edge computing, especially in battery-powered or cost-sensitive deployments. Unlike Demo 1 which migrated the application, this demo will scale down replicas to reduce power consumption while maintaining service availability. Starting with 1 replica. As load increases, power consumption will rise. Our target is to keep the node below 200W to optimize energy efficiency.

4.2.6.1. Initialization

All the requisite applications (Tyris apps and Stress-NG) are instantiated as shown in *Figure 42* with a zoom in into the Stress-NG app showing 1 replica with 4 workers and 75% CPU load.

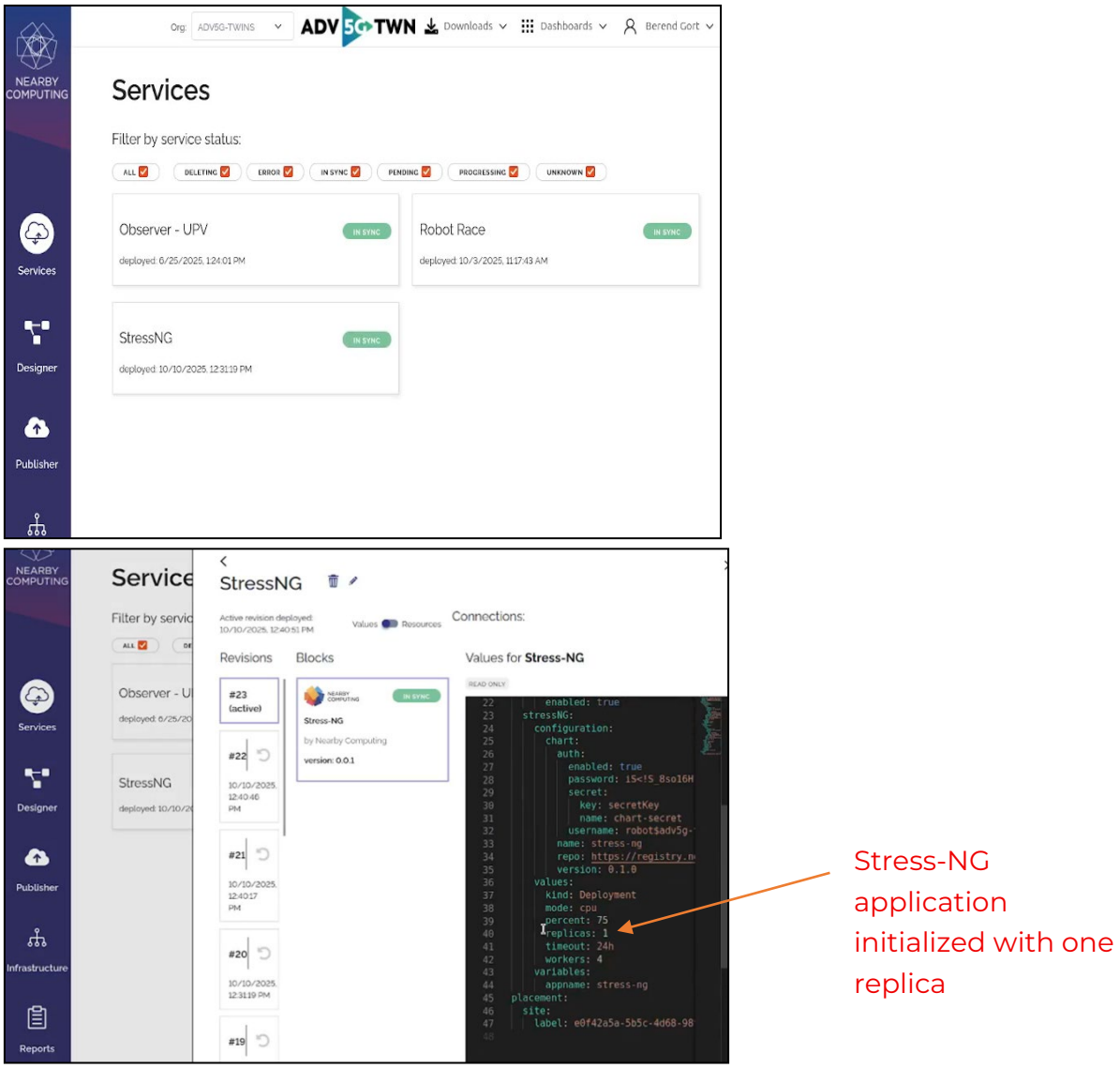


Figure 42:Instantiated apps (top) and configuration file for Stress-NG app (bottom)

4.2.6.2. Continuous monitoring

The current system state is continuously monitored. As seen from the Grafana dashboard of the Energy Demo in *Figure 43*, the energy consumption is shown (top) along with the node CPU utilization(middle) and the number of replicas of the stress-NG application (bottom). The

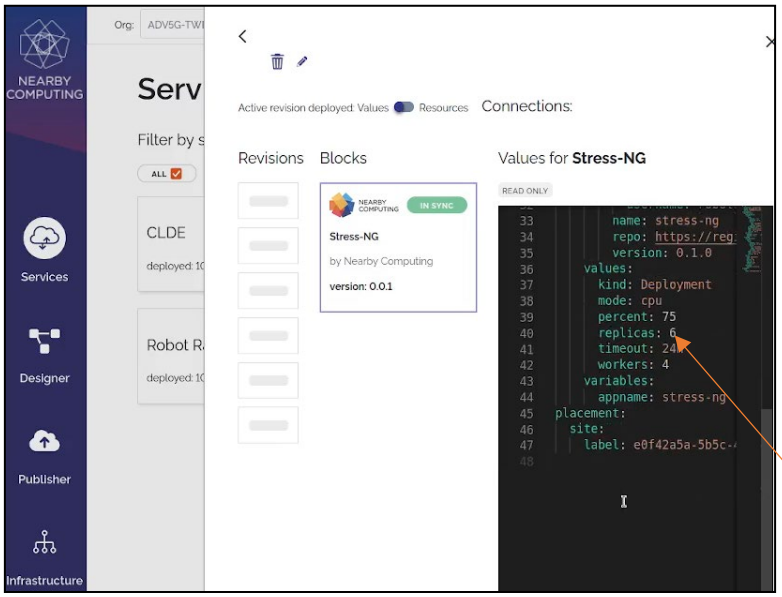
Figure 43: Grafana dashboard monitor for the node CPU utilisation, Energy consumption and number of application replicas

The Grafana dashboard illustrates a complete energy-aware scaling cycle over time. Initially, the system operates steadily with low CPU utilization (~0.15%), one active replica, and stable energy consumption around 270 W. Around 12:30, workload increases, prompting the Decision Engine to scale up replicas from 1 to about 8, which causes CPU utilization and total power consumption to rise sharply. The energy peaks near 310W as the system handles higher load. Once the predicted power usage exceeds the defined 350 W threshold, a scale-down event is triggered. Consequently, the number of replicas is halved, CPU usage falls, and total energy consumption returns to its baseline level (~270 W) by around 12:45. Overall, the graph demonstrates how the system dynamically manages workloads to maintain power within safe operating limits while responding to changing demand.

4.2.6.3. Predictive scaling

Figure 44: Predictive system

The informer model predicts future power violating budget threshold



The replicas reduced to 6 at that stage

Figure 45: Downscaling of the Stress-NG app replicas to 4

Figure 46: shows an extended view of the cycle where the peaks for node CPU utilization, number of replicas and energy consumption are indicated. From the measurements depicted, the 3 metrics peak at the same time indicating that the CPU utilization is correlated with the energy consumption. Then after the scaling operation, the power consumption is seen to decrease and normalize within the acceptable levels.

Figure 46:Power consumption normalisation

5. Mapping of the contributions to project objectives

This section maps the contributions of the highlighted contributions to the seven key objectives of the project.

Objective: Collaborate in the definition of use cases, KPIs and requirements for real-time digital twin applications in port environments, from the point of view of orchestration of computing resources in the IoT-edge-cloud platform.

The results related to this Objective are reported in deliverables related to architecture and planning activities. These deliverables analyze requirements, KPIs and use cases for the IoT-edge-cloud platform and the digital twin pilots. In Activity A8, deliverables A8-D2.1, A8-D2.2 and A8-D2.3 introduce the state-of-the-art in orchestration and AI/ML algorithms, discuss the requirements for the NearbyOne orchestrator, define the observability and ML frameworks, and describe the datasets that can be leveraged by the proposed AI/ML-based orchestration, all of which depend on clear KPIs and service requirements. A8-D4.1 complements this by presenting the work plan for the laboratory and pilots, including the design, implementation and test plan, where the concrete use cases and associated KPIs for the digital twin scenarios are defined. In Activity A7, A7-D2.1, A7-D2.2 and A7-D2.3 play a similar

role from an energy-efficiency perspective, specifying the energy-related requirements and KPIs for constrained edge nodes, while A7-D4.1 details the planning of the real-time digital twin laboratory and pilots, including the metrics and test criteria. The software deliverables A8-S2.1, A8-S2.2, A7-S2.1 and A7-S2.2, which implement orchestration and AI/ML algorithms, also implicitly depend on and therefore contribute to the formalization of these use cases, KPIs and requirements.

Objective: Design the architecture for an IoT-edge-cloud orchestration platform for real-time digital twins in port environments.

Activities undertaken relating to the design of the orchestration architecture are primarily documented in the D2.X series of the deliverables for both activities A7 and A8. In Activity A8, A8-D2.1 introduces the IoT-edge-cloud and AI/ML solution architecture for digital twins, A8-D2.2 refines this into an evolved architecture that includes the zero-touch closed-loop orchestration framework, the observability stack and the ML framework, and A8-D2.3 provides the final architecture of the IoT-edge-cloud platform together with the AI/ML solutions and software described in A8-S2.2. In Activity A7, A7-D2.1, A7-D2.2 and A7-D2.3 mirror this work with a specific focus on energy efficiency, describing how the same architectural concepts are adapted to energy-aware orchestration in Telco and IoT-edge-cloud environments. The deployment-oriented deliverables A8-D4.2 and A7-D4.2 further contribute to this objective by detailing the NearbyOne orchestration solution architecture, the provisioning of the ADV5G-TWINS edge infrastructure and the procedures for onboarding third-party applications and services. Together, these deliverables define the complete multi-cloud, multi-edge orchestration architecture required to support real-time digital twins in port environments.

Objective: Deploy an IoT-edge-cloud orchestration platform together with IoT ADV5G-TWINS-INFRA-PORT (hardware and edge computing nodes) and IoT ADV5G-TWINS-EDGE (5G open RAN software radio solution) in the port environment provided by ADV5G-TWINS-LOGISTICS, the iTEAM-UPV laboratories and the cloud with a cloud computing node.

This objective is addressed by the results documented in the deliverables related to planning, deployment, prototype and system validation that realize the architecture on real infrastructure. In Activity A8, A8-D4.1 reports the planning of the 5G IoT-edge-cloud laboratory and pilots, while A8-D4.2 details the actual implementation and start-up of the 5G IoT-edge-cloud platforms in the iTEAM-UPV laboratory and in the pilots, integrating, testing and validating the solutions and documenting the A8-P4.1 prototype and A8-VT4.1 validation pilot. A8-D4.3 then reports the setup and results of the final real-time digital twin pilots, and the sequence of software/demo/prototype deliverables A8-

P4.1, A8-P4.2, A8-VT4.1 and A8-VT4.2 tracks the creation, update and execution of the laboratory and pilot platforms.

In Activity A7, A7-D4.1 provides the work plan for the energy-efficiency oriented laboratory and pilots, A7-D4.2 details the implementation of the platforms (again documenting A7-P4.1 and A7-VT4.1) including provisioning of the ADV5G-TWINS edge infrastructure and application onboarding, and A7-D4.3 documents the final tests and results; similarly, A7-P4.1, A7-P4.2, A7-VT4.1 and A7-VT4.2 cover the creation, update and pilot execution for A7. Overall, these deliverables show how the IoT-edge-cloud orchestration platform is deployed over ADV5G-TWINS-INFRA-PORT, ADV5G-TWINS-EDGE and ADV5G-TWINS-LOGISTICS assets in the lab, the port environment and the cloud.

Objective: Design, validate and evaluate dynamic and distributed orchestration mechanisms for computing resources and applications in IoT-edge-cloud environments.

The results related to this objective are reported in a combination of architectural, software and pilot deliverables. A8-D2.2 and A8-D2.3 define the closed-loop orchestration framework and the architectural components (NearbyOne, observability stack, ML framework) that enable automated and distributed service management across edge and cloud. These mechanisms are implemented in the orchestration algorithms delivered in A8-S2.1 and A8-S2.2 and are evaluated in the pilots documented in A8-D4.2 and A8-D4.3, as well as in the prototype and validation phases A8-P4.1, A8-P4.2, A8-VT4.1 and A8-VT4.2. In Activity A7, A7-D2.2 and A7-D2.3 provide the corresponding architecture and final design of dynamic orchestration mechanisms with an energy-efficiency focus, A7-S2.1 and A7-S2.2 implement these mechanisms in software, and A7-D4.2, A7-D4.3 together with A7-P4.1, A7-P4.2, A7-VT4.1 and A7-VT4.2 validate and evaluate them in laboratory and pilot scenarios. Across both activities, these deliverables demonstrate how the platform can dynamically and distributively orchestrate computing resources and applications in realistic IoT-edge-cloud environments.

Objective: Design, validate and evaluate AI/ML-based orchestration algorithms that manage in real time the computing resources and applications on the IoT-edge-cloud platform, in both centralized and distributed ways, including self-configuration mechanisms that can also be applied to the 5G Core (5GC).

The project results related to this objective are reported on deliverables focusing on AI/ML-based orchestration. In Activity A8, A8-D2.1 introduces the state of the art in machine-learning-based orchestration algorithms for the IoT-edge-cloud continuum and captures the requirements for NearbyOne to

manage the lifecycle of digital twin services and edge infrastructure; A8-D2.2 presents the AI/ML-driven orchestration framework, including the observability and ML frameworks needed for training and deploying these algorithms; and A8-D2.3 documents the final design of the AI/ML algorithms, the datasets they use and how they can be leveraged to improve orchestration performance. The software deliverable A8-S2.2 then implements these AI/ML-based orchestration algorithms. In Activity A7, A7-D2.1, A7-D2.2 and A7-D2.3 focus on AI/ML algorithms tailored to energy-efficient orchestration and constrained edge nodes, describing both the models and the orchestration actions they trigger, and A7-S2.2 implements the corresponding AI/ML algorithms. These algorithms are ultimately validated in the deployment and pilot deliverables A8-D4.2, A8-D4.3, A7-D4.2 and A7-D4.3, which demonstrate their real-time operation and self-configuration capabilities in 5G IoT-edge-cloud platforms.

Objective: Provide the IoT-edge-cloud platform with KPI analysis and measurement capabilities for both the computing infrastructure and the applications, and generate resulting reports based on the tests.

The KPI analysis, measurement and reporting capabilities are mainly realised through the observability framework and the pilot/test deliverables. A8-D2.2 defines the observability stack that performs real-time system metrics scraping, data storage and visualization across the distributed edge-to-cloud infrastructure, while A8-D2.3 details the datasets that can be extracted from this stack and used by AI/ML algorithms. A8-D4.2 and A8-D4.3 then make extensive use of these capabilities to document the integration, testing and final results of the digital twin laboratory and pilots, effectively generating the KPI-based reports required by this objective. In Activity A7, A7-D2.2 and A7-D2.3 provide the same observability and dataset perspective with emphasis on energy efficiency, explaining how energy-related metrics are captured and used, and A7-D4.2 and A7-D4.3 report on the implementation and final tests of the energy-focused laboratory and pilots, including KPI-based evaluation of performance and energy efficiency.

Objective: Identify and quantify traffic bottlenecks for the real-time execution of AI/ML algorithms on IoT-edge-cloud platforms.

This objective is addressed through AI/ML related deliverables. A8-D2.2 and A8-D2.3 describe the closed-loop orchestration framework and the final AI/ML algorithms, including how datasets derived from the observability stack can be used to understand load, resource usage and system behavior; A8-S2.2 implements these AI/ML-based orchestration algorithms so that they must operate under realistic resource and traffic conditions. The pilots and validation phases documented in A8-D4.2, A8-D4.3, A8-VT4.1 and A8-VT4.2

provide concrete experimental setups where CPU, latency and other KPIs are monitored, allowing traffic and resource bottlenecks that affect real-time AI/ML execution to be identified and quantified. In Activity A7, A7-D2.2 and A7-D2.3 extend this analysis to energy-efficiency scenarios, with A7-S2.2 implementing energy-aware AI/ML orchestration algorithms. In these tests, the interaction between workload patterns, energy constraints and orchestration actions reveal the bottlenecks that arise when executing AI/ML workloads in real time on edge-cloud platforms. Moreover, the resource requirements and computational complexity of the proposed AI/ML algorithms are exhaustively evaluated against the state-of-art algorithms in the project results reported in [1] and [2].

6. Conclusion

The ADV5G-TWINS-ORCHESTRATION project focuses on the design, development, and deployment of advanced 5G IoT-edge-cloud platforms and intelligent orchestration frameworks aimed at enabling efficient management and operation of digital twin applications. These applications exploit the full potential of cutting-edge 5G and emerging 6G technologies to transform industrial and logistics sectors, thus driving smarter, more connected, energy-efficient, and highly automated operations.

To achieve these objectives, the project implemented a range of coordinated activities, including the design and development of AI/ML-driven algorithms for dynamic orchestration and energy performance optimization. These algorithms have been seamlessly integrated into the project's demonstration and validation activities, ensuring practical applicability and measurable impact.

This deliverable presents and analyzes the key outcomes of these two core activities. The results demonstrate a substantial advancement in autonomous resource management, achieved through the integration of intelligent orchestration mechanisms across distributed and heterogeneous infrastructures. Collectively, these innovations contribute to improved orchestration efficiency, enhanced system resilience, and optimized energy management, marking an important step forward toward next-generation autonomous 5G/6G-enabled ecosystems.

7. References

[1] Gort, Berend JD, et al. "Forecasting Trends in Cloud-Edge Computing: Unleashing the Power of Attention Mechanisms." *IEEE Communications Magazine* (2024).

[2] B. J. D. Gort, G. M. Kibalya and A. Antonopoulos, "AERO: Adaptive Edge-Cloud Orchestration With a Sub-1K-Parameter Forecasting Model," in *IEEE Transactions on Machine Learning in Communications and Networking*, vol. 3, pp. 463-478, 2025

[3] A. H. Gupta, "Approaches and applications of early classification of time series: A review," *IEEE Trans. Artif. Intell.*, pp. 47-61, 2020.

[4] P. Frazier, "A tutorial on Bayesian optimization," *preprint arXiv:1807.02811*, 2018.

[5] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *J. Syst. Eng. Electron.*, pp. 162-169, 2017.

[6] J. N. Van Rijn, "Hyperparameter importance across datasets," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 2367-2376, 2018.

[7] K. Yi, Q. Zhang, W. Fan, H. He, L. Hu, P. Wang, N. An, L. Cao, and Z. Niu, "FourierGNN: Rethinking multivariate time series forecasting from a pure graph perspective," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] L. Donghao and W. Xue, "ModernTCN: A Modern Pure Convolution Structure for General Time Series Analysis," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=vpJMJerXHU

[9] T. Meuser, L. Lovén, M. Bhuyan, S. G. Patil, S. Dustdar, A. Aral, S. Bayhan, C. Becker, E. De Lara, A. Y. Ding et al., "Revisiting edge ai: Opportunities and challenges," *IEEE Internet Computing*, vol. 28, no. 4, pp. 49–59, 2024.

[10] S. Arbat, V. K. Jayakumar, J. Lee, W. Wang, and I. K. Kim, "Wasserstein adversarial transformer for cloud workload prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 12 433–12 439.

[11] P. Chen, Y. ZHANG, Y. Cheng, Y. Shu, Y. Wang, Q. Wen, B. Yang, and C. Guo, "Pathformer: Multi-scale Transformers with Adaptive Pathways for Time Series Forecasting," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=lJkOCMP2aW

[12] X. Kong, Y. Wu, H. Wang, and F. Xia, "Edge computing for internet of everything: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 472–23 485, 2022.

[13] V. Ekambaram, A. Jati, N. Nguyen, P. Sinthong, and J. Kalagnanam, "Tsmixer: Lightweight mlp-mixer model for multivariate time series forecasting," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 459–469.

[14] S. Lin, W. Lin, W. Wu, H. Chen, and J. Yang, "SparseTSF: Modeling Long-term Time Series Forecasting with 1k Parameters," in *International Conference on Machine Learning*, 2024.

[15] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container orchestration using co-simulation and gradient based optimization for fog computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, 2021.

[16] D. Zhang, W. Ding, H. Wang, and B. Zhang, "Autocorrelation convolution networks based on deep learning for automatic modulation classification," in 2020 15th IEEE conference on industrial electronics and applications (ICIEA). IEEE, 2020, pp. 1561–1565.

[17] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "The Power of Prediction: Microservice Auto Scaling via Workload Learning," in Proceedings of the ACM Symposium on Cloud Computing, 2022.

[18] X. Tang, "Large-Scale Computing Systems Workload Prediction Using Parallel Improved LSTM Neural Network," *IEEE Access*, vol. 7, pp. 40 525–40 533, 2019

[19] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in Proceedings of the AAAI conference on artificial intelligence, vol. 37, no. 9, 2023, pp. 11 121–11 128.

[20] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "BoTorch: A framework for efficient Monte-Carlo Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21,524–21,538, 2020.

[21] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration," *Advances in Neural Information Processing Systems*, vol. 31,2018

[22] A. Chen et al., "Developments in MLflow: A system to accelerate the machine learning lifecycle," in Proc. 4th Int. Workshop Data Manag. End-to-End Mach. Learn., ser. Deem '20, New York, NY, USA: Assoc. Comput. Mach., 2020. [Online]. Available: https://doi.org/10.1145/3399579.3399867