

D2.3: Core and Edge enablers delivery result

Revision: v.1.0

Work package	WP 2
Task	Task 2.5
Due date	31/05/2025
Submission date	29/05/2025
Deliverable lead	ERI
Version	1.0
Authors	Diego San Cristobal, Rocio Dominguez (ERI), Antti Pauanne, Jani Pellikka, Kenichi Komatsu, Hamid Ahmed, Mahdi Salmani, Afeez Afuwape (UOULU), Aurora Ramos, Javier Godas, Enrique Lluesma (CGE), Javier Fernández (i2CAT), Fernando Pargas (TID)
Reviewers	Jarno Pinola (VTT), Miguel Glassée (IMEC), Mohammed Al-Rawi, Jonathan Rodriguez (IT)
Abstract	This document reports the work done by the WP2 partners on developing, deploying and validating the integration of the Core and Edge enablers of the project
Keywords	XR, Extended Reality, Core, Edge, enablers, orchestration, APIs, slicing, optimization, session management

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	08/04/2024	1st version of the template	Diego San-Cristobal (Ericsson)
V0.2	25/04/2025	Version ready for partners review	All authors
V0.3	05/05/2025	Version after partners review to be addressed by the authors	Jarno Pinola (VTT), Miguel Glassée (IMEC)
V0.4	15/05/2025	Version ready for TM review	All authors
V0.5	21/05/2025	Version after TM review to be addressed by the authors	Mohammed Al-Rawi (IT)
V1.0	29/05/2025	Version ready for submission	All authors

DISCLAIMER



Co-funded by
the European Union



Project funded by



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,
Education and Research EAER
State Secretariat for Education,
Research and Innovation SERI

6G-XR (6G eXperimental Research infrastructure to enable next-generation XR services) project has received funding from the [Smart Networks and Services Joint Undertaking \(SNS JU\)](#) under the European Union's [Horizon Europe research and innovation programme](#) under Grant Agreement No 101096838. This work has received funding from the [Swiss State Secretariat for Education, Research, and Innovation \(SERI\)](#).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

© 2023 - 2025 6G-XR Consortium

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)	✓
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	EU RESTRICTED under the Commission Decision No2015/ 444	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision No2015/ 444	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision No2015/ 444	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

This document reports the work done by the partners to deploy and develop in the South and North Nodes the Core and Edge enablers identified and designed in previous stages of the project.

Partners have followed a continuous development, deployment and integration approach to produce and validate the enablers. This means that each phase is an iterative process where small bits are implemented, tested and corrected if needed, before adding new pieces to the whole. The plan was to fully develop the enablers between March 2024 and October 2024. The deployment of the early releases of the enablers was started in July 2024, allowing time to deploy, test and debug the code, and finally accept the enablers by May 2025.

The Madrid Edge Orchestrator has been further enhanced to provide the framework to hold northbound interface (NBI) APIs, to support Graphic Processing Unit (GPU) based containerized applications, to be compatible with Helm packages for orchestrating applications, to support NVIDIA GPUs, and to allow persistent volume support for Helm. Two servers were deployed to form the Barcelona Edge platform, with support for GPU-intensive workloads. Both Edges are federated, providing features of federation management, availability zone synchronization, artefact management, application onboarding management and application deployment management.

A mechanism to adapt the QoS for data sessions has been developed. The QoS can be adapted by interacting with the Quality on Demand (QoD) CAMARA API, which then calls the QoS Session Network Exposure Function (NEF) API for the modification in the 5G network. The Simple Edge discovery tool makes use of the User Equipment (UE) Location NEF API for identifying the most suitable Edge. Additionally, 5G network performance and utilization metrics can be requested using the data collection API.

The IMS Data Channel (IMSDC) solution is deployed in the South Node to allow enhanced extended reality (XR) services when dialling from the phone. Work was done to optimize connectivity and data transmission performance. The IMSDC has been integrated with the 5G Core at 5Tonic lab.

The North Node Adapter can obtain metrics per slice from Qosium probes, can create and delete network slices through the CumuCore REST API, interacts with the AI-based network resource optimization tool, instantiates virtual machines via Open Source MANO and calls the open vSwitch to apply policy rates.

It has been validated that the clients subscribed to all relevant energy management topics successfully received live data from energy monitoring equipment. The forecasting APIs are correctly called, parsed, formatted, stored and visualized.

The deliverable ends with a section to collect the evidence of the integration validation tests, certifying that the enablers work as expected and are therefore ready for the project use cases validation.



TABLE OF CONTENTS

- Disclaimer 3
- Copyright notice 3
- 1 INTRODUCTION..... 12**
- 1.1 Objective of the deliverable..... 12
- 1.2 Structure of the deliverable 12
- 1.3 Target Audience of the deliverable 12
- 2 INITIAL ENABLERS DEVELOPMENT AND INTEGRATION PLAN 13**
- 2.1 GENERAL APPROACH 13
- 2.2 INITIAL TIME PLAN 14
- 3 ENABLERS DEVELOPMENT PHASE 16**
- 3.1 SOUTH NODE USER PLANE ENABLERS 16
- 3.1.1 *E1.1 IEAP Edge orchestrator* development..... 16
- 3.1.2 *E2.1 Barcelona Edge orchestrator* development..... 19
- 3.1.3 *E4.1 QoS API* development 21
- 3.1.4 *E4.2 Simple Edge Discovery API* development..... 23
- 3.1.5 *E4.3 Traffic Influence API* development 24
- 3.1.6 *E5.1 Edge Federation* development 24
- 3.1.7 *E6.1 Service Parameter API* development..... 26
- 3.1.8 *E6.2 UE Location API* development 27
- 3.1.9 *E6.3 QoS Session API* development 28
- 3.1.10 *E6.4 Data Collection API* development..... 29
- 3.2 SOUTH NODE CONTROL PLANE ENABLERS..... 30
- 3.2.1 *E9.1 IMS Data Channel Server* development..... 30
- 3.3 NORTH NODE 3D DIGITAL TWIN ENABLERS 31
- 3.3.1 *E3.1 North Node adapter* development 31
- 3.3.2 *E3.2 3D Digital Twin* development 34
- 3.3.3 *E3.4 Resource Optimization* development 35
- 3.3.4 *E7.1 Cumucore Slice Management API* development 36
- 3.4 NORTH NODE ENERGY FRAMEWORK ENABLERS..... 36
- 3.4.1 *E3.3 Energy Management* development..... 37
- 3.4.2 *E8.1 OAIBOX* development 38
- 4 ENABLERS DEPLOYMENT PHASE 39**
- 4.1 SOUTH NODE USER PLANE ENABLERS 39

4.1.1	<i>E1.1 IEAP Edge orchestrator & APIs deployment</i>	39
4.1.2	<i>E2.1 Barcelona Edge orchestrator deployment</i>	40
4.1.3	<i>E5.1 Edge Federation deployment</i>	43
4.1.4	Network Exposure Function (NEF) APIs deployment	47
4.2	SOUTH NODE CONTROL PLANE ENABLERS.....	48
4.2.1	<i>E9.1 IMS Data Channel Server deployment</i>	48
4.3	NORTH NODE 3D DIGITAL TWIN ENABLERS	49
4.3.1	<i>E3.1 North Node adapter deployment</i>	49
4.3.2	<i>E3.2 3D Digital Twin deployment</i>	51
4.3.3	<i>E3.4 Resource Optimization Deployment</i>	53
4.3.4	<i>E7.1 Cumucore Slice Management API deployment</i>	54
4.4	NORTH NODE ENERGY FRAMEWORK ENABLERS.....	55
4.4.1	<i>E3.3 Energy Management deployment</i>	55
4.4.2	<i>E8.1 OAIBOX deployment</i>	56
5	ENABLERS INTEGRATION VALIDATION PHASE	58
5.1	SOUTH NODE USER PLANE ENABLERS	58
5.1.1	Edge Federation validation.....	58
5.1.2	QoS change validation	64
5.1.3	Finding closest Edge validation	65
5.1.4	Changing UPF validation.....	66
5.1.5	Collecting metrics validation	68
5.2	SOUTH NODE CONTROL PLANE ENABLERS.....	69
5.2.1	IMS VMs connectivity validation	69
5.3	NORTH NODE 3D DIGITAL TWIN ENABLERS	71
5.3.1	NNA – Qosium Integration Test.....	71
5.3.2	NNA – Cumucore Integration Test.....	72
5.3.3	NNA – AI/ML Integration Test	74
5.3.4	NNA – OSM Integration Test	74
5.3.5	NNA – OVS Integration Test	75
5.3.6	North Node Web Portal – Qosium Measurement Integration Test	76
5.4	NORTH NODE ENERGY FRAMEWORK ENABLERS.....	77
5.4.1	Data exchange between VTT and UOULU gNB sites using MQTT bridge broker integration	77
5.4.2	North Node UOULU forecasting APIs integration	79
6	CONCLUSIONS	83
7	REFERENCES	84

LIST OF FIGURES

FIGURE 1: CONTINUOUS DEVELOPMENT, DEPLOYMENT AND INTEGRATION IN 6G-XR WP2	13
FIGURE 2: PLAN FOR CGE'S ENABLERS	14
FIGURE 3: PLAN FOR I2CAT'S ENABLERS.....	14
FIGURE 4: PLAN FOR ERI'S ENABLERS.....	15
FIGURE 5: PLAN FOR TID'S ENABLERS	15
FIGURE 6: PLAN FOR UOULU'S ENABLERS	15
FIGURE 7: EDGE IN BARCELONA, HOSTED BY I2CAT	21
FIGURE 8: QOD SESSION CREATION WORKFLOW	23
FIGURE 9: SIMPLE EDGE DISCOVERY SESSION CREATION WORKFLOW	24
FIGURE 10: EWBI API AND THE TWO EDGES IN THE SOUTH NODE.....	25
FIGURE 11: API WORKFLOW FOR SLICES LIST RETRIEVAL USING UDR.....	26
FIGURE 12: API WORKFLOW FOR SLICE ASSIGNMENT USING UDR	27
FIGURE 13: API WORKFLOW FOR QOS CHANGE	28
FIGURE 14: CLASS DIAGRAM OF THE NNA APPLICATION	33
FIGURE 15: FUNCTIONAL DIAGRAM OF 3D DIGITAL TWIN AND THE EDGE SERVICES	34
FIGURE 16: UPDATED NORTH NODE HIGH LEVEL ARCHITECTURE	37
FIGURE 17: IEAP DEPLOYMENT DIAGRAM	40
FIGURE 18: IEAP HARDWARE AND VIRTUALIZATION DIAGRAM	40
FIGURE 19: SERVERS HOSTING THE 6GXR MAIN CLUSTER.....	41
FIGURE 20: TOWER PC HOSTING THE INFRASTRUCTURE USED FOR THE OPEN CALLS.....	41
FIGURE 21: OPENSTACK HOSTING THE EDGE INFRASTRUCTURE IN BARCELONA.....	42
FIGURE 22: COMPUTING RESOURCES	42
FIGURE 23: OVERVIEW OF THE SERVICES RUNNING	43
FIGURE 24: CURRENT WORKLOAD.....	43
FIGURE 25: PRIVATE REPOSITORY FOR THE MEF MANAGER	44
FIGURE 26: CLUSTER HOSTING THE MEF MANAGER	45
FIGURE 27: FEDERATION MANAGEMENT METHODS	45
FIGURE 28: AVAILABILITY ZONE INFO SYNCH	46
FIGURE 29: ARTEFACT MANAGEMENT.....	46
FIGURE 30: APPLICATION ONBOARDING.....	47
FIGURE 31: APPLICATION DEPLOYMENT MANAGEMENT	47
FIGURE 32: NEF SETUP IN THE SOUTH NODE.....	48
FIGURE 33: IMS DC DEPLOYMENT IN THE SOUTH NODE	49
FIGURE 34: NNA DEPLOYMENT WITH RELATED NETWORK COMPONENTS	50

FIGURE 35: 3D DIGITAL TWIN XR FAB LAB APP DEPLOYMENT IN THE LOCAL 5G EDGE SYSTEM	51
FIGURE 36: 5G MODEM CONNECTION WITH VR GLASSES	52
FIGURE 37: XR FAB LAB OF BABYLON.JS AND OPENVIDU CONNECTIVITY IMPLEMENTATION	52
FIGURE 38: XR FAB LAB APP PROCEDURE FOR QOSIUM MEASUREMENT VALIDATION	53
FIGURE 39: POLICY NETWORK TRAINING AND ALLOCATION TREND	54
FIGURE 40: THE ROLE OF NNA AND CUMUCORE IN SLICE MANAGEMENT	55
FIGURE 41: NORTH NODE ENERGY MEASUREMENT FRAMEWORK	56
FIGURE 42: OAIBOX POWER CONSUMPTION ENABLED BY NORTH NODE ENERGY MEASUREMENT FRAMEWORK.....	57
FIGURE 43: PLANNED FEDERATION FROM IEAP GUI	59
FIGURE 44: ACCEPT FEDERATION FROM IEAP GUI	59
FIGURE 45: PROXY LOGS FOR FEDERATION ACCEPTANCE	59
FIGURE 46: ACCEPTED FEDERATION FROM IEAP GUI	59
FIGURE 47: ZONE ACCEPTANCE FROM IEAP GUI	60
FIGURE 48: ZONE ACCEPTANCE WORKFLOW	60
FIGURE 49: PROXY LOGS FOR ZONE ACCEPTANCE	60
FIGURE 50: ZONE ACCEPTED FROM IEAP GUI	60
FIGURE 51: ARTEFACT UPLOADING FROM IEAP GUI	61
FIGURE 52: APPLICATION ONBOARDING FROM IEAP GUI	61
FIGURE 53: APPLICATION ONBOARDING WORKFLOW	62
FIGURE 54: PROXY LOGS FOR APPLICATION ONBOARDING	62
FIGURE 55: APPLICATION ONBOARDING RESULT FROM IEAP GUI	62
FIGURE 56: APPLICATION INSTANCE DEPLOYMENT FROM IEAP GUI.....	63
FIGURE 57: PROXY LOGS FOR APPLICATION INSTANCE DEPLOYMENT	63
FIGURE 58: APPLICATION INSTANCE DEPLOYMENT RESULT FROM IEAP GUI	63
FIGURE 59: APPLICATION INSTANCE DEPLOYMENT RESULT FROM I2CAT-BARCELONA	63
FIGURE 60: REQUEST AND RESPONSE FOR QOD CREATE SESSION (I).....	64
FIGURE 61: PROXY LOGS FOR ASSESSIONWITHQOS CREATE SUBSCRIPTION (I).....	65
FIGURE 62: SLICE PROFILE INFORMATION RETRIEVING (I).....	65
FIGURE 63: REQUEST AND RESPONSE FOR SIMPLEEDGEDISCOVERY (BARCELONA).....	65
FIGURE 64: REQUEST AND RESPONSE FOR SIMPLEEDGEDISCOVERY (MADRID)	66
FIGURE 65: PROXY LOGS FOR MONITORINGEVENT (BARCELONA).....	66
FIGURE 66: PROXY LOGS FOR MONITORINGEVENT (MADRID)	66
FIGURE 67: REQUEST AND RESPONSE FOR QOD CREATE SESSION (II).....	67
FIGURE 68: PROXY LOGS FOR ASSESSIONWITHQOS CREATE SUBSCRIPTION (II).....	67
FIGURE 69: SLICE PROFILE INFORMATION RETRIEVING (II).....	67
FIGURE 70: PING COMMAND FOR UE CONNECTED TO MADRID UPF	68



FIGURE 71: PING COMMAND FOR UE CONNECTED TO BARCELONA UPF68

FIGURE 72: DATA COLLECTION API EXAMPLE69

FIGURE 73: EXAMPLE OF IMS VMS CONNECTIVITY VALIDATION70

FIGURE 74: TEST LOGS OF THE INITIATION OF PER-SLICE MEASUREMENTS IN QOSIUM71

FIGURE 75: TEST LOGS OF FETCHING PER-SLICE DL/UL MEASUREMENT KPIS FROM QOSIUM.....72

FIGURE 76: TEST LOGS OF THE TERMINATION OF PER-SLICE MEASUREMENTS IN QOSIUM72

FIGURE 77: TEST LOGS OF CREATING AND DELETING SLICES IN CUMUCORE73

FIGURE 78: CREATED SLICES VISIBLE IN CUMUCORE GUI73

FIGURE 79: TEST LOGS OF THE AI/ML INTEGRATION TEST74

FIGURE 80: TEST LOGS OF THE OSM INTEGRATION TEST.....75

FIGURE 81: LOGS OF THE OVS INTEGRATION TEST.....76

FIGURE 82: QOSIUM MEASUREMENT RESULTS IN NORTH NODE PORTAL.....77

FIGURE 83: DATA EXCHANGE VALIDATION BETWEEN VTT AND UOULU79

FIGURE 84: SCREENSHOT OF GRAFANA VISUALIZATION OF INTEGRATED FORECASTING APIS80

FIGURE 85: FMI ENERGY WEATHER FORECAST 66H API VALIDATION81

FIGURE 86: ELSPOT ELECTRICITY PRICING 24 AHEAD API VALIDATION82

FIGURE 87: FINGRID CO₂ ESTIMATES API VALIDATION.....82



LIST OF TABLES

TABLE 1: IEAP RELEASE 2.1.0.0.0 FEATURES - 29/FEB/2024.....16

TABLE 2: IEAP RELEASE 2.2.1.0.0 FEATURES - 25/JUNE/2024.....17

TABLE 3: IEAP RELEASE 2.3.2.0.0 FEATURES - 11/DEC/2024.18

TABLE 4: 3GPP ASSESSIONSWITHQOS VERSION DISCREPANCY22

TABLE 5: QOS PROFILE AND SLICE PROFILES MAPPING22

TABLE 6: 3GPP MONITORINGEVENT VERSION DISCREPANCY23

TABLE 7: PYTHON PACKAGES USED BY THE NNA32

TABLE 8: IMS DATA CHANNEL SOLUTION FLOWS69

ABBREVIATIONS

5GTN	5G Test Network
AF	Application Function
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
DL	Downlink
DNN	Data Network Name
GPSI	Generic Public Subscription Identifier
GPU	Graphic Processing Unit
GUI	Graphical User Interface
ICE	Interactive Connectivity Establishment
IEAP	Intelligence Edge Application Platform
IMSDCS	IMS Data Channel Server
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MANO	NFV Management and Orchestration
MEC	Multi-access Edge Computing
ML	Machine Learning
MVP	Minimum Viable Product
NBI	Northbound Interface
NEF	Network Exposure Function
NFV	Network Functions Virtualization
NPN	Non-Public Network
OSM	Open Source MANO
OVS	Open vSwitch
PCF	Policy Control Function
QoD	Quality on Demand
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SBI	Southbound Interface
SMF	Session Management Function
SUPI	Subscription Permanent Identifier
TAI	Tracking Area Identity
TCP	Transmission Control Protocol
UC	Use Case
UDM	Unified Data Manager
UDR	Unified Data Repository
UE	User Equipment
UL	Uplink
UPF	User Plane Function
URSP	UE Route Selection Policy
VLAN	Virtual Local Area Network
VM	Virtual Machine
WSGI	Web Server Gateway Interface

1 INTRODUCTION

1.1 OBJECTIVE OF THE DELIVERABLE

The scope of this document is to report the results of the work performed in 6G-XR WP2 to develop, deploy and validate the integration of the enablers in the experimentation Nodes.

1.2 STRUCTURE OF THE DELIVERABLE

The deliverable is structured in four sections: (i) initial plan, (ii) development phase, (iii) deployment phase and (iv) integration validation phase.

The initial plan contains the proposals of each of the partners to develop and integrate the enablers described in previous D2.2 [1]. The development phase contains a report of the development work done. As the functional description and workflows of the enablers are already described in D2.2, the current deliverable provides some detail on when these features were developed, and the deviations incurred from the original plan. Therefore, workflows are only described in case there are modifications from what was detailed in D2.2. The deployment phase reports when and how the enablers were deployed and outlines the setup of the components specifically in the experimentation Nodes. The integration validation phase provides evidence that the enablers work as expected and they are ready to be used for the UCs validation in WP6.

1.3 TARGET AUDIENCE OF THE DELIVERABLE

This deliverable is intended for project consortium partners, academic and research institutions, EU Commission services, and other stakeholders with a technical background in wireless networks, in 5G and 6G technologies, network orchestration, and Extended reality (XR) applications, particularly in the context of system and resource optimization.

2 INITIAL ENABLERS DEVELOPMENT AND INTEGRATION PLAN

This chapter focuses on sharing the initial plans of all partners for developing their enablers and integrating them jointly in the project Nodes, to validate that the interaction among them works properly before the full 6G-XR use cases validation phase. These plans were provided by partners in March 2024 (M15).

2.1 GENERAL APPROACH

Partners in 6G-XR WP2 have followed continuous development, deployment and integration practices for the production and validation of the identified enablers. These practices emphasize constant testing of the performed work, to be able to evaluate early that the features behave as expected, resulting in a more agile software lifecycle management. With this idea in mind, within the scope of WP2, there have been three phases: (i) Enablers development, which is the phase when the software of the enablers is built; (ii) deployment of the enablers in the experimental Nodes; and (iii) integration validation, where the interaction among different enablers is tested to make sure they are ready for the use cases. Each of the phases are iterative processes that increase complexity.

Figure 1 below illustrates the way of working approach. In the enablers development phase, every enabler has been produced independently by the assigned partner. Each feature of the enabler has been built, tested and merged into a sequence of releases. This phase happened between M15 and M22, in time to report the details of the final enablers solution in D2.2 [1]. The deployment and integration validation phases were carried out together, between M19 and M29. As soon as stable releases of the enablers were produced, they were deployed at the Nodes, to confirm that the deployment worked well and to start testing the interaction with other enablers.

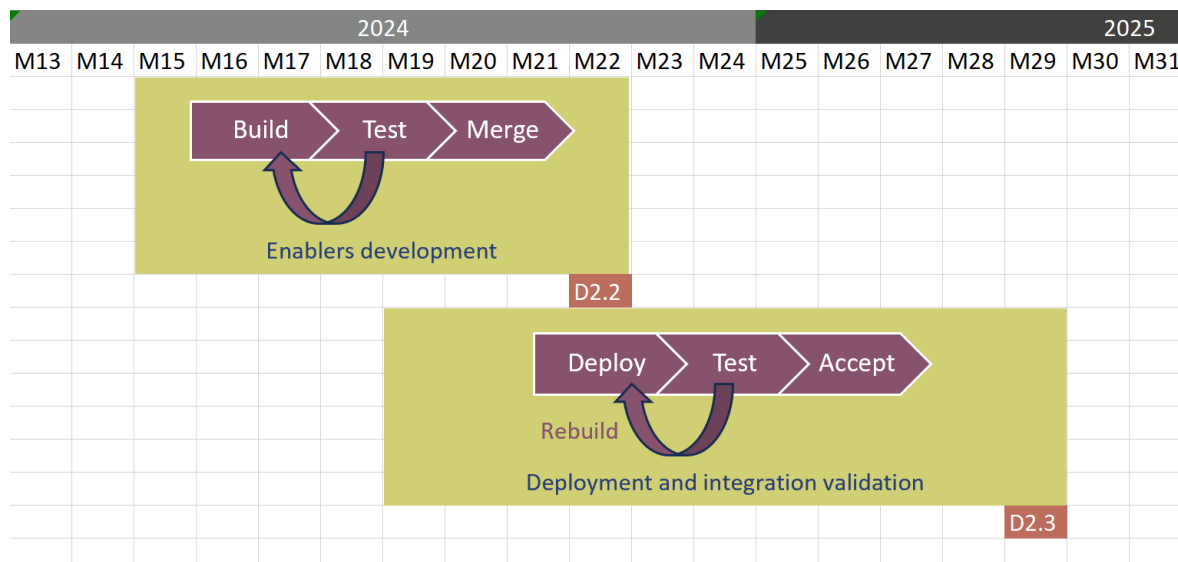


Figure 1: Continuous development, deployment and integration in 6G-XR WP2

2.2 INITIAL TIME PLAN

This chapter gathers the high-level plans from partners involved in WP2 for developing, deploying and integrating their enablers.

CGE shared the plan represented in Figure 2 below. In the scope of WP2, CGE developed the following enablers as identified in 6G-XR D2.2 [1]: *E1.1 Intelligent Edge Application Platform (IEAP)*, *E4.1 Quality on Demand (QoD) API*, *E4.2 Edge Discovery API*, *E4.3 Traffic Influence API* and *E5.1 East-Westbound Interface (EWBI)* on IEAP side. The development of *E4.3 Traffic Influence API* had to start later than the rest because it depended on some decisions on the related CAMARA API sub-project. The development of that one was scheduled between M21 and M24. The rest of the enablers from CGE were planned to be deployed between M15 and M21. The deployment of the enablers at 5Tonic in the South Node was targeted within the period M20-M23. These enablers would be ready for validation from M22 onwards.

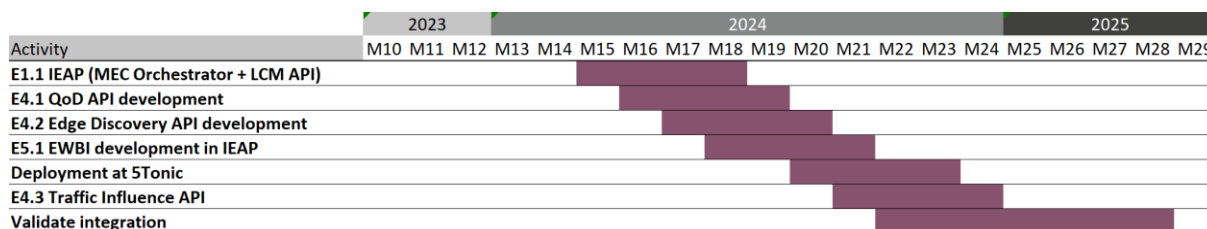


Figure 2: Plan for CGE's enablers

i2CAT aimed to follow the plan shown in Figure 3 below. In the scope of WP2, i2CAT developed the following enablers: *E2.1 MEF Manager* and *E5.1 EWBI* on i2EDGE side. The plan was to develop those enablers between M12 and M18. The integration with the other side of the South Node, the IEAP Edge at 5Tonic would start on M19.

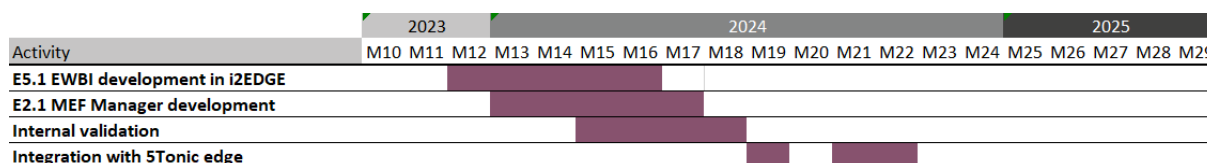


Figure 3: Plan for i2CAT's enablers

ERI developed Network Exposure Function (NEF) APIs named: *E6.1 Service Parameter API*, *E6.2 UE Location API*, *E6.3 QoS Session API* and *E6.4 Data Collection API*. The plans are depicted in Figure 4 below. The development of the APIs would be done in the period M16-M21. The APIs would be deployed between M20 and M23. From M21 there would be some APIs available for testing against other components.

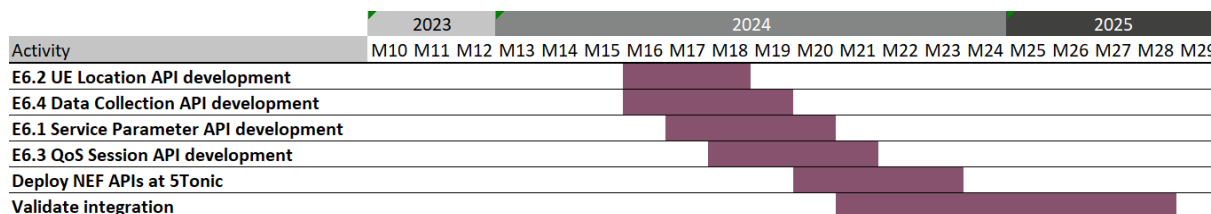


Figure 4: Plan for ERI's enablers

TID worked on the plan shown in Figure 5 below. The planned development of *E9.1 IMS Data Channel Server (IMSDCS) innovations* spanned from M10 to M19. That work progressed along with the deployment of an IMS Core and the IMS Data Channel solution at 5Tonic, which was done by ERI, and it was a pre-requisite to be able to deploy the final IMSDCS innovations over 5Tonic infrastructure in M20-M23. Finally, it would be ready for validation with other components developed in WP3 and WP6 from M24.

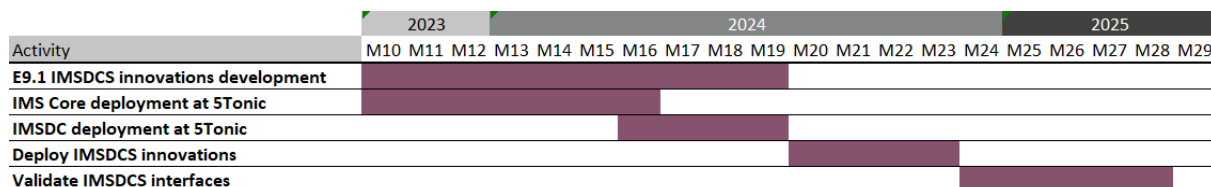


Figure 5: Plan for TID's enablers

UOULU followed the plan illustrated in Figure 6 below. As stated in 6G-XR D2.2 [1], UOULU worked on the enablers: *E3.1 North Node Adapter*, *E3.2 3D Digital Twin*, *E3.3 Energy Management*, *E3.4 Resource Optimization*, *E7.1 Cumucore Slice Creation* and *E8.1 OAIBOX*. The first developments to be started in M13 were *E3.2 3D Digital Twin* and *E3.3 Energy Management*. The last to be completed in M22 were expected to be *E3.1 North Node Adapter* and, again, *E3.3 Energy Management*. Whenever stable releases of the enablers were produced, they would be deployed at the North Node. This was scheduled between M15 and M23. Then all the enablers would be validated together in the period M21-M27.

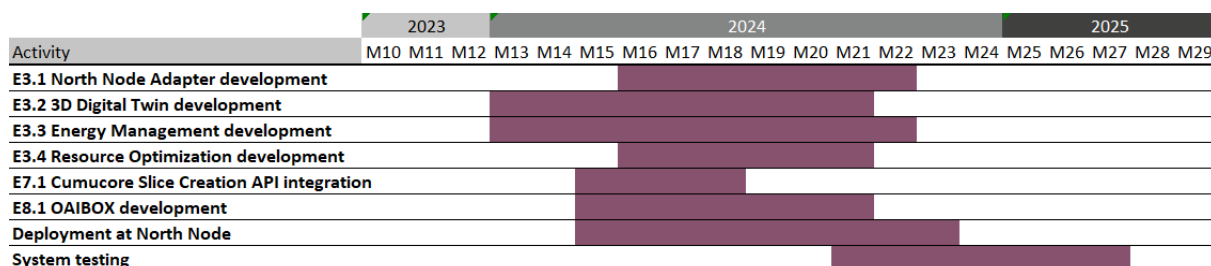


Figure 6: Plan for UOULU's enablers

3 ENABLERS DEVELOPMENT PHASE

This chapter reports the development work done by each partner independently to produce their enablers.

3.1 SOUTH NODE USER PLANE ENABLERS

This section covers the development report of the enablers needed in the South Node for UC1 *Resolution Adaptation or Quality on Demand* and UC2 *Routing to the Best Edge* (see D1.1 [2] for UCs description).

3.1.1 E1.1 IEAP Edge orchestrator development

The IEAP (Intelligence Edge Application Platform), is the Multi-access Edge Computing (MEC) [3] orchestrator (ETSI MEC compliant) used at 5Tonic within 6G-XR (see D2.2 [1] for details on its functionality).

The IEAP design took as conceptual starting point a previous simplified version of a Minimum Viable Product (MVP) MEC orchestrator by Capgemini called Ensconce, that was part of previous research projects such as FISHY [4].

Though taking the conceptual idea from FISHY project, the IEAP was completely developed from scratch overcoming the limitations found in that initial version. The main API for application lifecycle management was completely renewed and aligned with Open Gateway initiative by GSMA [5] and CAMARA Linux Foundation [6] project in order to enhance external exposure and simplify the accessibility by third party application providers. Besides, it can manage the deployment of applications in multiple kinds of compute infrastructures not only Virtual Machines but it is also compatible with Docker and importantly with Kubernetes (K8s).

The IEAP has been mainly developed in the period from October 2023 to June 2024, in two main cycles: from October 2023 to February 2024, and from March 2024 to June 2024. In January 2024 the first intensive test validation campaign was implemented. The second and final internal validation test campaign was performed during June 2024. After that, it was extended with fixes and new enablers (such as APIs on Edge Discovery and Traffic Influence) in December 2024 release.

The features provided in each of the releases are listed below.

Table 1: IEAP release 2.1.0.0.0 features - 29/Feb/2024.

Feature key	Description
F0001	Support for OOPG.02 NBI requirements that have been developed in IEAP as a reference set of REST APIs.
F0002	E/WBI Authentication (oAuth2.0) – Token based authentication of federated partners
F0003	Support for OPAG standard definition of Federation APIs
F0004	Support for edge/zone management

F0005	oAuth2.0 - NBI Authentication – Token based authentication of ISVs/Developers
F0006	Image vulnerability scanning
F0007	Support for LF CAMARA NBI APIs - QoD, Device Status, Location
F0008	Support for HELM packages for application orchestration
F0009	Support for orchestration of Virtual Machine(VM) type applications via KubeVirt on Kubernetes clusters
F0010	Persistent Volume (for Containers) via CEPH FS – Shared, Exclusive
F0011	Enterprise Management (Multi-Tenancy)
F0012	Support for GPU based Containerized applications
F0013	Monitoring, Troubleshooting, Alarm

Table 2: IEAP release 2.2.1.0.0 features - 25/June/2024.

Feature key	Description
F0014	Application LCM - OPAG NBI/EWBI Alignment
F0015	E/WBI Authentication (O-Auth)
F0016	Federation Planning/ Mgmt - OPAG E/WBI Alignment
F0017	Edge/ Zone Management
F0018	O-Auth - NBI Authentication
F0019	Image vulnerability scanning - OPAG NBI Alignment
F0020	CAMARA NBI APIs - QoD, Device Status, Location
F0021	K8s Application/Helm chart support,
F0022	VM application
F0023	Persistent Volume (for Containers)
F0024	OPAG E/WBI Alignment
F0025	Support for NVIDIA GPUs
F0026	Monitoring, Troubleshooting, Alarm
F0027	Container Application with multiple network – SRIOV/Multus
F0028	Support for Applications requesting huge pages
F0029	GSMA OPG.02 UNI interface – Application orchestration, discover and mobility for client devices

F0030	Support for Resource reservation via ISV Users
F0031	Docker application management (Docker Compose)
F0032	Support for Ingress, E-gress and internet Network Policies for developer application
F0033	Volume as a service and for container-as-a-Service
F0034	Support towards Containers
F0035	Enterprise with multiple heterogeneous runtimes/VIMs – Docker, Kubernetes, VMWare
F0036	Application deployment for user define group (Label)
F0037	CLI for local edge management
F0038	Docker compose - Support for External private Repo
F0039	Capacity license enablement

Table 3: IEAP release 2.3.2.0.0 features - 11/Dec/2024.

Feature key	Description
F0040	Persistent volume support for Helm, and docker apps (29 cases)
F0041	App upgrade for all application types
F0042	Multi component applications
F0043	Docker and helm configuration management.
F0044	Docker compose pull from App developers external repository
F0045	Docker Swarm
F0046	Metrics reporting to AWS Cloudwatch
F0047	Alarm/Event Notification to AWS SNS
F0048	DNS based Service Discovery for applications
F0049	Application config management
F0050	Extended roles with reduce scope
F0051	Adaptation of Edge/Tenant
F0052	Run time Cluster manager reduce scope - Infra as a service
F0053	Docker/VM Start and Stop
F0054	Lightweight deployment of IEAP, Co-Hosting of Central and Edge

F0055	Integration of IEAP (KeyCloak) with external LDAP
F0056	Portainer to SDK movement - Support low resource Runtimes (Docker based)
F0057	Automate certificate life-cycle management at IEAP Central and Edge
F0058	Volume as a service to be extended to dedicated cluster - Swarm and Docker host
F0059	Syslog support for upstream K8S, K3S
F0060	Support shared Runtimes - Docker node and docker swarm
F0061	Support for GPU - vGpu
F0062	Improvement for Notification information to make it more usable
F0063	Integration of Central/Edge certificate management with PKI(s) - EJBCA
F0064	CAMARA API Simple Edge Discovery
F0065	Application configuration management (Modify application configuration workflow as per SE approach)
F0066	File and Image management via EMP modules (Use SE modules for file and image management)
F0067	Support for ARM runtimes
F0068	Combined minimum deployment (Align with SE deployment approach)
F0069	API Gateway for securing/rate-limiting external APIs exposed by IEAP Central and Edge
F0070	CAMARA API Traffic Influence

3.1.2 E2.1 Barcelona Edge orchestrator development

The Barcelona Edge infrastructure has been finalized and is now operational for validation purposes. The final setup includes integrated computing and networking resources to support a wide range of applications and experiments.

3.1.2.1 Computing Resources

The infrastructure's computing component comprises multiple servers organized into distinct Kubernetes clusters. These clusters dynamically allocate resources based on application demands, including CPU, memory, disk capacity, and GPU support. Kubernetes nodes run either on virtual machines managed by OpenStack or directly on bare-metal servers—such as multimedia desktop towers—depending on performance requirements.

The i2CAT EDGE node operates within an OpenStack-managed cluster. In this environment, a dedicated tenant has been provisioned, allocating all resources to three virtual machines (VMs): one serving as the Kubernetes (K8S) master and two as K8S worker nodes.

For GPU-intensive workloads, the clusters leverage virtualized GPU resources through the NVIDIA GPU Operator. This approach enables efficient GPU sharing across multiple services, optimizing resource utilization while ensuring high-performance computing capabilities.

3.1.2.2 Networking Resources

The Barcelona Edge is connected to a dedicated VLAN within REDIRIS¹, the Spanish academic and research network that provides advanced communication services for universities and research institutions. This connection facilitates seamless interconnection between the Barcelona Edge, hosted at i2CAT, and the Madrid Edge, hosted at 5TONIC. This network setup ensures robust, low-latency communication essential for distributed edge applications.

Figure 7 presents a high-level overview of the edge computing infrastructure deployed in Barcelona. This setup comprises two distinct physical machines: an edge server and a tower PC (MIA Tower3). The edge server's resources are virtualized and managed through OpenStack, primarily allocating resources to three virtual machines (VMs). These VMs function as nodes within a Kubernetes (K8s) cluster, with one serving as the master node and the remaining two as worker nodes—one of which is equipped with GPU capabilities. The green rectangle inside the cluster represents the internal virtual network, used for node communication. Additionally, a fourth VM hosts a network function, the Congestion Detection Function (CDF).

MIA Tower3 operates an independent, all-in-one bare-metal Kubernetes deployment. The entire infrastructure is orchestrated by i2CAT's edge orchestrator, i2EDGE, which resides in the cloud. In the context of the 6G-XR project, interactions with i2EDGE are exclusively handled by the MEF Manager, also deployed in the cloud.

¹ <https://www.rediris.es/>

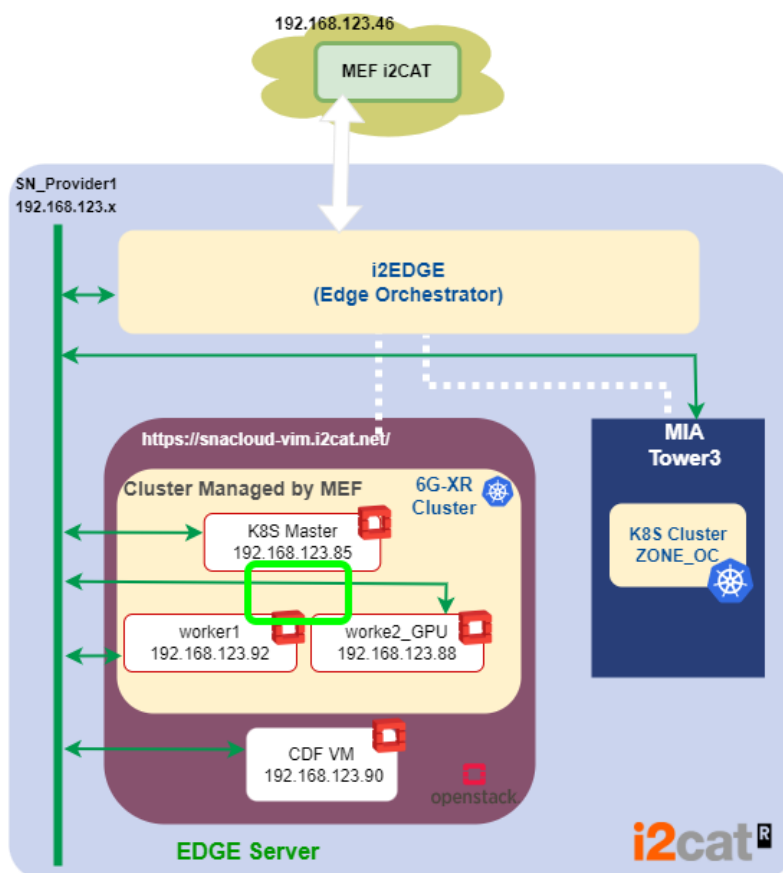


Figure 7: Edge in Barcelona, hosted by i2CAT

3.1.3 E4.1 QoD API development

The Quality on demand (QoD) API provides the capability of setting the quality for a flow within an access network connection and gets a notification in case the network cannot fulfil it. Further details on the functionality, methods and workflows were provided in D2.2 [1].

The development of this API has been realized in alignment with CAMARA Linux Foundation version 0.9.0, in which Capgemini is a contributor. The development period as part of the 6G-XR infrastructure has taken place from April 2024 to July 2024 (see feature F0007 in Table 1).

Among the methods already described in D2.2 the ones implemented at IEAP side are the following:

- POST /qod/v0/sessions
- GET /qod/v0/sessions/{sessionId}
- DELETE /qod/v0/sessions/{sessionId}

Due to the flow requirements for UCs, only the first POST method was found to be needed, the other methods mentioned above were deemed unnecessary.

Interaction with the NEF was done through the AsSessionsWithQoS 3GPP API. More specifically, calling the following API method:

- POST /3gpp-as-session-with-qos/v1/AS/subscriptions

An important consideration to take into account is that there is a difference between the 3GPP API request made from the IEAP and the request expected by the NEF, as illustrated in Table 4.

Table 4: 3GPP AsSessionsWithQoS version discrepancy

API	IEAP Version	NEF Version
AsSessionsWithQoS	1.2.0	1.1.2

Additionally, the QoS profile naming convention in CAMARA differs from the slice profile names defined in 5Tonic's NEF. The CAMARA QoS profiles represent four different QoS requirement levels that might be demanded by a service. The NEF slice profile names are the identifiers of the different slices provisioned in the 5G network, which will be tied to certain values of 5G QoS Identifier (5QI) and Data Network Name (DNN). Table 5 below shows the mapping among CAMARA QoS profiles and 5Tonic's NEF slice profiles. The NEF slice profiles named "-low" are defined with a Non-Guaranteed Bit Rate (NGBR) 5QI, while the ones named "-high" are defined with A Guaranteed Bit Rate (GBR). In conclusion, depending on the QoS profiles selected in CAMARA, the user traffic in the 5G network will be handled with a different QoS.

Table 5: QoS profile and slice profiles mapping

IEAP QoS profiles	NEF QoS slice profiles
QOS_E	madrid-low
QOS_S	barcelona-low
QOS_M	madrid-high
QOS_L	barcelona-high

Henceforth, it was decided that proxy components would be needed in-between to: (i) handle mappings that would format the fields from CAMARA standard to NEF implementation, and (ii) to adapt HTTP parameters to facilitate integration between both systems. The full communication workflow is depicted in Figure 8.

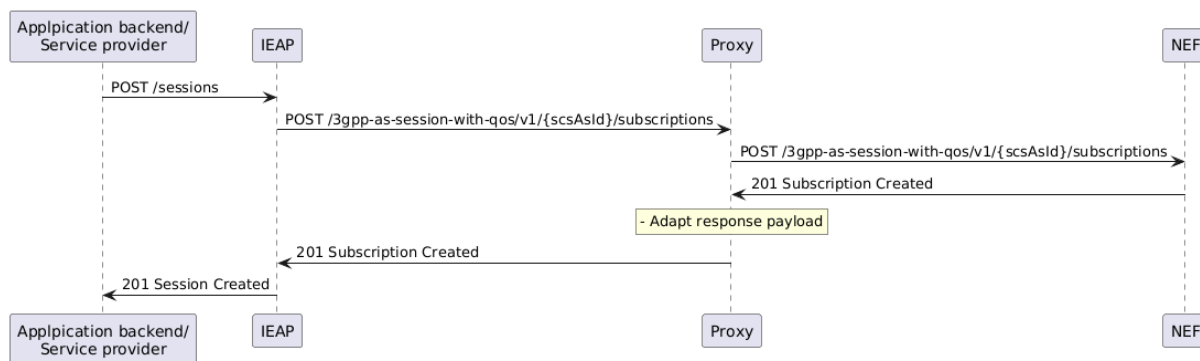


Figure 8: QoS session creation workflow

3.1.4 E4.2 Simple Edge Discovery API development

The edge discovery API enables retrieving a list of available and suitable edge platforms according to the requirements set for each application and as close as possible to the User Equipment (UE) location. Further details on the functionality, methods and workflows were provided in D2.2 [1].

The development of this API has been realized in alignment with CAMARA Linux Foundation of Simple Edge Discovery API version 0.9.3 in which Capgemini is contributor. The development period as part of the 6G-XR infrastructure has taken place from May 2024 to September 2024 (see feature F0064 in Table 3).

The Simple Edge Discovery API provides the following method which retrieves the closest edge to the UE as explained above:

- GET /mec-platforms: Returns the names of the available Edge Cloud Zones that are the closest to the user device identified in the request.

The API expects as input parameters information to facilitate the UE identification such as its IPv4 address and the Phone Number. Subsequently, the IEAP sends this information—along with additional parameters—to the NEF, which responds with data regarding the User Equipment (UE) location. The most relevant piece of information returned is the Tracking Area Code (TAC), as it is used by the IEAP to identify the nearest edge location.

For establishing the communication between IEAP and NEF, the 3GPP API MonitoringEvent was utilized, specifically the following HTTP method:

- POST /3gpp-monitoring-event/v1/{scsAsId}/subscriptions

Nevertheless, for the same reason explained in above - a version mismatch between requests made and expected for by IEAP and NEF respectively - a proxy was necessary for the communication. This version discrepancy is shown in Table 6. The full workflow is also shown in Figure 9.

Table 6: 3GPP MonitoringEvent version discrepancy

API	IEAP Version	NEF Version
MonitoringEvent	1.2.1	1.0.17.0

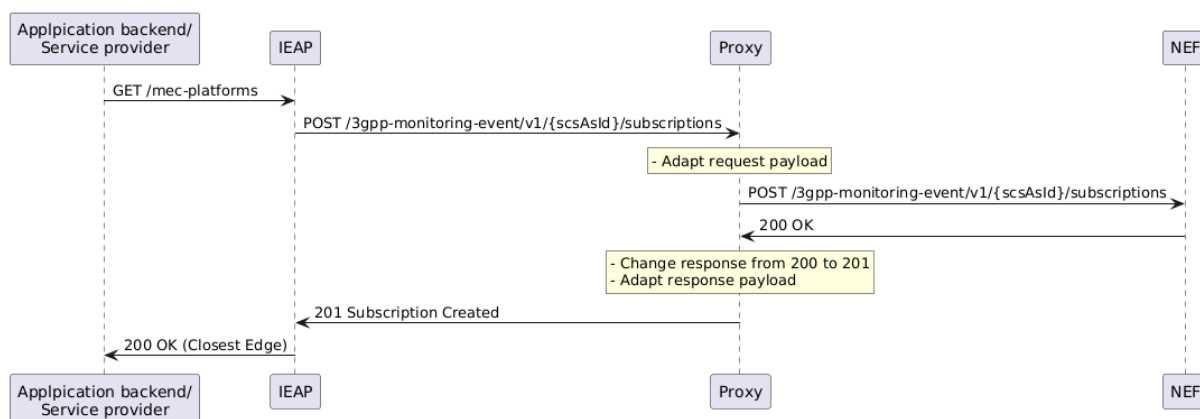


Figure 9: Simple Edge Discovery session creation workflow

3.1.5 E4.3 Traffic Influence API development

The Traffic Influence (TI) API provides the capability for the application providers to request the minimal latency in a specific geographical area. This is done by leveraging local instances of the application deployed at the Edge and influencing the traffic routing from the UE towards the edge instance of the application. Initial list of methods and first version of the workflow was provided in D2.2 [1].

The development of this API has been realized in alignment with CAMARA Linux Foundation version in which Capgemini is contributor. The development period as part of the 6G-XR infrastructure has taken place from October 2024 to December 2024 (see feature F0070 in Table 3).

Despite TI API having been developed and being fully functional at IEAP side, no tests could be conducted in the context of 6G-XR as 3GPP Traffic Influence API is not available at NEF side. Nevertheless, leveraging QoD API and AsSessionsWithQoS 3GPP API already explained in section 3.1.3, The traffic was successfully influenced by changing the UPF to which the different UEs were connected. Further information about the validation is available in section 5.1.4.

3.1.6 E5.1 Edge Federation development

The 6G-XR project has decided to federate the two edge infrastructures in the South Node by adopting the GSMA Operator Platform Group (OPG) standard for the East-West Bound Interface. To achieve this, the MEF (MEC Federator Manager) was implemented as the key component for interfacing with other operators' platforms via the East-West Bound Interface.

Development began by ensuring that i2EDGE supported all the required functionalities for its designated roles, a process initiated in April 2024. In parallel, the development team worked on the MEF API, which implements the EWBI to expose the functionalities required by the partner operator. The MEF API implementation was completed in May 2024, followed by internal validation of all components in June 2024.

The MEF is responsible for the following key functions:

- Managing federations between operators.
- Synchronizing information on shared resources with partner operators.
- Exposing and monitoring edge cloud resources to other partner operators.

- Facilitating the transfer of application artifacts (e.g., container images) and generating the necessary metadata at partner operators.
- Handling the lifecycle management of instantiated applications.

The MEF Federator interacts with the MEC platform, which fulfils the roles of Capabilities Exposure and Service Resource Management. Within the Barcelona EDGE infrastructure, i2EDGE serves as the MEC Platform Manager, orchestrating Kubernetes clusters and overseeing the lifecycle of deployed applications.

In the context of 6G-XR, the external interface at the Barcelona Edge—accessible to partner operators—is exclusively provided by the MEF. The i2EDGE interface remains strictly internal, facilitating interaction with the MEF, which handles all external communications.

The following diagram (Figure 10) illustrates the two EDGE locations in the South Node, Madrid and Barcelona, sharing resources via federation through the EWBI.

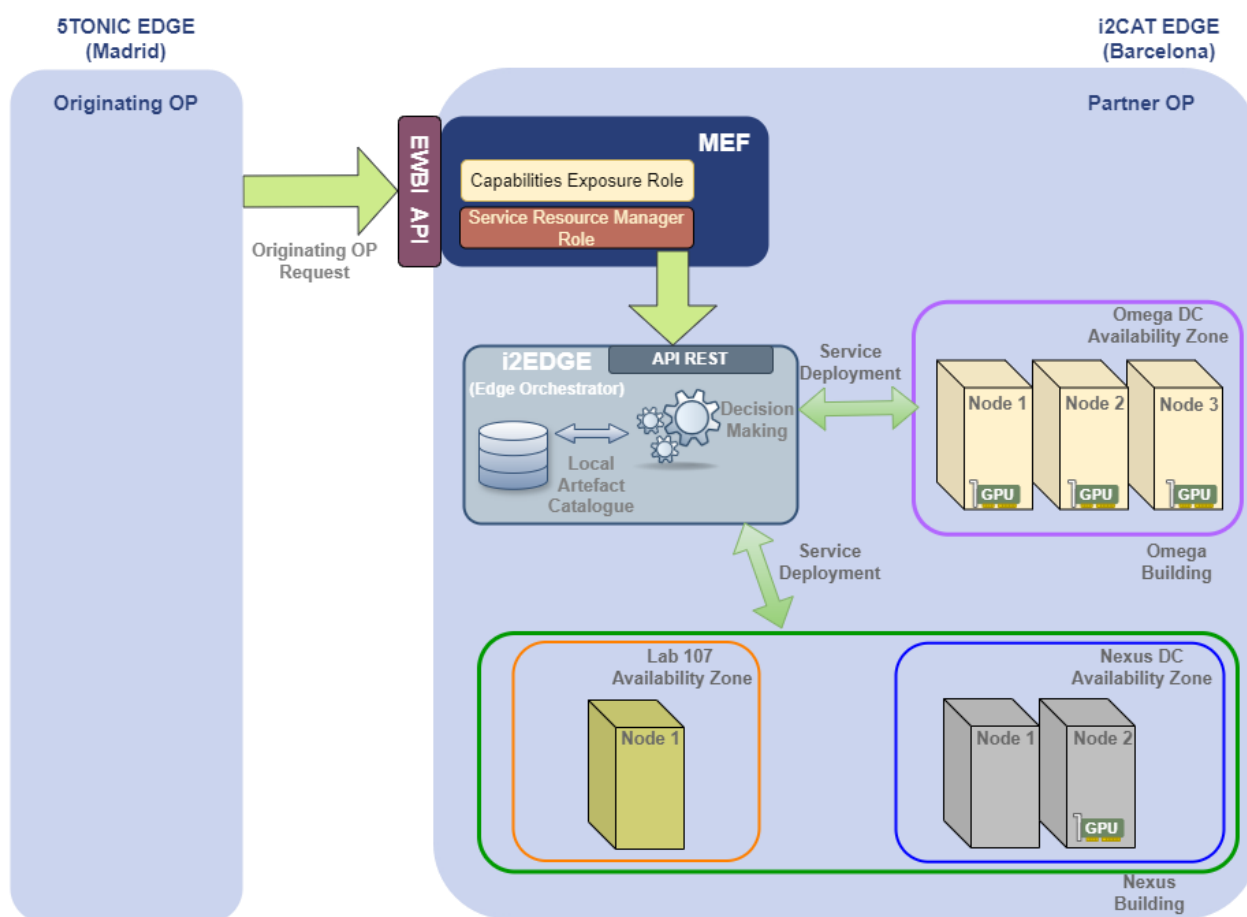


Figure 10: EWBI API and the two edges in the South Node

i2CAT led the implementation of the MEF. To validate the MEF component, a dummy operator was developed by i2CAT to test and verify the API functionality. The validation effort took place during the month of July 2024. This approach allowed for streamlining the appropriate method calls, which significantly facilitated the integration with the Edge in Madrid.

It is worth noting that the GSMA’s East-West Bound Interface (EWBI) specification is an evolving standard. While future versions may introduce some changes, the core functionality appears to be stable, and no significant modifications are anticipated based on the current version.

3.1.7 E6.1 Service Parameter API development

This enabler is aimed to assign the most appropriate slice to the end users. On a high level, the features required are the retrieval of the list of available slices in a non-public network (NPN) and the assignment of an end user to a slice.

The development of this enabler was impacted by a delay in the deployment of a Policy Control Function (PCF) in 5Tonic’s 5G Core, which was not available before December 2024 (M24). A solution which was not described in D2.2 [1] was implemented. This solution meant two main changes: First, the southbound interface (SBI) had to perform the requested actions on the Unified Data Repository (UDR). Second, the lack of a PCF also implied the absence of the UE Route Selection Policy (URSP) feature. Without this feature, it is not possible to push a change of Data Network Name (DNN) to a User Equipment (UE) from the network. Thus, it was decided to use four different slices in the network, all with the same DNN, and the selection of UPF would be done according to the slice identifier. As a result, the assignment of slice to a particular end user would be done via the NBI API POST `/subscriptions/{subscriptionId}/profile/{profileId}` or POST `/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions`.

The workflow for getting the list of slices available in the network using UDR is illustrated in Figure 11.

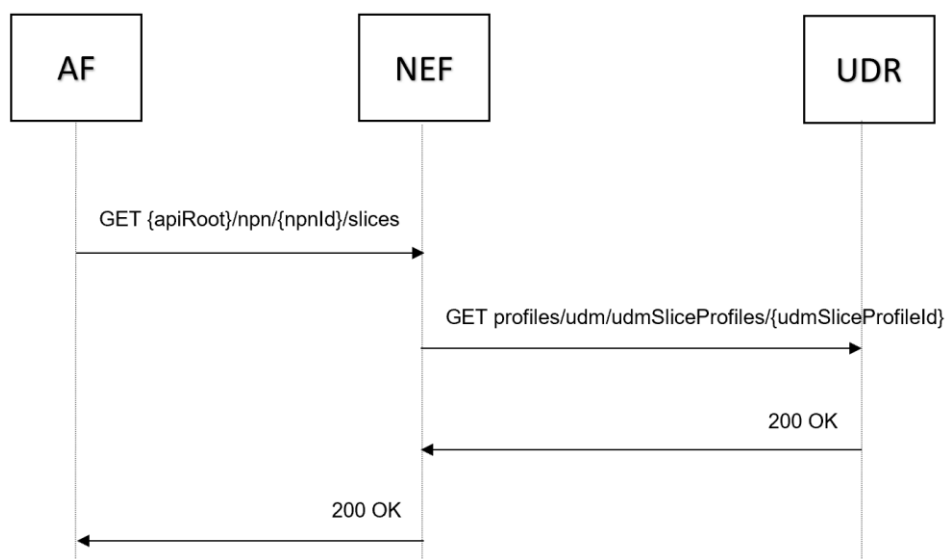


Figure 11: API workflow for slices list retrieval using UDR

The Application Function (AF), which is the API client, will trigger the slices list retrieval available in a specific NPN by calling the GET `/nnp/{nnpId}/slices` method. Then the NEF propagates this request southbound by sending a GET `profiles/udm/udmSliceProfiles/{udmSliceProfileId}`. The field `udmSliceProfileId` is equal to “physical_twins.{nnpId}.slice-profile”. Finally, the HTTP 200 response is sent to NEF and to the API client with the list of available sliceIds.

Figure 12 represents the workflow for assigning a slice to a specific end user.

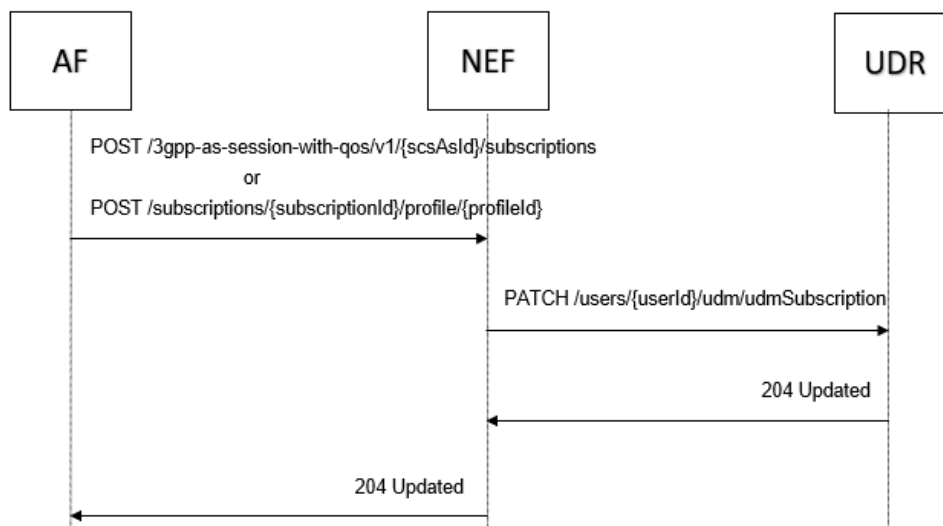


Figure 12: API workflow for slice assignment using UDR

First, the AF sends a NEF Service Parameter API request (POST /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions or POST /subscriptions/{subscriptionId}/profile/{profileId}) to select a specific slice profile from the available slice profiles for a user, providing the UE IP or the Generic Public Subscription Identifier (GPSI) and the selected slice profile identifier. Then the NEF forms a PATCH /users/{userId}/udm/udmSubscription request using the GPSI for the userId variable and including the obtained profileId in the body to identify the slice. The UDR will update the assignment, and if successful, will send a HTTP 204 response to the NEF and the API client.

Regarding the SBI, an action was listed in the development backlog in May 2024 to find and test methods to interact with the Unified Data Manager (UDM) and the Unified Data Repository (UDR) to create or modify slicing profiles. As a result, during May the methods to see all UDM profiles registered, to create a UDM slice profile, to see a user slice profile in UDM and to set a UDM slice profile to a user were designed. These methods were tested in the following days and the item was marked as done in June.

The item in the backlog for implementing the retrieval of the list of available slices in a specific NPN (GET /nnp/{nnpId}/slices) was assigned to a developer in July 2024. This task was completed in October.

A task for building the SBI API to update a slice profile in the UDR (POST /subscriptions/{subscriptionId}/profile/{profileId}) was created in the development backlog in August 2024. The feature was developed and tested in October.

The tasks allowing to use this enabler with UDR were completed around two months later than originally planned, but this fact did not compromise the success of the project thanks to the buffer time allocated in the plan.

3.1.8 E6.2 UE Location API development

This API allows to obtain information about where a User Equipment (UE) is located. It is useful for being able to monitor the cell that is providing the service to the UE or to assign the closest User Plane Function (UPF) to the end user. More details on the API can be obtained in D2.2 [1]. The notes made in the activity backlog by the development team are reported below.

In May 2024 the item to obtain the cell where a UE is located via Access and Mobility Management Function (AMF) providing the UE IP address as input was created in the development backlog. During May the API was defined, and the task was considered completed in July.

In June 2024 the requirement was registered in the backlog to obtain via Session Management Function (SMF) the cell and Tracking Area Identity (TAI) of a UE providing its IP address as input. In July the solution was designed to retrieve from the SMF the GPSI or Mobile Subscriber Integrated Directory Number (MSISDN) from an IP address, and to retrieve the cell and TAI provided an IMSI. In August, this feature was integrated in the exposure function.

During August 2024 a task was defined and implemented to extend the API to obtain the cell where a UE is located via SMF providing GPSI as input.

The development of these features was completed 2 months later than the original plan but had no impact on the project because of the existing buffer time in the plan.

3.1.9 E6.3 QoS Session API development

The NEF QoS Session API facilitates the modification of the QoS value of the user plane traffic generated by a UE, in order to adapt to the service needs. There was a modification from the workflow described in D2.2 [1]. Instead of the method PUT /subscriptions/{subscriptionId}/qos between CAMARA and NEF, it has been replaced with POST /subscriptions/{subscriptionId}/profile/{profileId}. The final workflow is depicted in Figure 13.

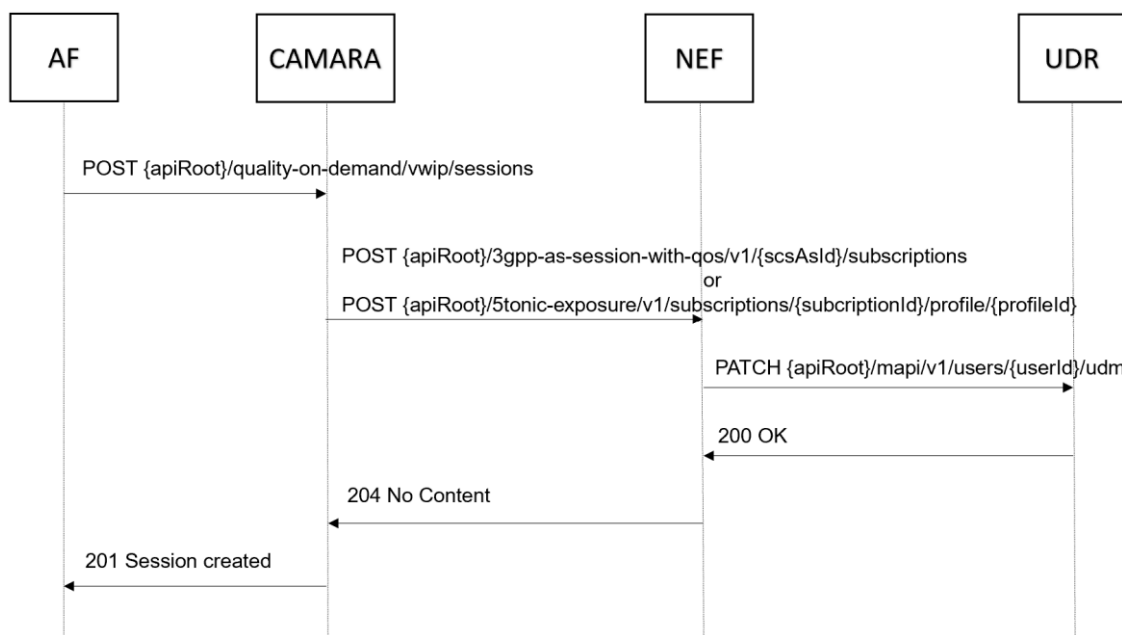


Figure 13: API workflow for QoS change

Initially, the AF initiates a CAMARA QoD API request (POST /sessions) to request a QoS change of an existing device connection, specifying the desired QoS Profile and other parameters like the identifier of the device (IPv4 address or Phone number) and the identifier of the application server. Then, the CAMARA QoD calls NEF by means of QoS Session API (POST /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions or POST /subscriptions/{subscriptionId}/profile/{profileId}) to modify device connection QoS, providing the UE IP or the GPSI and the requested QoS Profile. Next, Southbound in the network, the NEF makes a request to UDR, to use the received QoS Profile, that is mapped to a set of

QoS parameters, for the requested user data flow. Then, the UDR propagates a request to the SMF, which initiates the PDU session modification, and acknowledges the NEF with a 201 response. Then, the NEF returns HTTP response code 201 with the information of the created subscription to CAMARA and the CAMARA QoS creates a QoS session for the QoS change request and stores the session information. The status of the QoS session is set to REQUESTED, which indicates that QoS change has been requested by creating a session. Finally, the CAMARA QoS returns HTTP response code 201 with the information of the created QoS session (including session ID and the REQUESTED status) to the AF.

The methods involved here are the same as already referred to in section 3.1.7 Service Parameter API development. So, as it was mentioned, the methods to interact with the UDM and UDR to create or modify slicing profiles were tested and validated by June 2024, and the NEF NBI API to update a slice profile was considered completed in October.

The enabler development was completed less than a month later than the original plan.

3.1.10 E6.4 Data Collection API development

This enabler is devoted to providing network metrics to an API client. Those metrics can be used later for example for detecting congestion. More information about the API detailed in D2.2 [1]. The notes made in the activity backlog by the development team are reported below.

In May 2024 the request to develop an API to fetch Key Performance Indicators (KPIs) in the exposure function was noted in the backlog. It is required that different KPIs can be stored by cellId or by UE IP. It was completed in May.

The throughput per UE counters (*UEUplinkThroughput*, *UEDownlinkThroughput*) were introduced in the backlog in May 2024. Code changes were committed in September 2024.

Also in May 2024, a task was created for developing the throughput per cell counters (*RanDownlinkThroughput*, *RanUplinkThroughput*). The implemented code was committed in September 2024.

In May 2024 a task was noted to develop a metric to calculate the number of active users on a cell (*ActiveUsersDL*, *ActiveUsersUL*). Completed in September 2024.

In July 2024 the requirement to include the following KPIs proposed by i2CAT was noted in the development backlog:

- HARQ transmission counters:
 - pmMacHarqDIAck16Qam - Total number of successful downlink HARQ transmissions using 16-QAM modulation.
 - pmMacHarqDIAck64Qam - Total number of successful downlink HARQ transmissions using 64-QAM modulation.
 - pmMacHarqDIAck256Qam - Total number of successful downlink HARQ transmissions using 256-QAM modulation.
 - pmMacHarqDIAckQpsk - Total number of successful downlink HARQ transmissions using QPSK modulation.
 - pmMacHarqUIAck16Qam - Total number of successful HARQ transmissions in uplink using 16-QAM modulation.
 - pmMacHarqUIAck64Qam - Total number of successful HARQ transmissions in uplink

using 64-QAM modulation.

pmMacHarqUIAck256Qam - Total number of successful HARQ transmissions with 256-QAM uplink.

pmMacHarqUIAckQpsk - Total number of successful HARQ transmissions in uplink using QPSK modulation.

- Distribution of MCS counters:

pmRadioPdschTable1McsDistr - Distribution of MCS used for PDSCH transmissions where MCS index table 1 is applied and UE instances are enabled with up to 64-QAM.

pmRadioPdschTable2McsDistr - Distribution of MCS used for PDSCH transmissions where MCS index table 2 is applied and UE instances are enabled with up to 256-QAM.

pmRadioPdschTable3McsDistr - Distribution of MCS used for PDSCH transmissions where MCS index Table 3 is applied and UE instances are enabled.

pmRadioPuschTable1McsDistr - Distribution of MCS used for PUSCH transmissions where MCS index table 1 is applied and UE instances are enabled with up to 64-QAM.

pmRadioPuschTable2McsDistr - Distribution of MCS used for PUSCH transmissions where MCS index table 2 is applied and UE instances are enabled with up to 256-QAM.

pmRadioPuschTable3McsDistr - Distribution of MCS used for PUSCH transmissions where MCS index table 3 is applied and UE instances are enabled.

Ericsson proposed to include the following counters:

- RBSym utilization:

pmMacRBSymUtilDIDistr - Distribution of utilization for downlink resource block symbols averaged over 1 second.

pmMacRBSymUtilUIDistr - Distribution of utilization for uplink resource block symbols averaged over 1 second.

In September 2024 the RBSym utilization counters were implemented in the exposure API with names *UtilizationDistributionDL* and *UtilizationDistributionUL*. In September 2024 the HARQ and MCS counters were included in the exposure.

This enabler was completed two months later than expected in the original plan, but did not impact the overall progress of the project as it was in line with the targeted time to have all enablers ready and the deployment and validation phases were shortened.

Some bug corrections were triggered and solved during November 2024: (i) to set KPIs to N/A if there is any internal error in the retrieval of the data, (ii) to set the KPI called *TrafficKpiUeData* to N/A whenever there is no *targetUe* input provided and (iii) to set the KPI called *TrafficKpiRanData* if there is no *cellID* input provided.

3.2 SOUTH NODE CONTROL PLANE ENABLERS

This section provides details on the development efforts done for the enablers of UC3 *Control plane innovations* in the South Node (see UC description in D1.1 [2]).

3.2.1 E9.1 IMS Data Channel Server development

The IMS Data Channel Server is described in D2.2 [1].

For the development phase of the IMSDC server, the work plan was divided into two phases. The initial development was carried out in the IMS core of the Ericsson lab in Sweden because the 5Tonic lab was not available to start the first tests. The idea was to advance in parallel with the PoC while the 5Tonic lab continued its deployment. Once the deployment was completed at 5Tonic, the only thing that would have to be done was to migrate from the Sweden servers to the new ones in 5Tonic (IMS Core) hosted in Azure in Madrid (IMSDC server).

Regarding architecture and signaling, connectivity between the different components (Ericsson IMS Core in Sweden lab, DCS in Azure, and Matsuko servers in Azure) were completed. The information and message exchange between the DCS, the devices and the application servers are described in D2.2 [1].

Firstly, the service number (which is the number the client would dial to start the service) was activated in Ericsson Sweden lab DCS, allowing clients to access it via VoWiFi. But in the case of 5Tonic lab, the VoNR alternative has been enabled. Although VoNR offers more limited range, this technology allows a more stable and efficient connection in areas with 5G coverage.

Once the connection between all the elements involved was established, efforts were focused on optimizing connectivity and data transmission performance through the end-to-end Data Channel connection to ensure that the DCS enabler provides the necessary capacity and meets the requirements to provide a third-party application like MATSUKO with the desired final user experience.

These efforts include optimizations in the connectivity and in the performance of the WebRTC/WebGL application. For example, there was a configuration in the ICE (Interactive Connectivity Establishment) candidate policies parameters such as “bundle policy” to control how different media streams (audio, video, data) are multiplexed or not on a single connection. Additionally, file compressions with Gzip and Brotli were used to reduce the file size from 62 MB to 10 MB and to improve data transmission reducing bandwidth usage and optimizing the application's overall performance in different environments.

Finally, a significant issue was detected in the initial testing because the current WebRTC standard implementation shows a 256 kB limitation for data transmission (In the case of MATSUKO's application, a frame was approximately 5 MB in size). The most effective solution was to split the data into 65.000-byte chunks, using HTTP to control the order of the chunks, and adding additional headers to facilitate reassembly on the client.

3.3 NORTH NODE 3D DIGITAL TWIN ENABLERS

This section reports the work done on the enablers for UC4 *Collaborative 3D Digital Twin-like Environment* in the North Node (see D1.1 [2]).

3.3.1 E3.1 North Node adapter development

The North Node Adapter (NNA) is described in D2.2 [1] and so far, the NNA development work has been carried out as described in the D2.2 document.

The NNA has been implemented using the Python programming language and runs on a Linux-based virtual machine under Supervisor, a process control system that allows its users to monitor and control several processes. Python was selected as the development language for its simple syntax and availability of a wide range of useful software packages.

The Python packages and their versions used by the NNA are listed in Table 7.

Table 7: Python packages used by the NNA

Python package	Version
Flask [7] [7]	3.0.3
Waitress [8] [8]	3.0.0
Requests [9] [9]	2.32.3

The Flask package is used for implementing a REST API to be called by the North Node Web portal (for starting and stopping experiments and getting the status of ongoing experiments). Waitress, in turn, is a Web Server Gateway Interface (WSGI) server, which is used to run the NNA's Flask-based REST API on the Linux server. Waitress is a production-quality pure-Python piece of software, which runs in one process and uses several thread workers in its operation. Finally, Requests is a software library, which the NNA uses to make REST queries to other NNA components (Qosium, AI/ML, OSM, and Cumucore). The REST API provided by the NNA was implemented in M22.

The NNA application is implemented using object-oriented design, which consists of several classes. The class structure of the NNA application is depicted in Figure 14. The main class is Experiment, which contains the logic of an experiment and provides an API for FlaskApp for controlling the experiment (starting, stopping, and getting the status of the experiment). The Experiment class was implemented in M24 of the project.

FlaskApp is a Flask application, which contains four decorated functions. These functions constitute a REST API for the North Node web portal. The FlaskApp along with the REST API was implemented early in the project, in M22. The REST API has been described in detail in D2.2 [1].

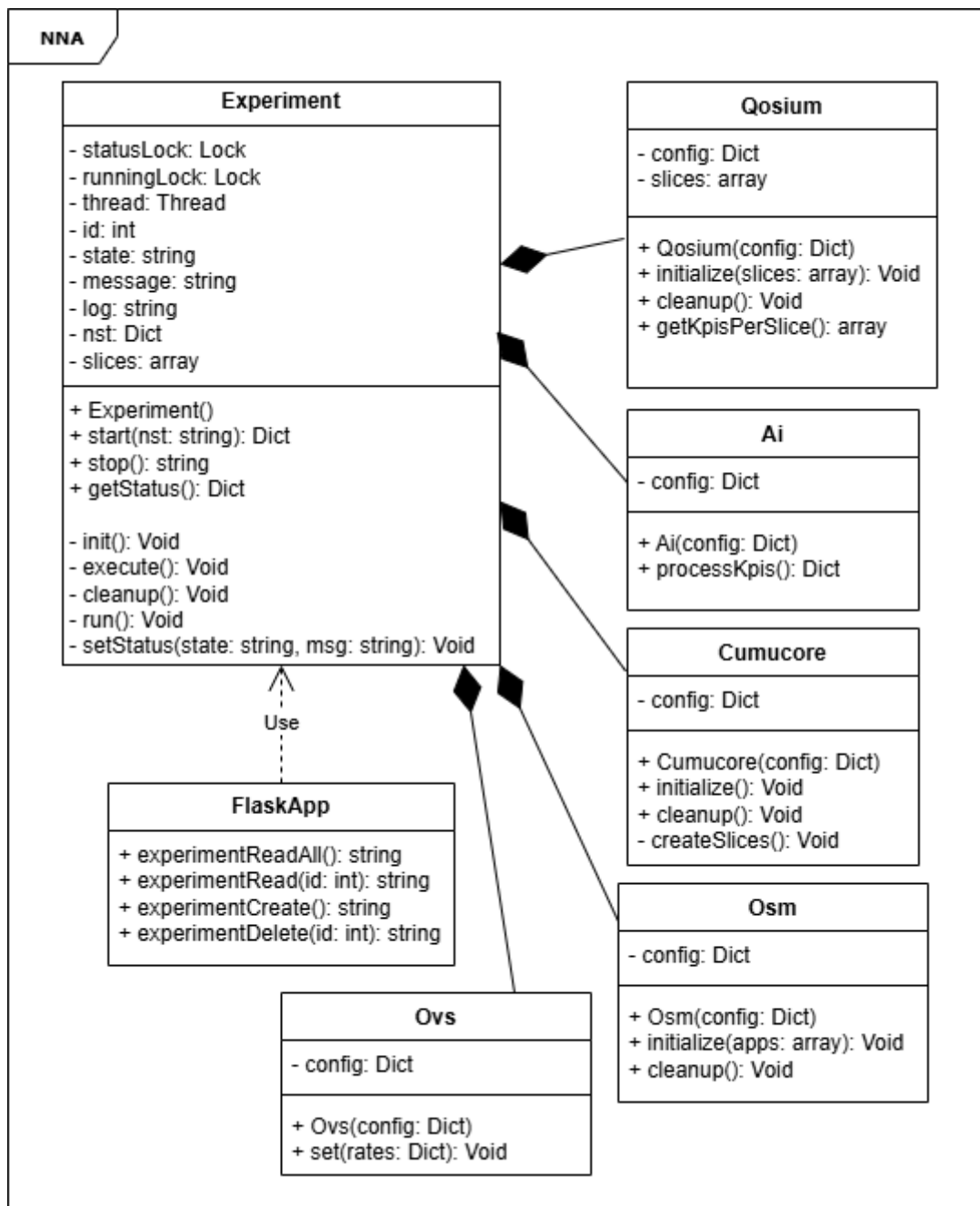


Figure 14: Class diagram of the NNA application

Experiment uses threading to run the logic of an experiment, which means the inner variables of the Experiment class are protected with mutexes, i.e. Python locks. There is one lock for status variables and one lock for the thread variable inside the Experiment class. In theory, there could be several Experiment classes (i.e. several experiments run simultaneously) managed by the FlaskApp class, but for the sake of simplicity in our implementation, only one static Experiment instance is created and used.

The Experiment class has 5 aggregated classes with their own responsibilities in the application logic. The classes are Qosium, Ai, Cumucore, Osm, and Ovs.

The Qosium class is responsible for managing the actual measurements carried out in an experiment and for providing measured DL/UL KPIs (per slice) for the Experiment class. The Qosium class interacts with Qosium measurement system (Storage) through a REST API. This class was implemented in M22.

The Cumucore class was implemented next in M23. It is responsible for setting up and tearing down slices in the beginning and end of an experiment according to the NNA configuration.

Next, the Ai class was finished in M23. The Ai class interacts with the AI/ML component described in detail in section 3.3.3 and is responsible for feeding KPI data from Qosium to the AI/ML component and getting per slice allocation back in response.

The Osm class handles instantiation of virtual machines and installation of applications in them when an experiment is started. This class interacts with Open-Source Mano (OSM) component through a REST API. It was implemented in M24.

Finally, the Ovs class was implemented last in M26. The Ovs class is responsible for interacting with Open vSwitch (OvS), which has all Cumucore UPFs connected and manages throughput on N3 and N6 interfaces. The Ovs class sets the DL/UL ingress rates according to the allocation decision from the AI/ML component using JSON-RPC-formatted messages to the switch.

3.3.2 E3.2 3D Digital Twin development

The 3D Digital Twin system has been developed as planned, utilizing the North Node Adapter API interface as the Edge service XR application within the Cumucore local 5G network, and is organized into the following functional blocks as shown in Figure 15.

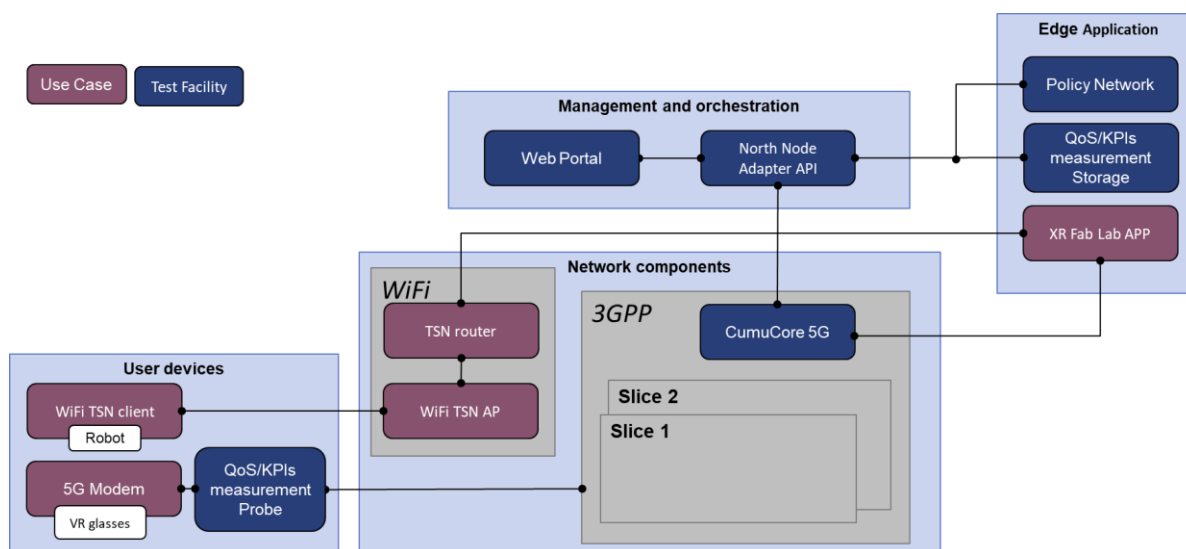


Figure 15: functional diagram of 3D digital twin and the Edge services

- **5G System Connectivity:**
 - The 5G system connects the local edge services, the XR Fab Lab facility, and XR Fab Lab users.
 - Individual slices are allocated to each user within the 5G radio network to ensure an adequate quality of service.
- **XR Fab Lab Application:**

- The XR Fab Lab application is hosted on the edge service, delivering and operating XR functionalities over the local 5G radio system.
- XR Fab Lab users can communicate with one another via VR glasses while remotely controlling the 3D printer in the Fab Lab Facility.
- **Network KPI Measurement:**
 - Quality of Service and Network KPIs are measured on the XR user side within the 5G Radio.
 - The collected measurement data are sent and stored on the edge service storage.
- **Edge Service Policy Network:**
 - Within the edge service, a policy network monitors the traffic of the stored data. It controls the 5G network slice via the North Node Adapter to enhance XR service performance.
- **5G Cumucore Adaptation:**
 - The 5G Cumucore adapts the slices proposed by the policy network function in the edge services. The policy network controls the slices on Cumucore optimizing the 5G radio network performance dynamically based on traffic conditions.

The implementation of this 3D Digital Twin use case, incorporating an AI/ML-based Policy Network and 5G slicing, begins with establishing connectivity for all 5G devices under the Cumucore 5G system in M22. QoS and KPI measurements on the 5G network were configured using Qosium Probe and Qosium Storage during M23–M24, integrated with the Edge services of the Digital Twin application and the AI/ML-based Policy Network. In M24–M25, end-to-end verification with VR glasses was conducted, utilizing the NNA’s REST API to orchestrate the edge services and manage 5G slices.

3.3.3 E3.4 Resource Optimization development

The AI-driven resource optimization component is developed to provide real-time, intelligent network slice resource allocation based on dynamic network KPIs. As described in D2.2 [1], this component plays a critical role in enhancing the performance and responsiveness of network slicing by enabling data-driven decisions that adapt to the current state of the network.

In December 2024, the development of the resource optimization model was tightly integrated with the North Node Adapter (NNA), which serves as the gateway for real-time network telemetry. The AI module receives KPIs from the NNA via a REST API, processes them into a state vector, and returns optimal allocation weights for each network slice.

During the development phase, it became evident that static rule-based resource allocation was not sufficient to meet the responsiveness and adaptability required by the project use cases. Thus, the reinforcement learning approach was selected to enable adaptive decision-making based on real-time observations. The training of the AI model was initially conducted in a simulated environment during January 2025, using synthetic KPI data, and the model was later exported and deployed in a virtualized environment in March 2025 after the training.

The AI model is encapsulated within a Virtual Machine that includes:

- The trained PyTorch model
- The Flask-based REST API interface
- Model inference logic
- Required runtime dependencies

This packaging ensures a modular deployment that is platform-agnostic and can be easily integrated with the existing 5GTN infrastructure.

The API exposed by the resource optimization module includes an `/allocate_resource` endpoint, which receives network state and KPI information in JSON format and returns resource allocation decisions. The allocation results are typically expressed as proportional weights for each slice (e.g., slice1: 0.6, slice2: 0.4) that guide the actual enforcement of resources at the data plane level.

3.3.4 E7.1 Cumucore Slice Management API development

Cumucore provides a REST API for managing network slicing. Using the Network Slice Management API the user can define slice sizes, quality parameters and traffic rules for each slice. As described in D2.2 [1], the Cumucore slice management API was supposed to be used in the 5GTN facility for dynamic slice management by the NNA.

During our development, we found out that the Cumucore REST API is not able to handle dynamic network parameter management. Therefore, we tried a UPF-based solution for allowing different throughput capacities for slices. In this approach (tried around M22), we intended to use several pre-configured UPFs with different throughput capacities: whenever a slice needed a higher or lower throughput, the NNA would assign a different UPF with higher or lower capacity to the slice on-the-fly. The process of assigning a new UPF to a slice is as follows:

1. Remove the current UPF assignment to the slice via the REST API
2. Attach new UPF with different configured throughput to the slice via the REST API
3. Configure a new static IP address from the new UPF's pool to UE via the REST API
4. Manually re-attach the UE to the network to get the configured static IP address as part of the PDU session establishment process

In this UPF-based solution, we configured each UPF with an IP address pool which means that UE would get a static IP address when a slice was assigned to different UPF. The problem with this approach was that for the UE to get a new IP address (from the new UPF's pool) a new PDU session establishment is needed. This means that every time we assign a UPF to a slice, the UE must disconnect and re-attach to the network manually. We found that this solution does not work for the project use cases as the management of slices and their throughputs must be as automatic and programmatic as possible.

Therefore, Cumucore is not used for dynamically adjusting slice parameters as it is not supported by the Cumucore product. We still use Cumucore as the core network in our use case (responsible for creating and deleting slices, and managing UEs and UPFs), but dynamic resource allocation is enforced using Open vSwitch (OvS), which has N3 and N6 interfaces connected. To achieve dynamic resource allocation based on decisions from AI/ML, we adjust ingress policy rates on the N3 and N6 interfaces on-the-fly at the switch. This OvS-based method was implemented and tested in M24 of the project.

3.4 NORTH NODE ENERGY FRAMEWORK ENABLERS

This section contains the development work done on the network enablers for UC5 *Energy Measurement Framework for Energy Sustainability* in the North Node (see D1.1 [2]).

3.4.1 E3.3 Energy Management development

The North Node energy measurement framework, described in D2.2 [1], has been implemented, with the integration phase now complete. The developed features inside the energy measurements framework include real-time data collection of the production side (renewables), consumption side (open source and commercial gNB sites) as well as the storage side (Battery Energy Storage System, BESS), including real-time inverter data, solar chargers, external meters and on-site sensors data and energy consumption of various components within the E2E data path, as shown in Figure 16.

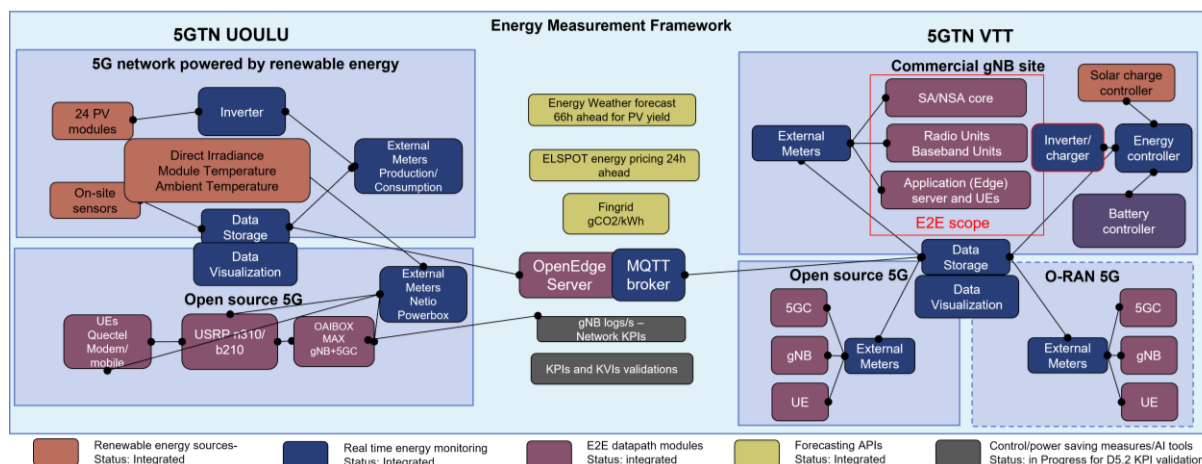


Figure 16: Updated North Node High level architecture

These datasets lay the foundation for developing energy budgeting, which aids in control and power-saving measures for experimentation research and will be reported in D5.2. Key developments of features thus far from the perspective of energy management as an enabler and E2E energy efficiency include:

- Almost real-time electricity metering system covers the whole E2E-system was developed in May 2024. It now includes Netio Powerbox and Carlo-Gavazzi, EM111 and EM511 energy analyzers.
- Nokia OpenEdge Blade server was deployed as a central controller in March 2024 for storing and managing data orchestration within the energy measurement framework to enable bridge connection between North Node gNB sites for orchestration, data exchange, and power saving measures.
- The deployment of forecasting APIs inside the central controller to external data services such as FMI (Finnish Meteorological Institute) energy weather forecast for the next 66 hours, ELSPOT hourly electricity spot pricing for the next 24 hours, and real time CO₂ emission estimates to grid intake by Fingrid Open data (absolute gCO₂/procured kWh's). This task was completed in May 2024.
- Real-time monitoring of the collected data using a central database was deployed using Grafana panels. Energy budgeting for the next 24 hours was developed to assist and provide control functionalities for the selection of the optimal parameters for validation of the KPIs provided in D5.1 [10] and will be reported in D5.2.
- The deployment of the Mosquitto Message Queuing Telemetry Transport (MQTT) broker inside the central controller enables publish/subscribe (pub/sub) data exchange, establishing a bridge connection between the North Node gNB sites. The forecasting APIs are accessible

to each 5GTN network component, allowing them to subscribe for control and assistance in selecting either full power or power-saving modes based on RAN configurations.

The development of the features phase has been completed both from UOULU and VTT sites. Additionally, the North Node energy measurement framework has been implemented for validation of KPIs and the sustainability experimentation framework since January 2025. The real-time data collection and the historic data sets are now available at the North Node side for validation and testing purposes. The only change from the proposed architecture is the introduction of open source 5G environment OAIBOX at UOULU site instead of using a commercial base station.

3.4.2 E8.1 OAIBOX development

As described in D2.2 [1], the OAIBOX will serve as an enabler within the 5GTN facility for the energy measurement framework, acting as an open source 5G environment for sustainability experimentation. The integration of OAIBOX with the energy measurement framework has been completed. The developed features enabled by the framework at the UOULU site include real-time power consumption monitoring of individual OAIBOX components, facilitated by an external meter (Netio Powerbox).

For data exchange between North Node sites, an MQTT bridge has been established. Each component of the open source 5G environment can publish power consumption data to the central controller through the MQTT broker (Mosquitto). These components include the OAIBOX gNB and the Open5GS core, the USRP N310 and the USRP B210, as well as the Quectel Modem 500 and the Quectel Modem 520 acting as UEs. The enabler is developed and integrated into the North Node 5GTN facility. It facilitates the sustainability framework in the following ways:

- **Energy Measurements:** Real-time energy consumption measurement of all the components of open source 5G environment (OAIBOX, USRPs, UEs) was enabled in June 2024 using Netio PowerBox as external meter.
- **Restricting 5G NR Bandwidth:** Using the OAIBOX dashboard, various bandwidth configurations were tested including 100 MHz, 80 MHz, 60 MHz, 40 MHz and 20 MHz in November 2024 to observe the impact of changing bandwidth on the energy consumption of the E2E OAIBOX setup.
- **Selecting Time-Division Duplex (TDD) Frame Structure:** Various TDD slots configurations were tested in December 2024 which includes configuration options such as TDD=7*DD-F-2*UU, 3*DD-F-1*UU, 7*DD-F-2*UU, 2*DD-F-7*UU, 2*DD-F-2*UU, 5*DD-F-4*UU. The impact of changing TDD slots was observed and stored inside the central database.
- **Modulation Constellation Restrictions:** MCS for DL and UL was changed using the configuration file and the upper limits were fixed using the yaml file in November 2024, such as QPSK (max_mcs =4), 16QAM (max_mcs =10), 64QAM (max_mcs =19) and 256QAM (max_mcs =28). The impact of changing MCS over energy consumption was observed and stored inside the central database.
- **DL MIMO Mode Selection:** Finally with the combination of different bandwidths, the effect of MIMO 2x2 and SISO 1x1 was also observed and stored in central database in December 2024. All the results will be analysed in D5.2.

4 ENABLERS DEPLOYMENT PHASE

This section reports the work done to deploy the 6G-XR WP2 enablers on the experimental nodes.

4.1 SOUTH NODE USER PLANE ENABLERS

This part describes the deployment and setup of the enablers needed for UC1 *Resolution Adaptation or Quality on Demand* and UC2 *Routing to the Best Edge* in the South Node (see D1.1 [2]).

4.1.1 E1.1 IEAP Edge orchestrator & APIs deployment

The 5TONIC IEAP and related APIs deployment consists of several servers that host VMs with Ubuntu OS, that have different functions.

IEAP Central is the bare metal server that contains the core of the IEAP installation, this is the MEC Orchestrator and all the enabler APIs explained above. The IEAP application is installed on a Docker setup inside the server.

For the edge infrastructure where the IEAP will deploy the applications, OpenStack VMs are provided to work as the edge nodes. More details were provided in D2.2 [1].

Regarding the proxy developed for interaction with NEF that manages requests for API workflows — QoD and SimpleEdgeDiscovery—already detailed in sections 3.1.3 and 3.1.4, it has been containerized through Docker and deployed in a Kubernetes environment also at the 5TONIC testbed.

Therefore, a refined version of the IEAP deployment along with its integration components is depicted in Figure 17. The various levels of virtualization (if applicable) for IEAP and edges components, including the underlying hardware, are illustrated in Figure 18.

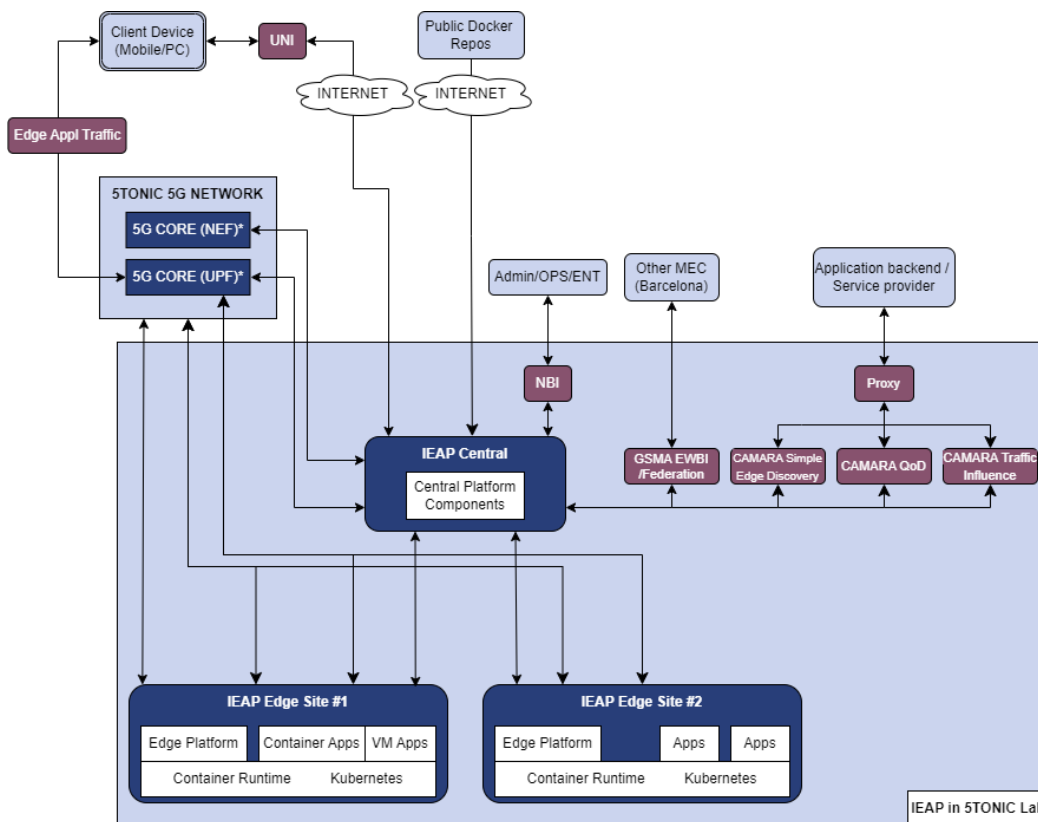


Figure 17: IEAP deployment diagram

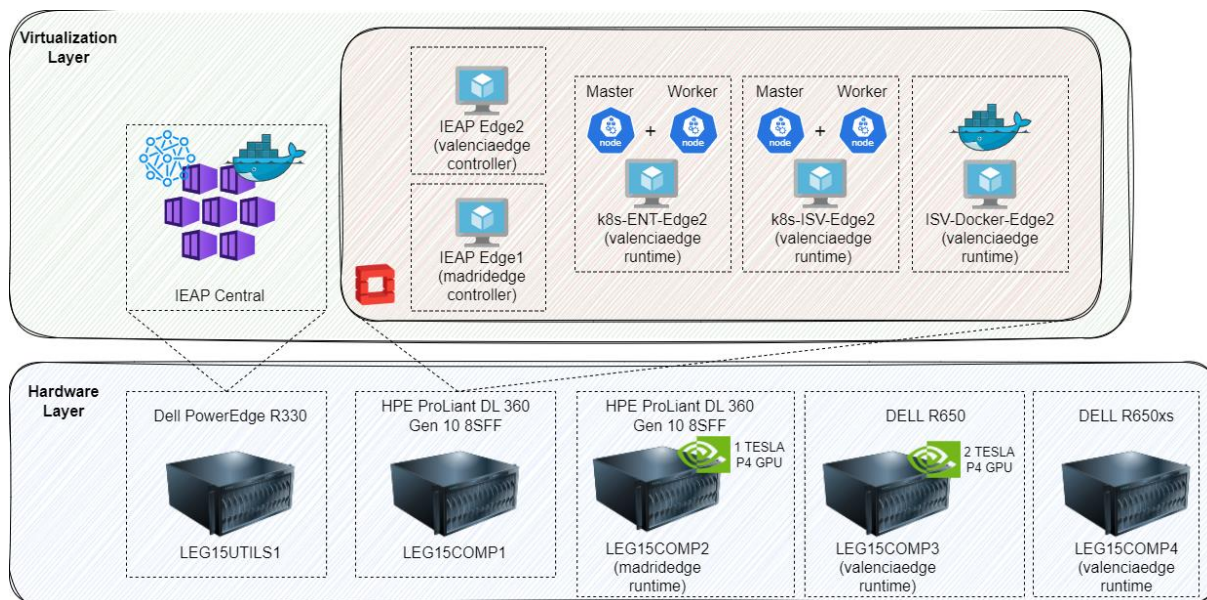


Figure 18: IEAP hardware and virtualization diagram

4.1.2 E2.1 Barcelona Edge orchestrator deployment

The Barcelona Edge infrastructure consists of two physical servers that have been in place since the project's inception. Later, a tower PC was added to support various projects from the FSTP programs

across different Open Calls. So far, it has hosted one project from Open Call 1 (Requiem) and two from Open Call 2 (6G Remix and EMSEOS).

The following images illustrate the servers and PC locations, and the various GUIs used to manage the clusters and resources. The image below specifically highlights the servers hosting the EDGE in Barcelona (Figure 19) and the Tower PC (Figure 20).



Figure 19: Servers hosting the 6GXR main cluster



Figure 20: Tower PC hosting the infrastructure used for the open calls

Figure 21 features the OpenStack GUI, which serves as the infrastructure manager for provisioning virtual nodes. It showcases a Kubernetes (K8S) cluster, including the master node and two worker nodes, one of which is equipped with a GPU.

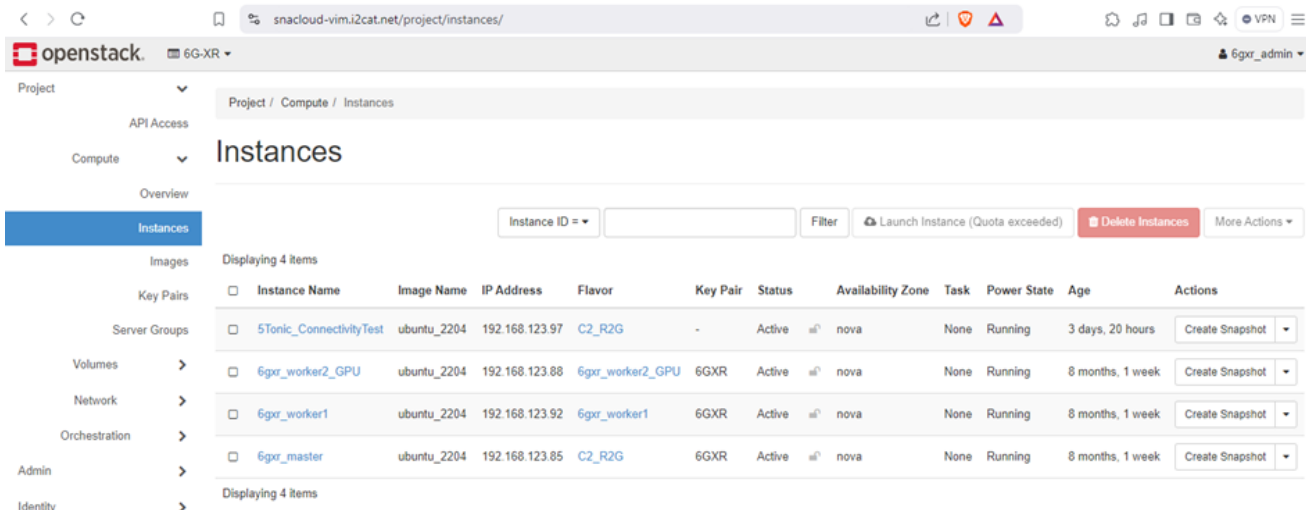


Figure 21: OpenStack hosting the Edge Infrastructure in Barcelona

View of the cluster status (Figure 22), and the workloads running on the cluster (Figure 23 and Figure 24).

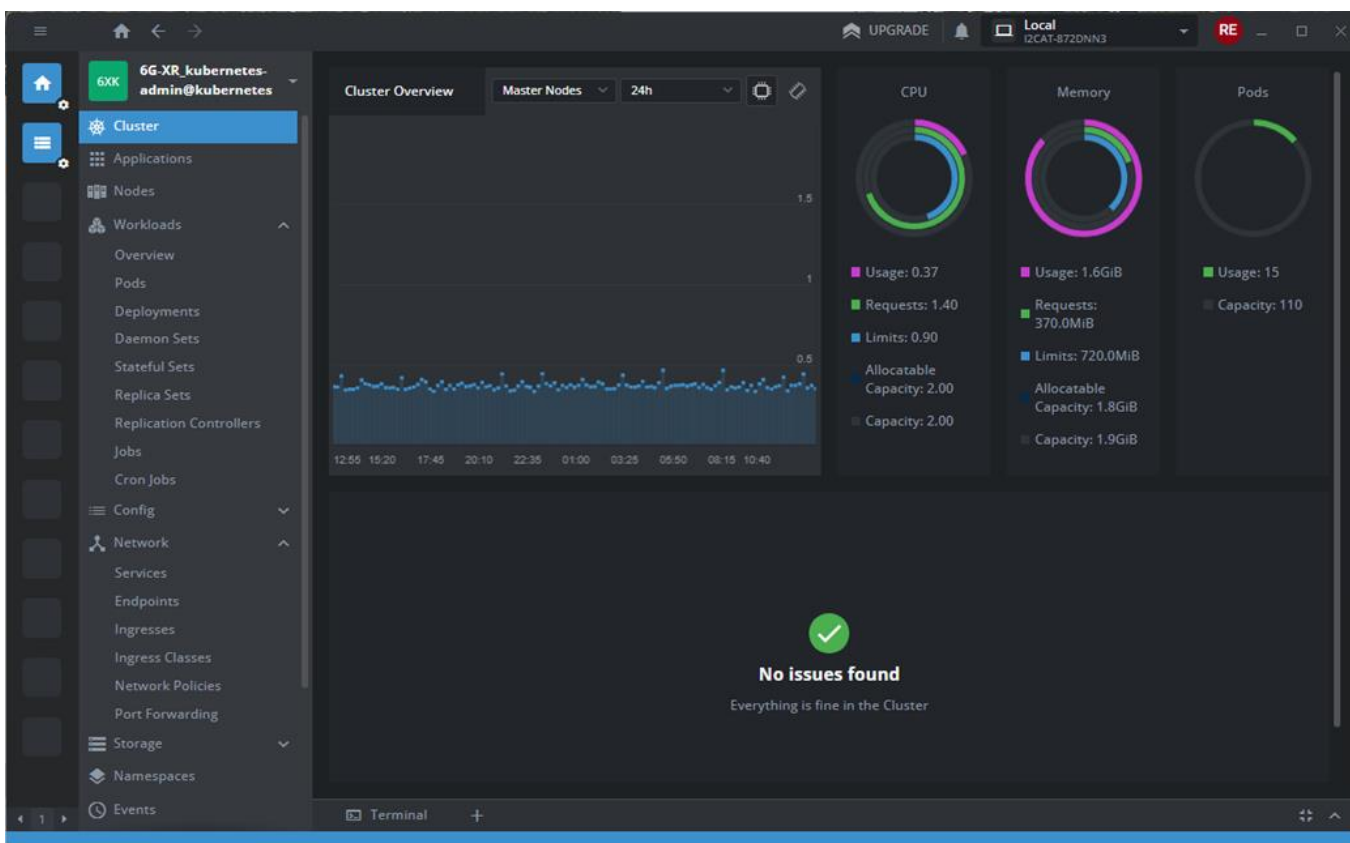


Figure 22: Computing resources

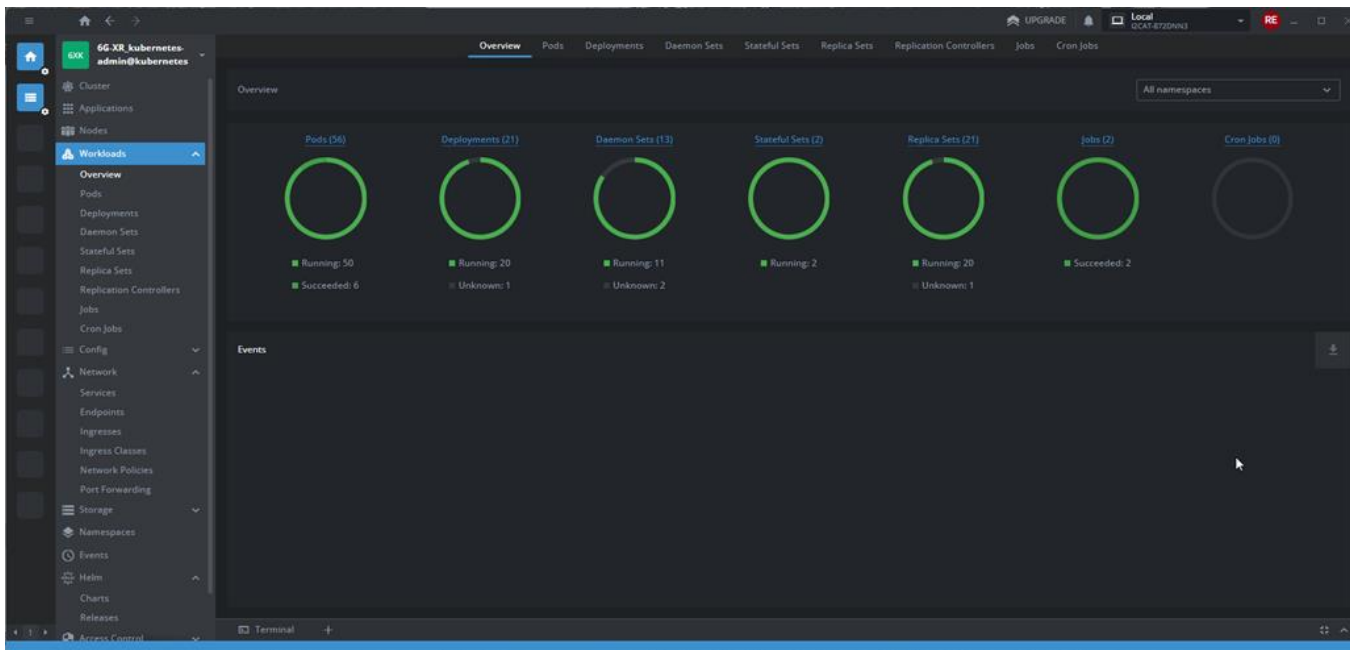


Figure 23: Overview of the services running

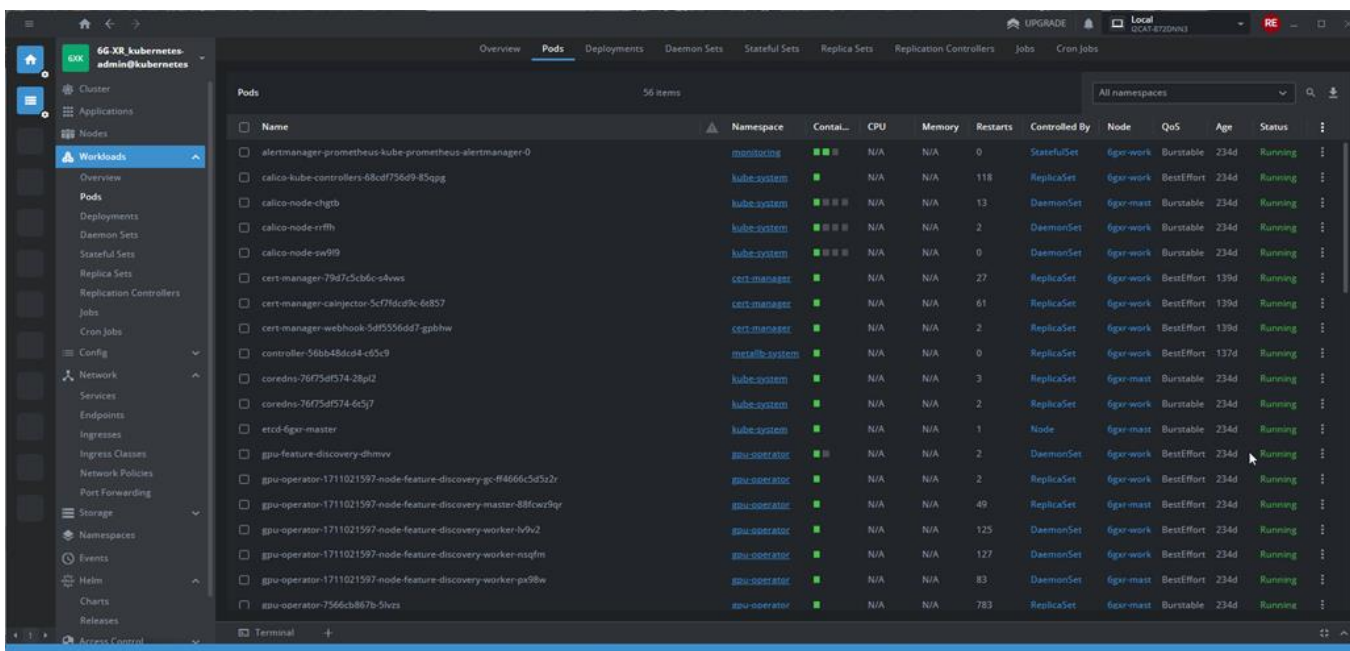


Figure 24: Current workload

4.1.3 ES.1 Edge Federation deployment

i2CAT has implemented the East-West Bound Interface (EWBI) to enable interaction with resources in the Barcelona Edge. The API is fully compliant with the reference “.yaml” file proposed by the GSMA, ensuring adherence to industry standards. The implementation has been validated using this reference file alongside tools such as Swagger.

The MEF Manager is the main component responsible for implementing the functionality defined in the EWBI API. It serves as a lightweight layer that facilitates interaction with the MEC platform underneath. The MEF Manager has been deployed on a private cloud at i2CAT, running on a virtual machine (VM). This VM is configured with an interface connected to the provider network, ensuring reliable connectivity with the rest of the infrastructure.

Since the beginning of MEF Manager development in April 2024, the code has been managed using a local GitLab instance, where all necessary deployment files are maintained. The picture below shows a capture of the Gitlab's GUI (Figure 25).

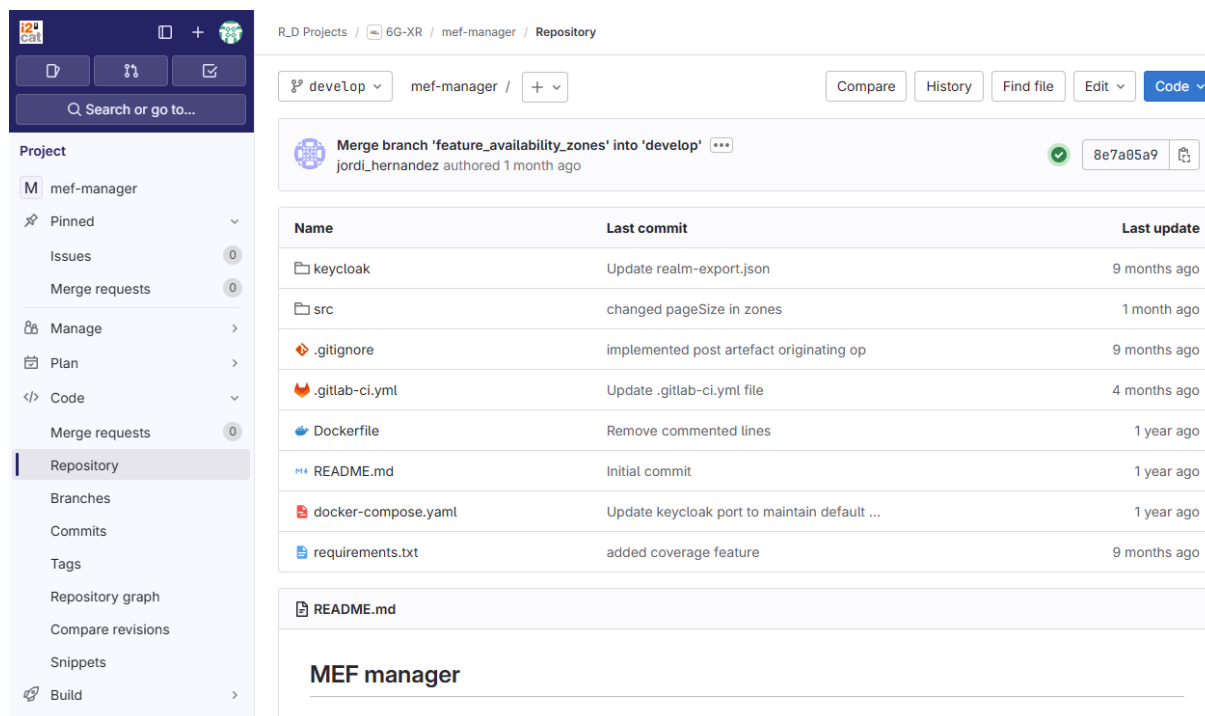


Figure 25: Private Repository for the MEF manager

To streamline the deployment process, a docker-compose.yaml file is provided. This file orchestrates the deployment of both the MEF Manager and Keycloak, with Keycloak handling user authentication and generating the required tokens for partners to invoke the API.

The MEF Manager runs on a lightweight Kubernetes (K8S) cluster. To ensure the latest version is deployed after code modifications, the simplest approach is for the developer to terminate the existing MEF pod. This prompts K8S to automatically retrieve the updated image from the registry and launch a new instance. The picture below (Figure 26) shows the pods running in the MEF manager.

Name	Namespace	Cont...	CPU	Memor	Restart	Controlled B	Node	QoS	Age	Status
coredns-76f75df574-hvjql	kube-system	Running	N/A	N/A	1	ReplicaSet	mef-mana	Burstable	405d	Running
coredns-76f75df574-qmbwg	kube-system	Running	N/A	N/A	1	ReplicaSet	mef-mana	Burstable	405d	Running
etcd-mef-manager	kube-system	Running	N/A	N/A	21	Node	mef-mana	Burstable	405d	Running
keycloak-94db5b66d-x8c6l	keycloak	Running	N/A	N/A	0	ReplicaSet	mef-mana	BestEffort	28d	Running
kube-apiserver-mef-manager	kube-system	Running	N/A	N/A	91	Node	mef-mana	Burstable	405d	Running
kube-controller-manager-mef-manag	kube-system	Running	N/A	N/A	216	Node	mef-mana	Burstable	405d	Running
kube-flannel-ds-8gfcw	kube-flannel	Running	N/A	N/A	1	DaemonSet	mef-mana	Burstable	405d	Running
kube-proxy-dnwwn	kube-system	Running	N/A	N/A	1	DaemonSet	mef-mana	BestEffort	405d	Running
kube-scheduler-mef-manager	kube-system	Running	N/A	N/A	224	Node	mef-mana	Burstable	405d	Running
mef-manager-app-5b6bfb655-mzdxl	mef-manager	Running	N/A	N/A	0	ReplicaSet	mef-mana	Burstable	28d	Running
mongodb-67d64674c-hdt2f	mongodb	Running	N/A	N/A	1	ReplicaSet	mef-mana	BestEffort	404d	Running
nfd-gc-5b987cb58f-sjg6d	node-feature-dis	Running	N/A	N/A	1	ReplicaSet	mef-mana	BestEffort	405d	Running
nfd-master-7bff75887-8ms9g	node-feature-dis	Running	N/A	N/A	1	ReplicaSet	mef-mana	BestEffort	405d	Running
nfd-worker-fqztc	node-feature-dis	Running	N/A	N/A	173	DaemonSet	mef-mana	BestEffort	405d	Running

Figure 26: Cluster hosting the MEF manager

Additionally, the repository includes a “gitlab-ci.yml” file, which automates testing and builds a new image whenever code changes are committed. The updated image is then stored in the registry, ensuring the availability of the latest version for deployment.

Once the MEF is up and running, developers can interact with it by using Swagger tool. The following screenshots (Figure 27, Figure 28, Figure 29, Figure 30, and Figure 31) highlight the basic set of functionalities implemented from the extensive API definition defined by GSMA. The rest of the functionalities defined in the EWBI are out of the scope of this project.

Method	Endpoint	Description
POST	/partner	Creates one direction federation with partner operator platform.
GET	//{federationContextId}/partner	Retrieves details about the federation context with the partner OP. The response shall provide info about the zones offered by the partner, partner OP network codes, information about edge discovery and LCM service etc.
PATCH	//{federationContextId}/partner	API used by the Originating OP towards the partner OP, to update the parameters associated to the existing federation
DELETE	//{federationContextId}/partner	Remove existing federation with the partner OP

Figure 27: Federation Management methods

AvailabilityZoneInfoSynchronization		
POST	<code>/{{federationContextId}}/zones</code>	Originating OP informs partner OP that it is willing to access the specified zones and partner OP shall reserve compute and network resources for these zones.
DELETE	<code>/{{federationContextId}}/zones/{zoneId}</code>	Assert usage of a partner OP zone. Originating OP informs partner OP that it will no longer access the specified zone.
GET	<code>/{{federationContextId}}/zones/{zoneId}</code>	Retrieves details about the computation and network resources that partner OP has reserved for this zone.

Figure 28: Availability Zone Info Synch

ArtefactManagement		
POST	<code>/{{federationContextId}}/artefact</code>	Uploads application artefact on partner OP. Artefact is a zip file containing scripts and/or packaging files like Terraform or Helm which are required to create an instance of an application.
GET	<code>/{{federationContextId}}/artefact/{artefactId}</code>	Retrieves details about an artefact.
DELETE	<code>/{{federationContextId}}/artefact/{artefactId}</code>	Removes an artefact from partner OP.
POST	<code>/{{federationContextId}}/files</code>	Uploads an image file. Originating OP uses this api to onboard an application image to partner OP.
DELETE	<code>/{{federationContextId}}/files/{fileId}</code>	Removes an image file from partner OP.
GET	<code>/{{federationContextId}}/files/{fileId}</code>	View an image file from partner OP.

Figure 29: Artefact Management

ApplicationOnboardingManagement		
POST	<code>/{federationContextId}/application/onboarding</code>	Submits an application details to a partner OP. Based on the details provided, partner OP shall do bookkeeping, resource validation and other pre-deployment operations.
DELETE	<code>/{federationContextId}/application/onboarding/app/{appId}</code>	Deboards the application from any zones, if any, and deletes the App.
PATCH	<code>/{federationContextId}/application/onboarding/app/{appId}</code>	Updates partner OP about changes in application compute resource requirements, QoS Profile, associated descriptor or change in associated components
GET	<code>/{federationContextId}/application/onboarding/app/{appId}</code>	Retrieves application details from partner OP
DELETE	<code>/{federationContextId}/application/onboarding/app/{appId}/zone/{zoneId}</code>	Deboards an application from partner OP zones
POST	<code>/{federationContextId}/application/onboarding/app/{appId}/additionalZones</code>	Onboards an existing application to a new zone within partner OP.
POST	<code>/{federationContextId}/application/onboarding/app/{appId}/zoneForbid</code>	Forbid/allow application instantiation on a partner zone

Figure 30: Application onboarding

ApplicationDeploymentManagement		
POST	<code>/{federationContextId}/application/lcm</code>	Instantiates an application on a partner OP zone.
GET	<code>/{federationContextId}/application/lcm/app/{appId}/instance/{appInstanceId}/zone/{zoneId}</code>	Retrieves an application instance details from partner OP.
DELETE	<code>/{federationContextId}/application/lcm/app/{appId}/instance/{appInstanceId}/zone/{zoneId}</code>	Terminate an application instance on a partner OP zone.
GET	<code>/{federationContextId}/application/lcm/app/{appId}/appProvider/{appProviderId}</code>	Retrieves all application instance of partner OP

Figure 31: Application Deployment Management

4.1.4 Network Exposure Function (NEF) APIs deployment

The deployment of all NEF APIs (*E6.1 Service Parameter API*, *E6.2 UE Location API*, *E6.3 QoS Session API* and *E6.4 Data Collection API*) was done on the same component in the South Node. The idea of the NEF is to expose in a secure way some useful capabilities from the 5G network to the Application Functions (AFs). In this way, the critical functions of the 5G Core from direct access by third parties (3PPs). Besides, the features that are exposed to the AFs must be controlled to make sure that the requests made by a single AF will not exhaust the resources for the rest of the network users. The NEF component at 5Tonic is a server running containers that have an IP address in the subnet

10.3.3.0/24, in a range separated from the 5G Core. The NEF can reach the 5G Core functions (the well-known AMF, SMF, AuSF, UDM, UDR, etc.) via layer 3 routing. Also, any AF and the CAMARA functions running on the Edge platform managed by Capgemini can reach the NEF, so they can make the calls for requesting the adaptation of the network to the needs of the services. The deployed setup is illustrated in Figure 32.

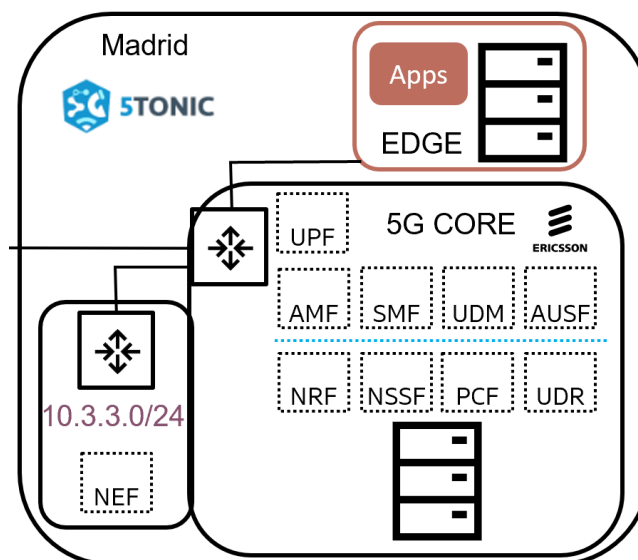


Figure 32: NEF setup in the South Node

4.2 SOUTH NODE CONTROL PLANE ENABLERS

This section details the deployment and setup of the enabler needed for UC3 *Control plane innovations* in the South Node (see D1.1 [2]).

4.2.1 E9.1 IMS Data Channel Server deployment

The IMS Data Channel Server (DCS) used in the South Node was originally deployed on an Azure environment located in Paris, which was the closest location to 5Tonic at the moment of the deployment, in May 2024. The IMS DCS VMs were deployed in a few days and then the establishment of the VPN was completed by June. Afterwards, during June 2024, an Azure platform was made available in Madrid, so the IMS DCS VMs were migrated and the VPN endpoint was moved to the new platform.

As shown in Figure 33, the full IMS Data Channel solution requires the legacy IMS Core functions (depicted on the left side) and the MATSUKO signaling and media servers (depicted on the right side). In the scope of WP2, the IMS Core functions were deployed in 5Tonic. The functions from Matsuko were deployed in the scope of WP3, so they are out of scope of this document.

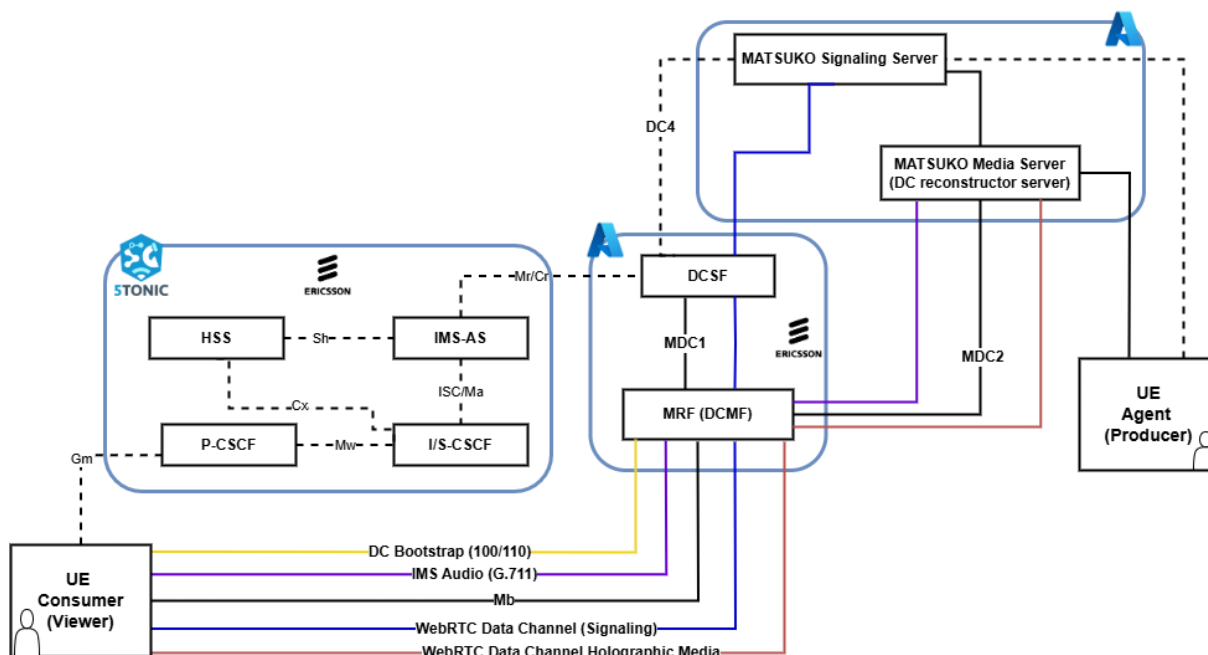


Figure 33: IMS DC deployment in the South Node

The IMS Core functions IMS-AS, P-CSCF and I/S-CSCF were deployed in 5Tonic between March and September 2024. In November 2024, the connectivity among those VMs and IMS DCS VMs was validated.

The Home Subscriber Server (HSS) function was delayed because it was dependent on the deployment from scratch of a new full 5G Core at 5Tonic including the prior installation of new servers, new switches and cabling. The HSS deployment was completed in the beginning of April 2025.

4.3 NORTH NODE 3D DIGITAL TWIN ENABLERS

This section describes the deployment and setup of the enablers required for UC4 *Collaborative 3D Digital Twin-like Environment* in the North Node (see D1.1 [2]).

4.3.1 E3.1 North Node adapter deployment

NNA is a software component that plays a crucial role in the setup and management of network slices and KPI measurement jobs within the 5GTN. It acts as a communication bridge between the North Node Web Portal and the underlying 5GTN resources and management technologies. NNA is deployed at the 5GTN on a Linux-based (Ubuntu 24.04) virtual server under Supervisor, a process management software tool. The deployment of NNA and its related components was successfully completed in M24. The deployment of NNA including related network components is depicted in Figure 34.

NNA provides North Node Web Portal a REST API through which the web portal can send an experiment specification document, i.e. Network Slice Template (NST), to initiate an experiment. Through the API the web portal can also monitor the state of the running experiments or force them to end. The API that NNA provides to the North Node Web Portal is described in detail in D2.2 [1] and was the first piece of functionality deployed in the project in M18.

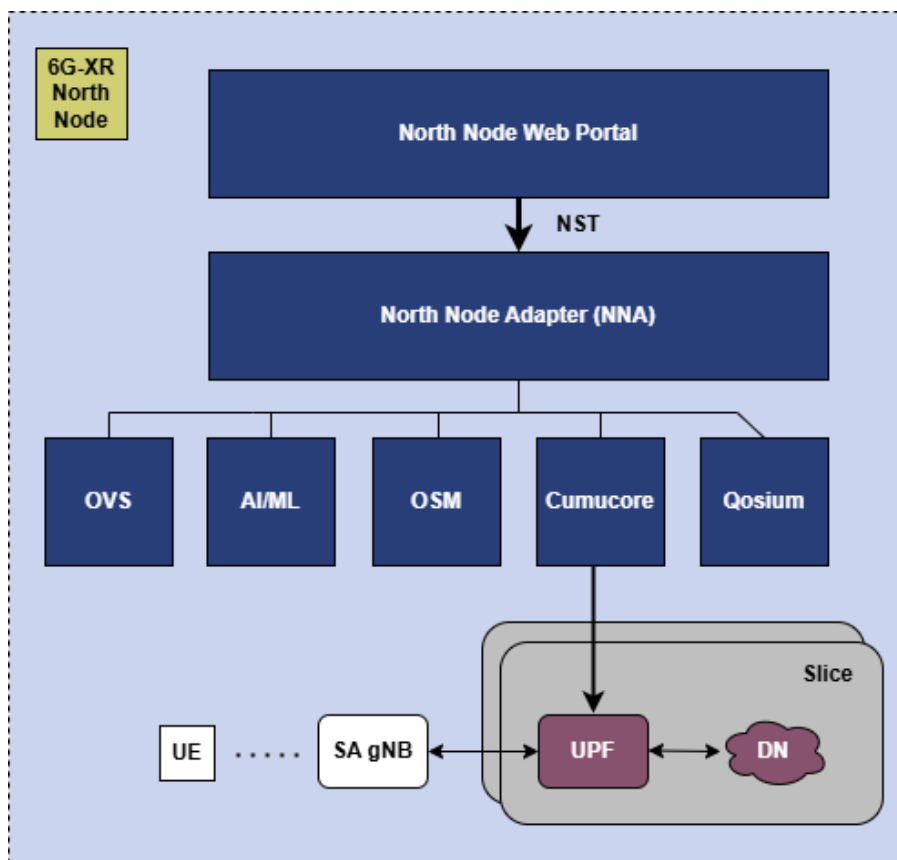


Figure 34: NNA deployment with related network components

NNA oversees the lifecycle of KPI measurement jobs by using Qosium Storage, a server and database software tool providing REST API for managing measurements and retrieving collected KPI data. Qosium Storage and its REST API are used by NNA to get DL/UL KPI values per slice in one second intervals. NNA deployment with the Qosium component was completed in M24.

Cumucore is the core network solution used in the North Node use cases, and NNA uses it for creating and deleting network slices through a REST API. Cumucore is configured for UE subscriptions and UPFs, which are deployed on VMs with N3 and N6 interfaces connected through Open vSwitch (OvS). NNA deployment and validation of interworking with Cumucore was completed in M24.

To instantiate virtual machines at the edge and deploy software on them as requested by the user of the North Node web portal, NNA uses the Open-Source MANO (OSM) component. OSM provides a REST API for managing VMs by OpenStack. This REST API is called by NNA, once at the beginning of an experiment to initiate VMs and once at the end of the experiment to tear down the OpenStack VMs. Support for the OSM interface in NNA was developed relatively late in the project (in M26) as this feature did not have high priority in the project.

NNA relays per-slice DL/UL KPI measurement data from Qosium to the AI/ML component, which in turn returns optimized resource allocation (i.e. bandwidth) between the used network slices. The AI/ML component provides NNA with a REST API. The support for AI/ML communication in NNA was completed in M22.

Finally, NNA enforces dynamic resource allocation obtained from AI/ML by communicating with OvS. NNA sends JSON-RPC messages over a TCP socket to OvS to configure ingress policy rates and burst

values to the N3 and N6 interfaces connected to the slice-specific UPFs. The feature of enforcing network slice resource allocation at the switch was tested and deployed in M25 of the project.

4.3.2 E3.2 3D Digital Twin deployment

As a 3D Digital Twin use case, the service of 5G slices for VR glasses XR Fab Lab App is adaptably controlled with AI/ML Policy Network in the edge service. Its deployments are structured in Figure 35 as follows,

- 5G slice service is delivered by CumuCore 5G with slice capable 5G modems.
- Qosium measurement probe is measuring QoS on the traffics of XR Fab Lab App on the user VR glasses device side, and storing the data into Qosium Storage in the edge.
- The service level of 5G slices in CumuCore is managed by Policy Network's AI/ML by analyzing the QoS data on Qosium measurement system via NNA's Rest APIs.

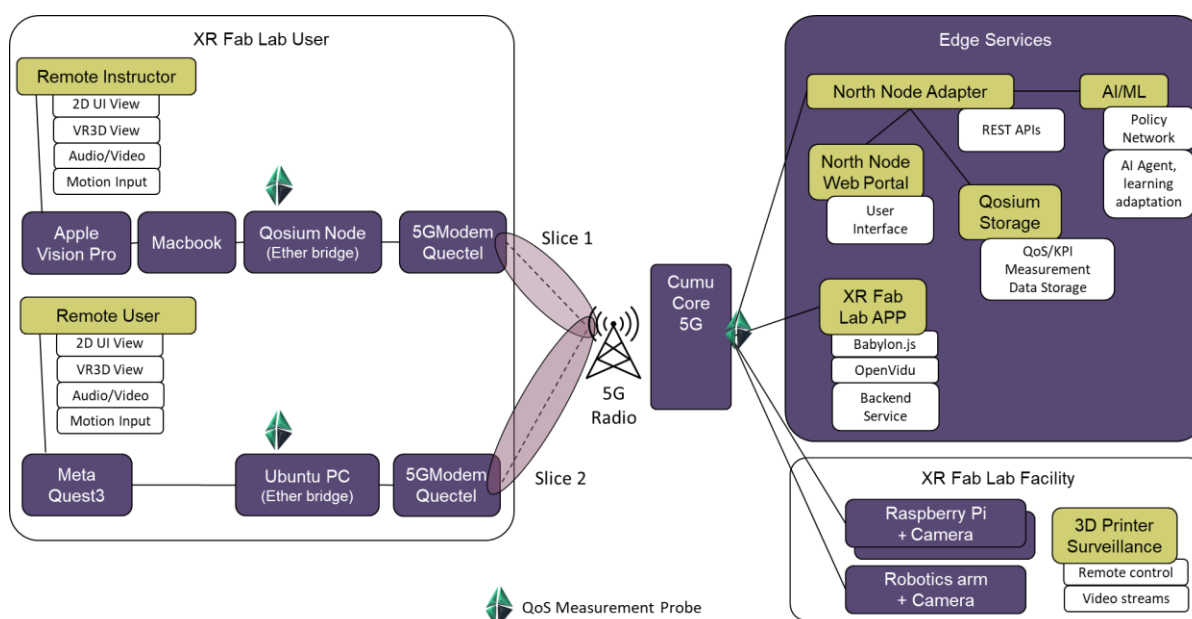
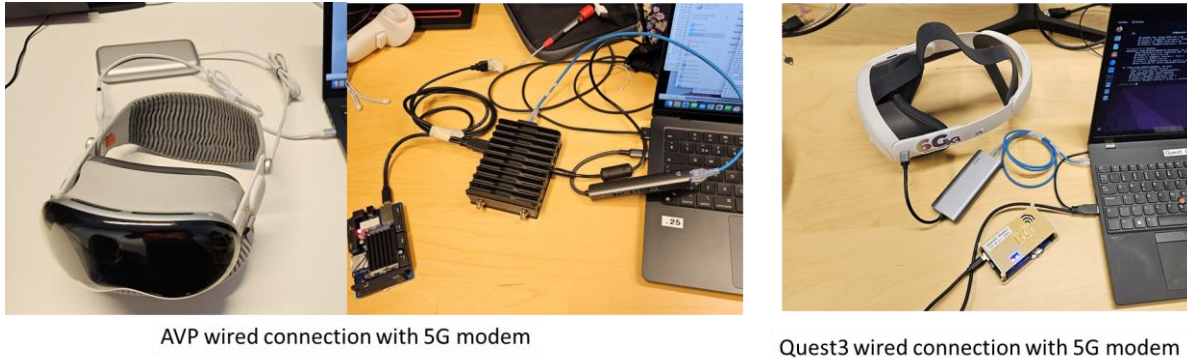


Figure 35: 3D Digital Twin XR Fab Lab App deployment in the local 5G edge system

4.3.2.1 5G System Connectivity:

Due to the lack of built-in 5G radio connectivity in VR glasses, as shown in Figure 36, a Quectel 5G modem is integrated via a wired connection. The Meta Quest 3 connects to an Ubuntu laptop through a USB-to-Ethernet adapter, with Ubuntu routing data from the 5G modem to the VR glasses while also performing Qosium Probe QoS/KPI measurements. Additionally, the Apple Vision Pro (APV) is connected to a MacBook using its internet-sharing function. Since the Quectel 5G modem driver requires a Linux platform, a Qosium Node is positioned between the MacBook and the Quectel modem to provide both 5G connectivity and Qosium probe measurement functionality.



AVP wired connection with 5G modem

Quest3 wired connection with 5G modem

Figure 36: 5G modem connection with VR glasses

4.3.2.2 XR Fab Lab Application:

The XR Fab Lab application is hosted on the local edge server and comprises Babylon.js and OpenVidu along with its backend functions. As implementation shown in Figure 37, Babylon.js is used to deploy the 3D VR environment on the WebXR platform for users' VR glasses, supporting a variety of XR gadgets within its open platform. The OpenVidu platform facilitates sessions for voice, video, and motion transmission for the VR glasses, and also supports video streaming for 3D printer surveillance.

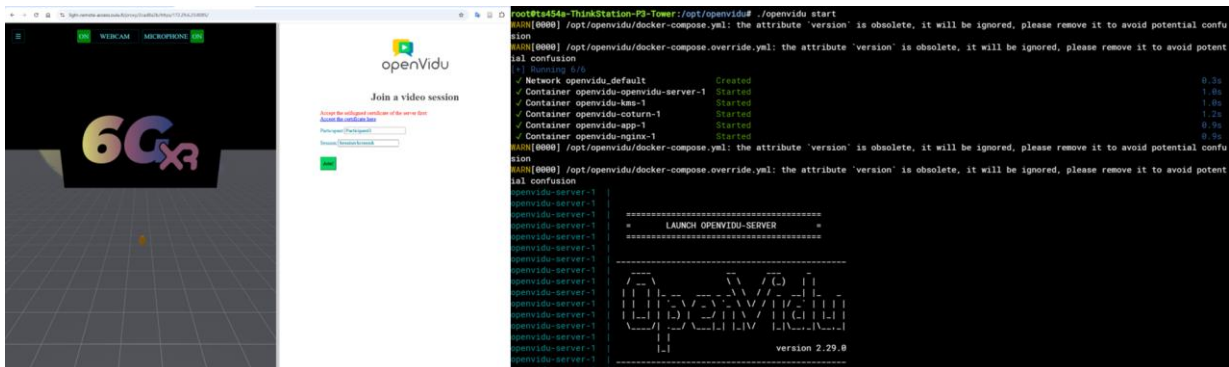


Figure 37: XR Fab Lab of Babylon.js and OpenVidu connectivity implementation

4.3.2.3 Network QoS/KPI Measurement:

For input to the AI-based resource optimization Policy Network, network traffic is measured using the Qosium platform, including its Probe and Storage components. The AI agent analyzes the traffic data stored and optimizes slicing and resource management within the Cumucore 5G system. The traffic consists of multi-modal communication, including voice, motion, video streaming, and VR scene environments. To support AI learning and analytics validation, a test procedure is established in Figure 38: (1) launch the Babylon.js VR scene, (2) start video streaming from the 3D printer surveillance, and (3) trigger VR scene transitions to generate diverse traffic patterns.

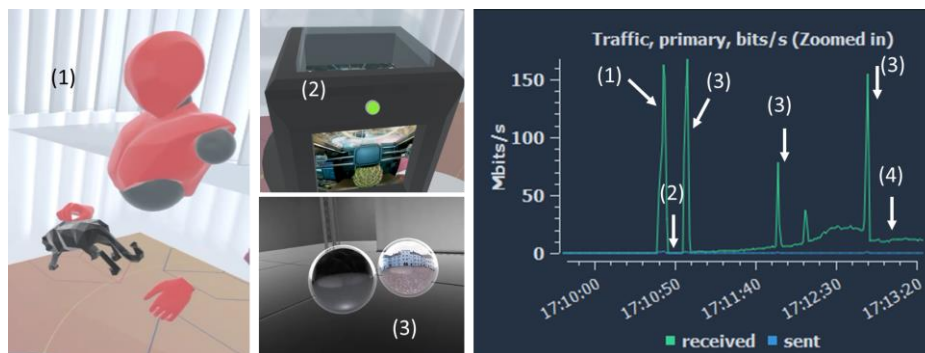


Figure 38: XR Fab Lab App procedure for Qosium measurement validation

4.3.3 E3.4 Resource Optimization Deployment

The AI-based resource optimization solution has been deployed as a self-contained virtual machine (VM), ensuring portability, ease of integration, and environment consistency. The deployment includes the pre-trained AI model, the Flask-based REST API, and the necessary runtime environment, all packaged within the VM to simplify distribution and integration with other system components.

The VM runs a Linux-based operating system and hosts a Python-based application stack. The core components of the deployment include:

- **AI Model:** A pre-trained policy network implemented in PyTorch, responsible for making real-time decisions on network resource allocation.
- **REST API:** Implemented using Flask, exposing endpoints to receive KPIs and return optimized resource allocation for multiple network slices.
- **Waitress WSGI Server:** Deployed as the production-grade server to run the Flask application, providing reliable and scalable API access.

The resource optimization VM is designed to interact directly with the North Node Adapter (NNA), which serves as the primary source of real-time KPIs. The integration between the NNA and the AI model is facilitated through RESTful communication. The NNA collects network measurement data and operational metrics and forwards them in JSON format to the resource optimization API running within the VM. Upon receiving the KPIs, the AI model processes the data, performs inference, and responds with optimized resource allocation decisions for each network slice.

This deployment architecture ensures that:

- The AI model operates independently but remains tightly coupled with the NNA for data-driven decision-making.
- Updates to the AI model or API can be managed centrally within the VM without affecting external systems.
- The solution can be replicated or scaled easily by deploying additional VM instances in other parts of the network.

This modular and isolated deployment approach makes the resource optimization engine a reliable plug-and-play component within larger network management architectures. It can be seamlessly

integrated into existing infrastructure and provides a scalable path for future updates or enhancements, such as online learning.

The resource optimization deployment was finalized in M25, following the successful validation of the AI model in simulated and real-time environments. The trained policy network is based on live data from our network provided by the NNA: it was initially pre-trained offline on a comprehensive collection of simulated KPI traces, spanning a wide variety of throughputs, latencies, jitters, packet-loss rates, and signal strength generated by the 5G test network. After this offline phase, an online fine-tuning step was done as soon as the VM was deployed and connected to the live NNA feed, adapting the policy to actual traffic patterns. Figure 39 below shows learning-curve snapshots to demonstrate the convergence and generalization across different KPI scenarios and the expected behavior of the AI model.

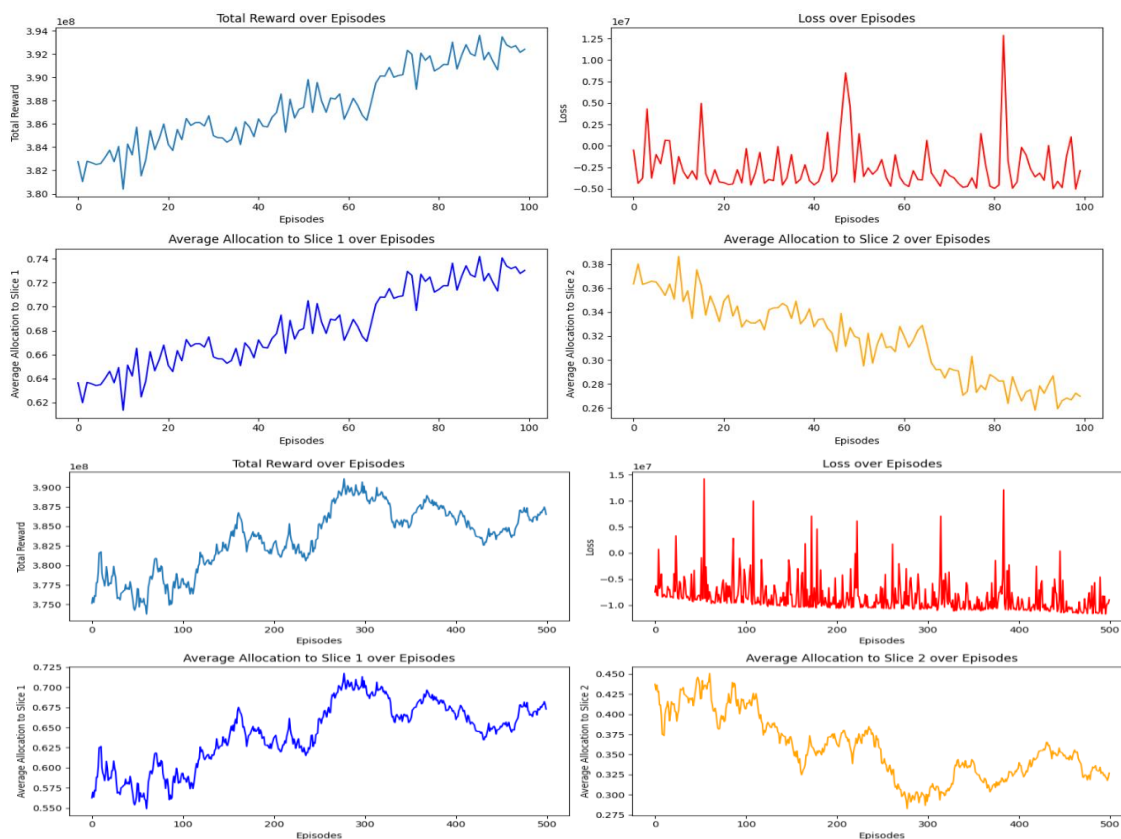


Figure 39: Policy network training and allocation trend

The integration and interaction between the AI model, API, and NNA have been designed to support both operational efficiency and system robustness in dynamic network environments.

4.3.4 E7.1 Cumucore Slice Management API deployment

Cumucore is deployed on a bare-metal server at the 5GTN facility and is configured to accept REST API calls from external hosts at port 9000. Cumucore was deployed early in the project in M21. Cumucore is accessed by NNA which requires that both reside in the same 5G test network and have no firewall in between.

Cumucore is used to configure UEs (subscriptions) and UPFs with DNs per slice as depicted in Figure 40. The information how slices are created and configured for experiments comes from NNA, which

takes advantage of the slice creation part of the CumuCore REST API. Controlling the slice parameters such as throughput is not managed via CumuCore as it has no support for optimizing slice resources on-the-fly.

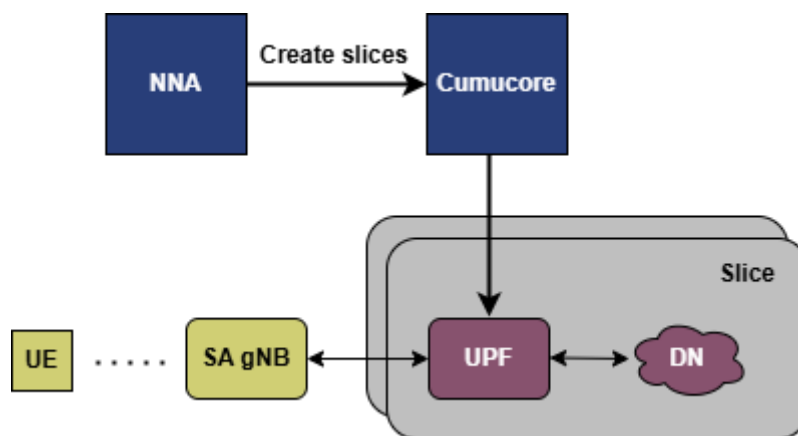


Figure 40: The role of NNA and CumuCore in slice management

Adjusting slice parameters (i.e. downlink/uplink throughput) on-the-fly is not done using the CumuCore REST API as in the development phase it was discovered that it did not work. An alternative method was therefore devised that involves adjusting slice DL/UL throughput at Open vSwitch level by setting ingress policy rates on N3 and N6 interfaces. This involves NNA sending JSON-RPC formatted messages to the switch whenever throughputs must be adjusted.

4.4 NORTH NODE ENERGY FRAMEWORK ENABLERS

This section explains the deployment of the enablers required for UC5 *Energy Measurement Framework for Energy Sustainability* in the North Node (see D1.1 [2] for UC description).

4.4.1 E3.3 Energy Management deployment

The sustainability experimentation framework includes bi-directional, multifunctional PV-hybrid systems on the production side and 5GTN components such as OAIBOX, USRPs, Quectel Modems as UE, and commercial gNBs on the consumption side. Within the energy measurement framework, the central controller integrates three forecasting APIs: energy weather forecasting for the next 66 hours, ELSPOT electricity spot pricing for the next 24 hours, and real-time CO₂ estimates from Fingrid's Open data API. These APIs are used to orchestrate, manage, and support decision making regarding power saving measures for the gNB sites. Figure 41 illustrates the integration of the developed features within the energy measurement framework. The three core functionalities include orchestration, data exchange, and forecasting APIs are managed by the central controller. On the UOULU side, this functionality is handled by the Nokia OpenEdge server, while on the VTT side, the Venus GX serves this role. The Venus GX also facilitates RAN consumption forecasting, PV yield prediction based on historical data, and dimensioning of the storage system to optimize energy utilization.

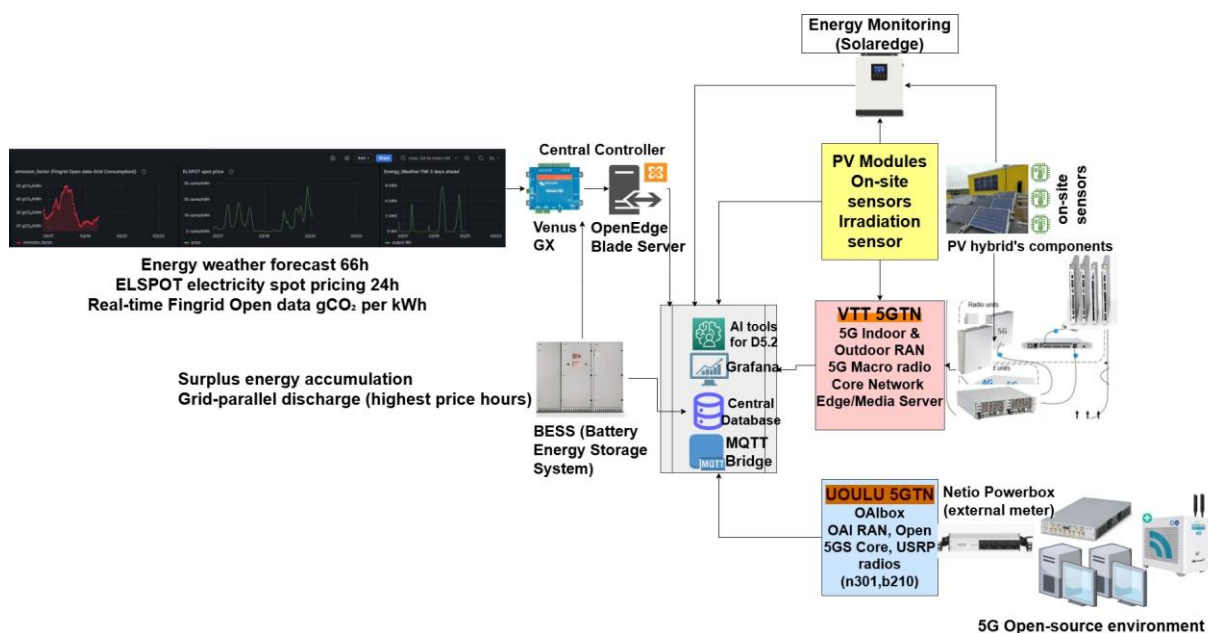


Figure 41: North Node Energy Measurement Framework

The energy measurement framework has been implemented using Python programming language, with the central controller running on an Open Edge server hosted on a Linux-based virtual machine. Mosquitto is used as the MQTT broker to enable data exchange for external meters, forecasting APIs, and on-site sensor data. MySQL serves as the central database for collecting production and consumption data. Finally, Grafana is used for real-time visualization of all the collected data.

Visual Studio Code is used to run the scripts and connect to the host server. The Python scripts perform the following tasks:

1. Fetch energy weather forecast data from FMI (66 hours ahead, updated every 3 hours), publish relevant data via the MQTT bridge, store it in MySQL, and visualize it in Grafana in real time.
2. Fetch electricity spot pricing data from ELSPOT (24 hours ahead), publish it using the MQTT bridge, store it in MySQL, and visualize it in Grafana.
3. Fetch real-time CO₂ estimates from Fingrid's API (updated every 3 minutes for gCO₂ per kWh), publish it using the MQTT bridge, store it in MySQL, and visualize it in Grafana.
4. Collect external meter data, on-site sensor data, and PV-hybrid system data (including BESS) and store it using Venus GX at VTT and the central database at UOULU.

4.4.2 E8.1 OAIBOX deployment

As described in D2.2 [1], the OAIBOX allows different RAN configurations (tested non-dynamically) to observe the effect on energy consumption. These configurations include bandwidth, TDD frame, MCS and MIMO/SISO. Measurements were repeated with two different Radio Units: USRP n310 and USRP b210. Four potential radio configuration changes to save energy were tested in North Node UOULU 5G indoor test environment using a single OAIBOX MAX device UE applied in the tests is a combination of 5G Quectel modem and Windows PC. The tested radio configuration changes were:

- Restricting the used bandwidth between 20 MHz and 100 MHz.
- Restricting the modulation constellation between QPSK, 16QAM, 64QAM and 256 QAM(max).

- TDD slot configuration selection, and DL/UL MIMO mode selection.

The purpose of the experiments was to observe if these radio configuration changes conserve energy in North Node UOULU O-RAN setup with the energy measurement framework. For example, by reducing the network capacity during low data traffic conditions such as nighttime, and using maximum bandwidth and MCS when high energy availability is predicted using energy weather forecasting.

These radio configuration changes and technical details about OAIBOX MAX device are described in the project deliverable D5.1 [10]. The power consumption was measured separately for the OAIBOX MAX device, SDR device (USRP B210 or N310) and 5G Quectel modem as shown in Figure 42. Monitoring the energy consumption separately for USRP provides energy consumption information of the radio unit of gNB. The OAIBOX MAX device contains the 5G core and other gNB functions (excluding radio unit). 5G Quectel modem is a communication part of a UE also including a computing device (Raspberry Pi 5). Netio Power Box was used as an AC power meter for measuring separately current (I), voltage (V), and true power factor (TPF). These values were used to calculate power (P), $P = I \cdot U \cdot TPF$, with the accuracy of 0.1 W. These energy-related parameter values are delivered to Grafana dashboard for visualizing the results and for data analytics.



Figure 42: OAIBOX power consumption enabled by North Node energy measurement framework

In most cases, radio configurations that provide higher bitrates consume more power during no traffic periods and during TCP speed tests. It seems that restricting the used bandwidth is the best option to conserve energy compared to other tested radio configuration changes. Notable power consumption reduction has been seen in all monitored devices (OAIBOX MAX, USRP B210, n310 and Quectel modem) when decreasing the bandwidth. This shows that adaptive tuning of bandwidth based on traffic needs such as network slicing is an effective energy saving method. However, when there is high traffic in the network then the higher bandwidth should be used because it provides remarkably higher amount of received data per energy unit. All the experimentation results, log files including test case validation and KPIs validation will be reported in D5.2.

5 ENABLERS INTEGRATION VALIDATION PHASE

This chapter reports the tests done to validate that the enablers interact well with each other and to provide the evidence that the enablers are ready for the project UCs.

5.1 SOUTH NODE USER PLANE ENABLERS

This section provides evidence of the integration validation of the enablers for UC1 *Resolution Adaptation or Quality on Demand* and UC2 *Routing to the Best Edge* in the South Node (see D1.1 [2] for UCs description).

5.1.1 Edge Federation validation

This section outlines the various tests conducted to successfully integrate the IEAP and MEF through the EWBI interface implemented at both sides in 5Tonic-Madrid and i2CAT-Barcelona respectively. The primary objective of this demonstration is to manage a third-party edge from the IEAP while simultaneously testing and validating the federation functionalities of both IEAP and MEF.

All tests were carried out through the IEAP GUI that allows Cloud Service Providers (CSPs) or customers to interact with the orchestrator. Another important consideration is that a middleware proxy has been developed and deployed to integrate IEAP and MEF. Further details on the proxy's role in each communication are illustrated in the figures presented in the following subsections.

5.1.1.1 Federation Creation

First, a new federation must be created through the IEAP GUI with the appropriate configuration parameters, including the operator, federation identifier, authentication endpoint, and the EWBI endpoint of the designated partner. In this case, the EWBI endpoint corresponds to the proxy, which subsequently forwards requests to the actual MEF endpoint.

Once the new federation is created, the IEAP GUI displays its status as *planned* (Figure 43). This action then triggers the acceptance procedure, as illustrated in Figure 44, which in turn invokes the following API method:

- POST /partner

The proxy logs capturing the MEF response are shown in Figure 45. Upon completion of the workflow, the federation status transitions from *planned* to *federated*, as depicted in Figure 46.

Operator	Country	Federation ID	Status	Last Update
barcelonaoperator	ES	barcelona2cat	planned	Apr 3, 2025

Figure 43: Planned federation from IEAP GUI

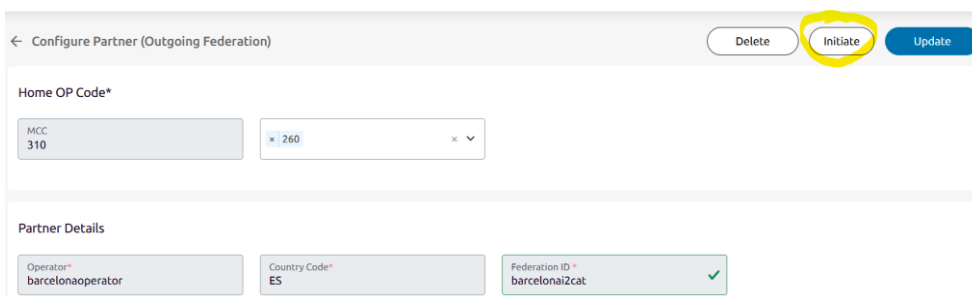


Figure 44: Accept Federation from IEAP GUI

```

2025-04-03 11:14:00,959 - INFO - HTTP Request: POST http://192.168.123.46:30989/operatorPlatform/federation/v1/partner "HTTP/1.1 200 OK"
2025-04-03 11:14:00,961 - INFO - Response: {'edgeDiscoveryServiceEndPoint': {}, 'federationContextId': '67ee6d786d87bc27ec751943', 'lcmServiceEndPoint': {}, 'offeredAvailabilityZones': [{'geographyDetails': 'omega_lab', 'geolocation': '41.3880,2.1150', 'zoneId': 'Omega'}], 'partnerOPCountryCode': 'ES', 'partnerOPFederationId': 'izcat', 'partnerOPFixedNetworkCodes': ['34'], 'partnerOPMobileNetworkCodes': {'mcc': '001', 'mncs': ['01']}, 'platformCaps': []]}
INFO: 10.15.125.10:41110 - "POST /operatorPlatform/federation/v1/partner HTTP/1.1" 200 OK
    
```

Figure 45: Proxy logs for Federation Acceptance

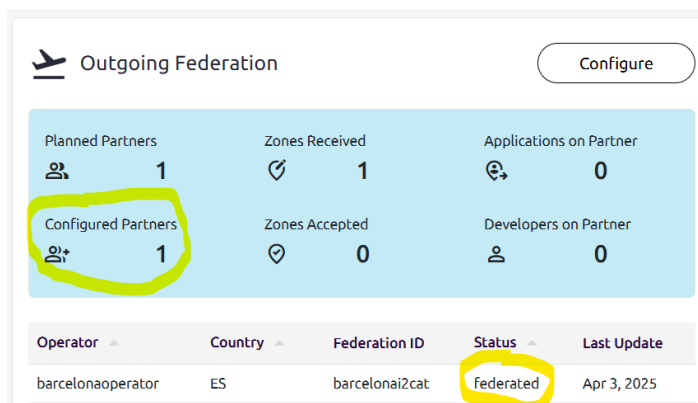


Figure 46: Accepted Federation from IEAP GUI

5.1.1.2 Zone Acceptance

Once a new federation has been configured, the next step is to accept the partner zone. Additionally, the partner will reserve compute and network resources for that specific zone. This process is executed using the following API method:

- POST /{federationContextId}/zones

When selecting the *Accept* option for the offered zone *Omega* (as shown in Figure 47), the previously introduced method is invoked. More specifically, the background processes associated with this action are illustrated in Figure 48. For further clarification, the corresponding proxy logs are provided in Figure 49. Finally, the IEAP GUI displays the zone as accepted, as depicted in Figure 50.

3. Uploading it to IEAP GUI (Figure 51)

Once the artifact is prepared, the application onboarding process is initiated using the following API methods:

- POST `/{federationContextId}/artefact`
- POST `/{federationContextId}/application/onboarding`

Onboarding the application through IEAP GUI (Figure 52) implies calling these methods sequentially. The first method uploads the artefact, while the second submits the application details to the federated operator platform. Moreover, HTTP request/response workflow is depicted in Figure 53, proxy logs are shown as well in Figure 54 for better clarity, and how the successful onboarding is reflected in IEAP GUI is illustrated in Figure 55.

← Create Artifact Clear **Create**

Artifact

Artifact Name* ✓

Artifact Version* ✓

Virt Type* ✓

Descriptor Type* ✓

Add/View user description (optional)

Compose File *

Elegir archivo

Figure 51: Artefact Uploading from IEAP GUI

← Application Onboarding Reset **Create**

Metadata

Name* ✓

Version* ✓

Access token* ✓

Add/View User Description

Application Modelling

Helm Chart Docker Compose ComponentSpec

Artifact File

Artifact ID*

QOS Profile

+ Add QOS Profile

Availability Zones

Zone* ✓ x

Figure 52: Application Onboarding from IEAP GUI

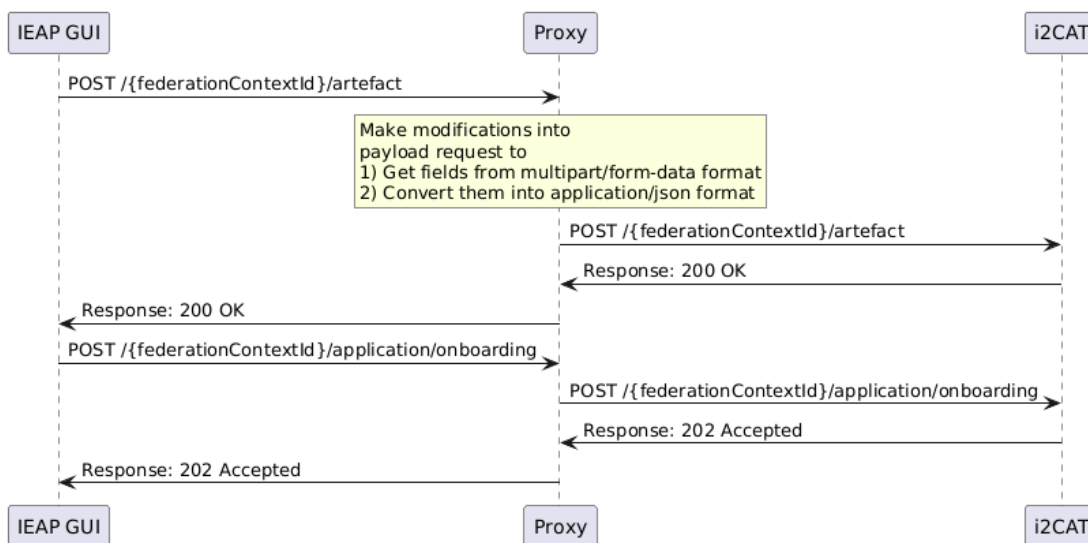


Figure 53: Application Onboarding Workflow

```

2025-04-01 10:15:29,413 - INFO - HTTP Request: POST http://192.168.123.46:30989/operatorplatform/federation/v1/67d304fd6f6cae3a48f37317/artefact "HTTP/1.1 200 OK"
2025-04-01 10:15:29,415 - INFO - Response: Artefact uploaded successfully
INFO: 10.15.125.10:6479 - "POST /operatorplatform/federation/v1/67d304fd6f6cae3a48f37317/artefact HTTP/1.1" 200 OK
2025-04-01 10:15:29,861 - INFO - HTTP Request: POST http://192.168.123.46:30989/operatorplatform/federation/v1/67d304fd6f6cae3a48f37317/application/onboarding "HTTP/1.1 202 ACCEPTED"
2025-04-01 10:15:29,863 - INFO - Response: Application onboarded request accepted
INFO: 10.15.125.10:33703 - "POST /operatorplatform/federation/v1/67d304fd6f6cae3a48f37317/application/onboarding HTTP/1.1" 202 Accepted
    
```

Figure 54: Proxy logs for Application Onboarding

List Of Applications					Zones Used					
App Name	Version	Type	App Id	Owner	Zones	Country	Operator	Status	Instances	Action
<input type="checkbox"/> sfuapplication2	1.1.21	SINGLE USER	d7aedc1682000008	admin	Omega	ES	barcelonaoperator	ONBOARDED		
<input type="checkbox"/> mediambx-2	1.0.0	SINGLE USER	j7b4c56373000008	admin						
<input type="checkbox"/> nginxapptest	1.1111	SINGLE USER	g7bf096a73000008	admin						
<input type="checkbox"/> pred-server	1.1	SINGLE USER	j790105a36000007	valdev						
<input checked="" type="checkbox"/> nginxfedtest	0.1.0	SINGLE USER	q7c6a23f23000008	adminopd efault						

Figure 55: Application Onboarding result from IEAP GUI

5.1.1.4 Application Instance Deployment

The final test involves creating an instance of the successfully onboarded application on the operator platform. This process is executed using the following API method:

- POST /{federationContextId}/application/lcm

No modifications are required from the proxy's perspective, meaning the request and response are passed through unchanged. The following figures illustrate the status in the IEAP GUI before and after the instance creation (Figure 56 and Figure 58) and provide the corresponding proxy logs (Figure 57) to verify the response. Finally, Figure 59 shows the app instance deployed at the edge from i2CAT-Barcelona side.

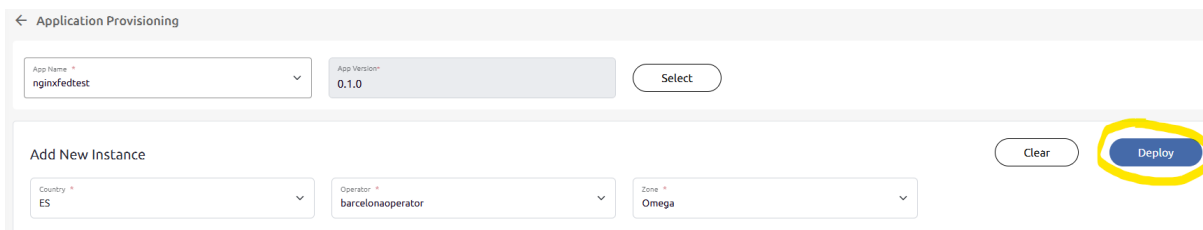


Figure 56: Application Instance Deployment from IEAP GUI

```
2025-04-09 12:33:14,016 - INFO - HTTP Request: POST http://192.168.123.46:30989/operatorplatform/federation/v1/67ee76cd6d87bc27ec751944/application/lcm "HTTP/1.1 202 ACCEPTED"
2025-04-09 12:33:14,018 - INFO - Response: {'appInstIdentifier': 'nginxfedtest1496891640170', 'zoneId': 'Omega'}
INFO: 10.15.125.10:29527 - "POST /operatorplatform/federation/v1/67ee76cd6d87bc27ec751944/application/lcm HTTP/1.1" 202 Accepted
```

Figure 57: Proxy logs for Application Instance Deployment

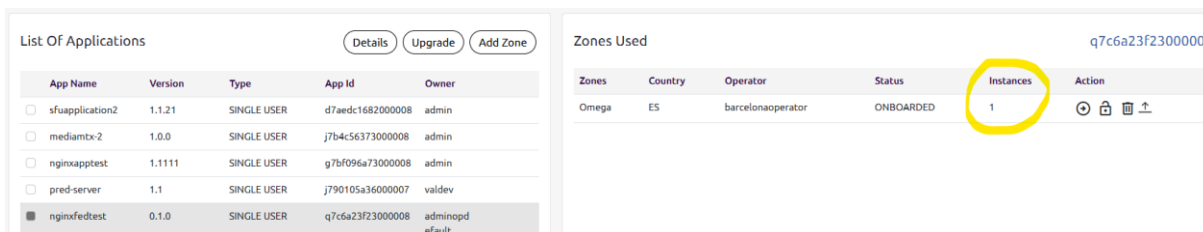


Figure 58: Application Instance Deployment result from IEAP GUI

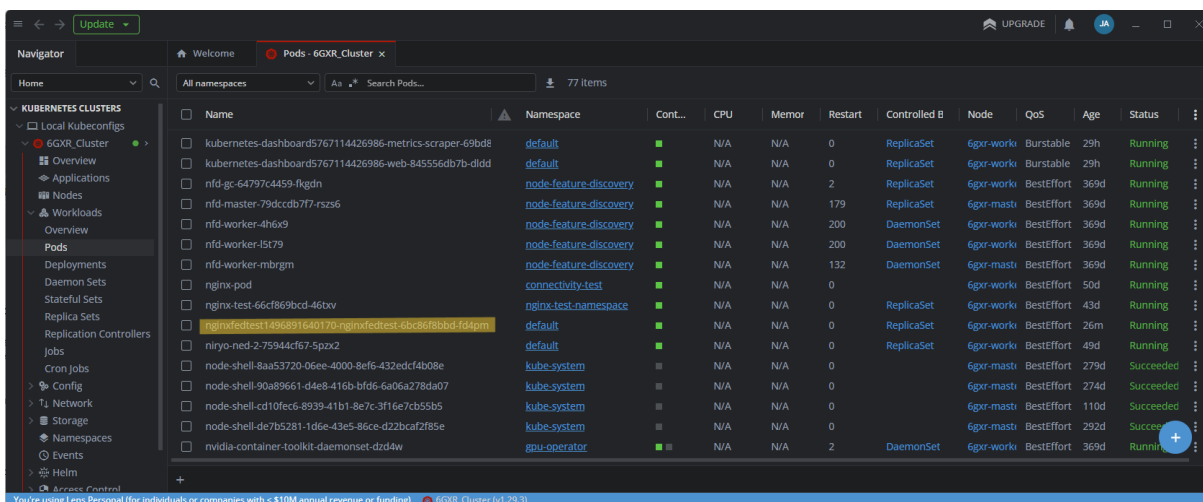


Figure 59: Application Instance Deployment result from i2CAT-Barcelona

5.1.1.5 Lessons Learned during Federation Integration

During the deployment and integration phase, it was identified that the GSMA OPG EWBI API definition lacked clarity and consistency. To facilitate development and maintain stability, the partners agreed to freeze the API at version 1.1.0, as the definition was undergoing frequent

changes. In subsequent versions, although the API specification was updated, the corresponding sample data was often not revised accordingly. In several instances, this led to ambiguities that required the development teams to reach a consensus on the expected parameter formats.

5.1.2 QoS change validation

The validation tests consist of modifying the QoS slice profiles assigned to a 5G subscriber, where different quality indicators are defined. These tests validate the enablers *E4.1 QoD API* and *E6.3 QoS Session API*.

It has been demonstrated by invoking the previously mentioned (section 3.1.3) QoD API POST method to change the slice profile from *barcelona-high* to *barcelona-low*. The API call response, along with the request payload detailing the input parameters, is shown in Figure 60. Additionally, the proxy logs corresponding to this operation are presented in Figure 61. Finally, in Figure 62, the requests for retrieving the slice profile associated with a specific phone number show different results before and after the QoD API call, thereby confirming the successful modification of the slice profile.

```

28 > session_creation_payload = {
29     "duration": 30000,
30     "device": {
31         "phoneNumber": "49165001028",
32         "ipv4Address": {
33             "publicAddress": "10.3.205.85",
34             "publicPort": 59762
35         },
36     },
37     "applicationServer": {
38         "ipv4Address": "192.168.0.1",
39         "ipv6Address": "2001:db8:85a3:8d3:1319:8a2e:370:7344"
40     },
41     "qosProfile": "QOS_S"
42 }
43

```

```

(6g-xr) $ python3 createSession.py
/home/ubuntu/venv/6g-xr/lib/python3.12/site-packages/urllib3/connectionpool.py:1097: Ins
ngly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warn
warnings.warn(
[2025-03-31 17:27:15] Request was successful! Status code: 201
[2025-03-31 17:27:15] Response JSON:
{
  "applicationServer": {
    "ipv4Address": "192.168.0.1",
    "ipv6Address": "2001:db8:85a3:8d3:1319:8a2e:370:7344"
  },
  "device": {
    "ipv4Address": {
      "publicAddress": "10.3.205.85",
      "publicPort": 59762
    },
    "phoneNumber": "49165001028"
  },
  "duration": 30000,
  "expiresAt": null,
  "qosProfile": "QOS_S",
  "qosStatus": null,
  "sessionId": "8ce7b173-dfec-4b69-8c83-684e6b80f1a9",
  "startedAt": null
}
(6g-xr) $

```

Figure 60: Request and response for QoD create session (I)


```

2025-03-31 15:27:13,608 - INFO - Modified URL: http://10.3.3.41:80/3gpp-as-session-with-qos/v1/AS/subscriptions
2025-03-31 15:27:13,608 - INFO - Payload After Modification: {'qosReference': 'barcelona-low', 'ueIpv4Addr': '10.3.205.85', 'dnn': 'parrot-emb'}
2025-03-31 15:27:14,563 - INFO - HTTP Request: POST http://10.3.3.41/3gpp-as-session-with-qos/v1/AS/subscriptions "HTTP/1.1 201 Created"
2025-03-31 15:27:14,564 - INFO - Response Headers After Modification: Headers({'date': 'Mon, 31 Mar 2025 15:27:14 GMT', 'content-type': 'applicat

```

Figure 61: Proxy logs for AsSessionWithQoS create subscription (I)

```

ericsson@server103330:~$ curl http://10.3.3.41/5tonic-exposure/v1/subscriptions/msisdn-49165001028/profile
{"Id": "barcelona-high"}
ericsson@server103330:~$ curl http://10.3.3.41/5tonic-exposure/v1/subscriptions/msisdn-49165001028/profile
{"Id": "barcelona-low"}
ericsson@server103330:~$

```

Figure 62: Slice profile information retrieving (I)

5.1.3 Finding closest Edge validation

The validation tests consist of retrieving the TAC from the NEF, and therefore, the IEAP determines the closest edge based on that information. These tests validate the enablers *E4.2 Simple Edge Discovery API* and *E6.2 UE Location API*.

Therefore, two tests have been performed, one of them for a UE located at Barcelona and the other for a UE placed at Madrid. Figure 63 and Figure 64 depict the SimpleEdgeDiscovery call for the first and second test respectively, and Figure 65 and Figure 66 show the proxy logs where the different TACs are returned by the NEF as UEs are in different testbeds.

```

35 # Barcelona
36 headers_eri = {
37     'accept': 'application/json',
38     'IP-Address': '10.3.202.66',
39     'Network-Access-Identifer': 'a4174760-b40d-43b9-9b39-e87d5f1b2b2b@domain.com',
40     'Phone-Number': '49165001028',
41     'Authorization': f'Bearer {accessToken}'
42 }
43
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(6g-xr) $ python3 eds_endpoints.py
/home/ubuntu/venv/6g-xr/lib/python3.12/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning:
ngly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
warnings.warn(
[2025-03-31 17:10:22] Request was successful! Status code: 200
[2025-03-31 17:10:22] Response JSON:
[
  {
    "edgeCloudProvider": "IEAP",
    "edgeResourceName": "Omega"
  }
]
(6g-xr) $

```

Figure 63: Request and response for SimpleEdgeDiscovery (Barcelona)

```

27 headers_eri = {
28     'accept': 'application/json',
29     'IP-Address': '10.3.202.68',
30     'Network-Access-Identifier': 'a4174760-b40d-43b9-9b39-e87d5f1b2b2b@domain.com',
31     'Phone-Number': '49165001055',
32     'Authorization': f'Bearer {accessToken}'
33 }
34 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

(6g-xr) $ python3 eds_endpoints.py
/home/ubuntu/venv/6g-xr/lib/python3.12/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning:
ngly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
warnings.warn(
[2025-03-31 17:32:17] Request was successful! Status code: 200
[2025-03-31 17:32:17] Response JSON:
[
  {
    "edgeCloudProvider": "IEAP",
    "edgeResourceName": "valenciaedge"
  },
  {
    "edgeCloudProvider": "IEAP",
    "edgeResourceName": "madridedge"
  }
]
(6g-xr) $

```

Figure 64: Request and response for SimpleEdgeDiscovery (Madrid)

```

(6g-xr) # hypercorn proxy:app -b 0.0.0.0:30801 --reload
[2025-03-31 15:10:07 +0000] [3213193] [INFO] Running on http://0.0.0.0:30801 (CTRL + C to quit)
2025-03-31 15:10:07,269 - INFO - Running on http://0.0.0.0:30801 (CTRL + C to quit)
2025-03-31 15:10:21,198 - INFO - Modified URL: http://10.3.3.41:80/3gpp-monitoring-event/v1/HoloMIT/subscriptions
2025-03-31 15:10:22,110 - INFO - HTTP Request: POST http://10.3.3.41/3gpp-monitoring-event/v1/HoloMIT/subscriptions "HTTP/1.1 200 OK"
2025-03-31 15:10:22,112 - INFO - Response After Modification: {'monitoringEventReport': {'locationInfo': {'trackingAreaId': '53006'}}},

```

Figure 65: Proxy logs for MonitoringEvent (Barcelona)

```

2025-03-31 15:32:15,935 - INFO - Modified URL: http://10.3.3.41:80/3gpp-monitoring-event/v1/HoloMIT/subscriptions
2025-03-31 15:32:16,839 - INFO - HTTP Request: POST http://10.3.3.41/3gpp-monitoring-event/v1/HoloMIT/subscriptions "HTTP/1.1 200 OK"
2025-03-31 15:32:16,841 - INFO - Response After Modification: {'monitoringEventReport': {'locationInfo': {'trackingAreaId': '53005'}}},

```

Figure 66: Proxy logs for MonitoringEvent (Madrid)

5.1.4 Changing UPF validation

The changing UPF validation tests are similar to the tests performed in section 5.1.2, where the change of QoS slice profiles assigned to a 5G subscriber was done. However, in this case the change was made from *madrid-low* to *barcelona-high*, which implies changing not only the QoS on demand, but also changing the UPF the UE is connected to from Madrid to Barcelona. Figure 67 shows the request done to QoD API, Figure 68 the proxy logs, and Figure 69 the change in UE slice profile.

```

28 session_creation_payload = {
29     "duration": 30000,
30     "device": {
31         "phoneNumber": "49165001028",
32         "ipv4Address": {
33             "publicAddress": "10.3.202.66",
34             "publicPort": 59762
35         },
36     },
37     "applicationServer": {
38         "ipv4Address": "192.168.0.1",
39         "ipv6Address": "2001:db8:85a3:8d3:1319:8a2e:370:7344"
40     },
41     "qosProfile": "QOS_L"
42 }

```

```

(6g-xr) $ python3 createSession.py
/home/ubuntu/venv/6g-xr/lib/python3.12/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning: Unverified HTTPS request is being made.
ngly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
warnings.warn(
[2025-03-31 17:21:27] Request was successful! Status code: 201
[2025-03-31 17:21:27] Response JSON:
{
  "applicationServer": {
    "ipv4Address": "192.168.0.1",
    "ipv6Address": "2001:db8:85a3:8d3:1319:8a2e:370:7344"
  },
  "device": {
    "ipv4Address": {
      "publicAddress": "10.3.202.66",
      "publicPort": 59762
    },
    "phoneNumber": "49165001028"
  },
  "duration": 30000,
  "expiresAt": null,
  "qosProfile": "QOS_L",
  "qosStatus": null,
  "sessionId": "1c012401-1b14-4e3b-af91-28f79d1eda04",
  "startedAt": null
}
(6g-xr) $

```

Figure 67: Request and response for QoS create session (II)

```

2025-03-31 15:21:26,327 - INFO - Modified URL: http://10.3.3.41:80/3gpp-as-session-with-qos/v1/AS/subscriptions
2025-03-31 15:21:26,327 - INFO - Payload After Modification: {'qosReference': 'barcelona-high', 'ueIpv4Addr': '10.3.202.66', 'dnn': 'parrot-emb'}
2025-03-31 15:21:27,289 - INFO - HTTP Request: POST http://10.3.3.41:3gpp-as-session-with-qos/v1/AS/subscriptions "HTTP/1.1 201 Created"
2025-03-31 15:21:27,291 - INFO - Response Headers After Modification: Headers({'date': 'Mon, 31 Mar 2025 15:21:27 GMT', 'content-type': 'application/json'})

```

Figure 68: Proxy logs for AsSessionWithQoS create subscription (II)

```

ericsson@server103330:~$
ericsson@server103330:~$ curl http://10.3.3.41/5tonic-exposure/v1/subscriptions/msisdn-49165001028/profile
{"Id": "madrid-low"}
ericsson@server103330:~$
ericsson@server103330:~$
ericsson@server103330:~$ curl http://10.3.3.41/5tonic-exposure/v1/subscriptions/msisdn-49165001028/profile
{"Id": "barcelona-high"}
ericsson@server103330:~$
ericsson@server103330:~$

```

Figure 69: Slice profile information retrieving (II)

The UE used for this particular test was located in the Barcelona testbed. Therefore, it is expected that the communication performance will vary depending on whether the UE is connected to the Madrid UPF or the Barcelona UPF, due to the proximity difference between the UE and the UPF.

Figure 70 and Figure 71 display the distinct time values obtained from a ping command to reach the UE, highlighting the latency differences between the two UPF connections.

```
ericsson@mini-pc-parrot:~$ cpe_ip
10.3.202.67
ericsson@mini-pc-parrot:~$ ping 192.168.123.190
PING 192.168.123.190 (192.168.123.190) 56(84) bytes of data.
64 bytes from 192.168.123.190: icmp_seq=1 ttl=247 time=31.6 ms
64 bytes from 192.168.123.190: icmp_seq=2 ttl=247 time=32.3 ms
64 bytes from 192.168.123.190: icmp_seq=3 ttl=247 time=36.9 ms
^C
--- 192.168.123.190 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 31.612/33.605/36.886/2.337 ms
```

Figure 70: Ping command for UE connected to Madrid UPF

```
ericsson@mini-pc-parrot:~$ cpe_ip
10.3.205.85
ericsson@mini-pc-parrot:~$ ping 192.168.123.190
PING 192.168.123.190 (192.168.123.190) 56(84) bytes of data.
64 bytes from 192.168.123.190: icmp_seq=1 ttl=251 time=17.9 ms
64 bytes from 192.168.123.190: icmp_seq=2 ttl=251 time=20.7 ms
64 bytes from 192.168.123.190: icmp_seq=3 ttl=251 time=16.7 ms
64 bytes from 192.168.123.190: icmp_seq=4 ttl=251 time=16.6 ms
^C
--- 192.168.123.190 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 16.583/17.955/20.677/1.657 ms
```

Figure 71: Ping command for UE connected to Barcelona UPF

5.1.5 Collecting metrics validation

This test consists in gathering performance metrics from the network and validates the enabler *E6.4 Data Collection API*. The metrics are used in the UCs by the Application Function (AF) called Congestion Detection Function (CDF) developed within WP3.

For the test, iPerf3 tool was used to produce 50 Mbps downlink traffic between an application server located at 5Tonic lab in Madrid and a UE located in Barcelona. Figure 72 demonstrates successful retrieval of metrics for the time period when traffic was ongoing. These metrics report a downlink data rate of 50 Mbps, as expected.

```
ericsson@server103330:~$ curl -H 'Content-Type:application/json' -d '{"analyEvent":{"value":"NPN_KPI"},"analyEventFilter":{"start":"2025-04-02T09:19:00Z","stop":"2025-04-02T09:20:00Z","cellId":"491683841"}}' -X POST http://10.3.3.41/5tonic-exposure/v1/analyticsexposure/Parrot/fetch

{"trafficPpiData":{"TrafficPpiRanData":{"RanDownlinkThroughput":{"mean":"50.633601","stddev":"1.264914","potentialAnomalies":"","cdfDistribution":[{"threshold":"1.000000","value":"0.000000"}, {"threshold":"2.000000","value":"0.000000"}, {"threshold":"4.000000","value":"0.000000"}, {"threshold":"8.000000","value":"0.000000"}, {"threshold":"16.000000","value":"0.000000"}, {"threshold":"32.000000","value":"0.000000"}, {"threshold":"64.000000","value":"1.000000"}]}, "RanUplinkThroughput":{"mean":"0.084812","stddev":"0.002390","potentialAnomalies":"","ActiveUsersDL":{"min":"1","max":"1"}, "ActiveUsersUL":{"min":"1","max":"1"}, "HarqDLAck16Qam":{"value":"14054"}, "HarqDLAck64Qam":{"value":"111525"}, "HarqDLAck256Qam":{"value":"0"}, "HarqDLAckQpsk":{"value":"437"}, "HarqULAck16Qam":{"value":"16048"}, "HarqULAck64Qam":{"value":"75241"}, "HarqULAck256Qam":{"value":"0"}, "HarqULAckQpsk":{"value":"34"}, "PdschTable1McsDistr":{"value":"map[0:0 1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0 9:0 10:0 11:0 12:0 13:0 14:0 15:0 16:0 17:0 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:0 29:0 30:0 31:0]"}, "PdschTable2McsDistr":{"value":"map[0:0 1:0 2:0 3:282 4:149 5:6062 6:5669 7:13 8:44 9:139 10:2043 11:3281 12:35815 13:70565 14:1941 15:16 16:4 17:0 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:10 29:805 30:11841 31:0]"}, "PdschTable3McsDistr":{"value":"map[0 1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0 9:0 10:0 11:0 12:0 13:0 14:0 15:0 16:0 17:0 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:0 29:0 30:0 31:0]"}, "PuschTable1McsDistr":{"value":"map[0:0 1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0 9:0 10:0 11:0 12:0 13:0 14:0 15:0 16:0 17:0 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:0 29:0 30:0 31:0]"}, "PuschTable2McsDistr":{"value":"map[0:3 1:7 2:4 3:3 4:17 5:120 6:477 7:1047 8:1336 9:7262 10:5813 11:14122 12:21944 13:26412 14:10597 15:19 11 16:244 17:22 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:93 29:945 30:565 31:0]"}, "PuschTable3McsDistr":{"value":"map[0:0 1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0 9:0 10:0 11:0 12:0 13:0 14:0 15:0 16:0 17:0 18:0 19:0 20:0 21:0 22:0 23:0 24:0 25:0 26:0 27:0 28:0 29:0 30:0 31:0]"}, "UtilizationDistributionDL":{"value":"map[5:202 10:0 15:0 20:1 25:0 30:0 35:0 40:0 45:0 50:0 55:0 60:0 65:0 70:0 75:0 80:5 85:12 90:84 95:0 100:0]"}, "UtilizationDistributionUL":{"value":"map[5:304 10:0 15:0 20:0 25:0 30:0 35:0 40:0 45:0 50:0 55:0 60:0 65:0 70:0 75:0 80:0 85:0 90:0 95:0 100:0]"}, "RanEnergyConsumption":{"value":"4"}, "RanDataVolume":{"value":"273578713"}, "RanEnergyConsumptionPerByte":{"value":"0.00000014621020605503031"}}, "TrafficPpiCoreData":{"UpfDataVolume":{"value":"361534338"}, "UpfDownlinkThroughput":{"mean":"50.140098","stddev":"1.351362"}, "UpfUplinkThroughput":{"mean":"3.420545","stddev":"0.611868"}}, "TrafficPpiNpnData":{"Latency":{"mean":"N/A","stddev":"N/A"}, "LinkLocalAvailability":{"value":"0.000000"}, "LinkCentralAvailability":{"value":"98.875923"}}}
```

Figure 72: Data Collection API example

5.2 SOUTH NODE CONTROL PLANE ENABLERS

This section reports the integration validation tests done for the enablers needed for UC3 *Control plane innovations* in the South Node (see D1.1 [2] for UC description).

5.2.1 IMS VMs connectivity validation

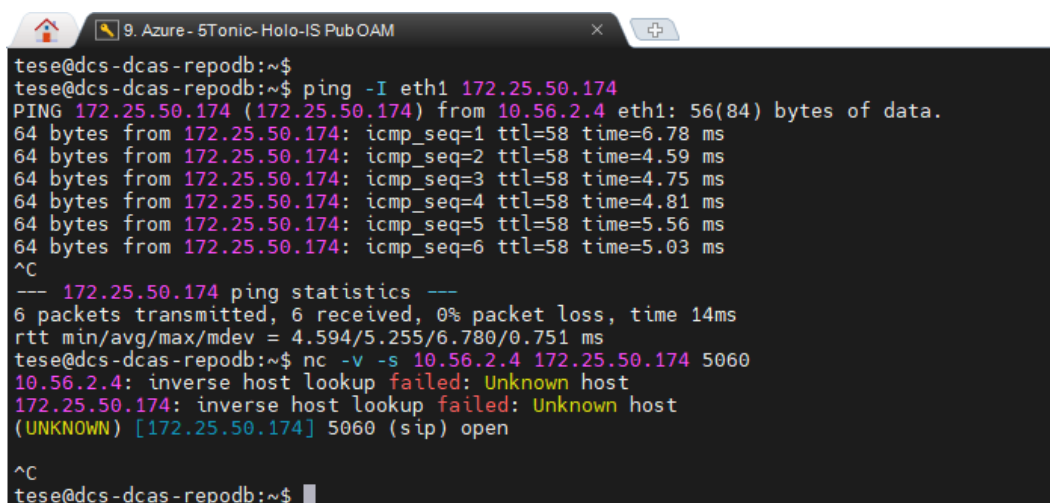
All the required flows among the IMS and the Data Channel VMs have been verified. The details of the flows, the required protocols, and the source and destination ports are listed in Table 8.

Table 8: IMS Data Channel solution flows

Flows	Protocol	Source Port	Destination Port
I/S CSCF -> DCAS	SIP/TCP, UDP	ANY	5060, 5061
DCAS -> I/S CSCF	SIP/TCP, UDP	ANY	5060
vBGF -> DCAS	RTP, SRTP, DTLS, SCTP	ANY	1024-65535
DCAS -> vBGF	RTP, SRTP, DTLS, SCTP	ANY	1024-65535
PBX-GW -> Agent	WebSocket /5000 (browser agent <-> pbx-gw), 8081 (alp logic <-> pbx-gw) /TCP	5000, 8081	ANY

Agent -> PBX-GW	WebSocket /5000 (browser agent <-> pbx-gw), 8081 (alp logic <-> pbx-gw) /TCP	ANY	5000, 8081
PBX -> PBX-GW	SIP/TCP, UDP	ANY	ANY
PBX-GW -> PBX	SIP/TCP, UDP	ANY	ANY
vBGF -> PBX-GW	RTP, SRTP, DTLS, SCTP	ANY	1024-65535
PBX-GW -> vBGF	RTP, SRTP, DTLS, SCTP	ANY	1024-65535
PBX -> Agent	SIP-WS/8089/TCP HTTP, HTTPS/8088, 8089/TCP	8088, 8089	ANY
Agent -> PBX	SIP-WS/8089/TCP HTTP, HTTPS/8088, 8089/TCP	ANY	8088, 8089
PBX -> Agent	RTP	ANY	1024-65535
Agent -> PBX	RTP	ANY	1024-65535
PBX -> vBGF	RTP	ANY	1024-65535
vBGF -> PBX	RTP	ANY	1024-65535
Agent -> vBGF	DTLS, SCTP	ANY	1024-65535
vBGF -> Agent	DTLS, SCTP	ANY	1024-65535

Figure 73 below shows an example of this validation. It is a connectivity test between DCAS and I/S-CSCF, and the connection to port 5060.



```
tесе@dcs-dcas-repodb:~$
tесе@dcs-dcas-repodb:~$ ping -I eth1 172.25.50.174
PING 172.25.50.174 (172.25.50.174) from 10.56.2.4 eth1: 56(84) bytes of data.
64 bytes from 172.25.50.174: icmp_seq=1 ttl=58 time=6.78 ms
64 bytes from 172.25.50.174: icmp_seq=2 ttl=58 time=4.59 ms
64 bytes from 172.25.50.174: icmp_seq=3 ttl=58 time=4.75 ms
64 bytes from 172.25.50.174: icmp_seq=4 ttl=58 time=4.81 ms
64 bytes from 172.25.50.174: icmp_seq=5 ttl=58 time=5.56 ms
64 bytes from 172.25.50.174: icmp_seq=6 ttl=58 time=5.03 ms
^C
--- 172.25.50.174 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 14ms
rtt min/avg/max/mdev = 4.594/5.255/6.780/0.751 ms
tесе@dcs-dcas-repodb:~$ nc -v -s 10.56.2.4 172.25.50.174 5060
10.56.2.4: inverse host lookup failed: Unknown host
172.25.50.174: inverse host lookup failed: Unknown host
(UNKNOWN) [172.25.50.174] 5060 (sip) open
^C
tесе@dcs-dcas-repodb:~$
```

Figure 73: Example of IMS VMs connectivity validation

5.3 NORTH NODE 3D DIGITAL TWIN ENABLERS

This section describes the integration tests run on North Node enablers. As NNA is the main enabler in the North Node, five module integration tests were implemented in the development phase. This section reports the results from those tests. The results of the integration test between the North Node Web Portal and Qosium are also presented.

5.3.1 NNA – Qosium Integration Test

Description: Test that two-point measurements per slice start correctly and downlink and uplink KPIs (throughput, jitter, latency, packet loss ratio) per slice can be obtained from Qosium. Finally, test that the measurements are stopped correctly.

Steps: Perform the following Qosium REST API calls per slice:

- POST /measurement/start
- GET /AverageResult?qmId=<id>&limit=1&sort=desc
- GET /measurement/stop?QsMeasId=<id>

Expected outcome: All REST API calls return 200 – OK status code and KPIs are returned in responses.

Result: The REST API calls returned OK status and KPIs were returned correctly. The initiation of per-slice measurements, fetching KPI data, and terminating the running measurements can be seen in Figure 74, Figure 75, and Figure 76, respectively. PASSED.

```

2025-04-08 13:54:33.219 DEBUG qosium: Starting measurement; SliceId=5480f617-8
d26-4ece-8b5a-a811b08f0012
2025-04-08 13:54:33.220 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:54:54.583 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"POST /measurement/start HTTP/11" 200 54
2025-04-08 13:54:54.585 INFO qosium: Measurement started; QSMeasId=11, Interna
lName=aut_10_535587800
2025-04-08 13:54:54.585 DEBUG qosium: Starting measurement; SliceId=c969da01-6
09b-4c71-ae96-1bece0ebf6c2
2025-04-08 13:54:54.587 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:15.865 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"POST /measurement/start HTTP/11" 200 54
2025-04-08 13:55:15.866 INFO qosium: Measurement started; QSMeasId=12, Interna
lName=aut_10_986910132
2025-04-08 13:55:15.867 DEBUG qosium: Initializing Qosium: Done

```

Figure 74: Test logs of the initiation of per-slice measurements in Qosium


```

2025-04-08 13:55:15.867 DEBUG qosium: Getting the latest measurement result; Q
sMeasId=11
2025-04-08 13:55:15.869 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:15.876 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"GET /AverageResult?qmId=11&limit=1&sort=desc HTTP/11" 200 None
2025-04-08 13:55:15.878 DEBUG qosium: Getting the latest measurement result; Q
sMeasId=12
2025-04-08 13:55:15.879 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:15.885 DEBUG urllib3.connectionpool: http://12.29.12.66:8080
"GET /AverageResult?qmId=12&limit=1&sort=desc HTTP/11" 200 None
2025-04-08 13:55:15.886 INFO qosium: [{'downlink': {'throughput': 2368.0, 'lat
ency': 0.24651818, 'jitter': 0.073, 'packetLoss': 0.0}, 'uplink': {'throughput
': 3880.0, 'latency': 0.2644818, 'jitter': 0.07, 'packetLoss': 0.0}}, {'downli
nk': {'throughput': 2368.0, 'latency': 0.38063797, 'jitter': 0.007, 'packetLos
s': 0.0}, 'uplink': {'throughput': 3880.0, 'latency': 0.48036203, 'jitter': 0.
591, 'packetLoss': 0.0}}]

```

Figure 75: Test logs of fetching per-slice DL/UL measurement KPIs from Qosium

```

2025-04-08 13:55:17.887 DEBUG qosium: Cleaning up Qosium: Stopping all running
measurements
2025-04-08 13:55:17.889 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:17.909 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"GET /measurement/status/all HTTP/11" 200 None
2025-04-08 13:55:17.911 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:22.196 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"GET /measurement/stop?QSMeasId=12 HTTP/11" 200 None
2025-04-08 13:55:22.198 DEBUG urllib3.connectionpool: Starting new HTTP connec
tion (1): 172.29.12.66:8080
2025-04-08 13:55:26.386 DEBUG urllib3.connectionpool: http://172.29.12.66:8080
"GET /measurement/stop?QSMeasId=11 HTTP/11" 200 None
2025-04-08 13:55:26.387 DEBUG qosium: Cleaning up Qosium: Done

```

Figure 76: Test logs of the termination of per-slice measurements in Qosium

5.3.2 NNA – Cumucore Integration Test

Description: Verify that two slices can be created and deleted through the Cumucore REST API by providing a set of slice configuration parameters.

Steps: Perform the following Cumucore REST API calls per slice:

- POST /api/v1.0/network-slice/slice-instance
- DELETE /api/v1.0/network-slice/slice-instance/<id>

Expected outcome: All REST API calls return 200 – OK status code and two slices are created and visible in Cumucore GUI until they are deleted.

Result: The REST API calls returned OK status, and the created two slices were visible in Cumucore GUI before they were deleted by the test. Figure 77 shows the logs from creating and deleting

network slices and Figure 78 demonstrates how the created slices are visible in Cumucore GUI. PASSED.

```

2025-04-08 15:02:13.363 DEBUG cumucore: Creating Cumucore slice '5480f617-8d26-4ece-8b5a-a811b08f0012'; SST=[{'sst': 1, 'sd': '000002'}]
2025-04-08 15:02:13.364 DEBUG urllib3.connectionpool: Starting new HTTPS connection (1): 172.29.19.2:3000
2025-04-08 15:02:13.393 DEBUG urllib3.connectionpool: https://172.29.19.2:3000 "POST /api/v1.0/network-slice/slice-instance HTTP/11" 200 79
2025-04-08 15:02:13.393 DEBUG cumucore: Creating Cumucore slice 'c969da01-609b-4c71-ae96-1bece0ebf6c2'; SST=[{'sst': 1, 'sd': '000003'}]
2025-04-08 15:02:13.394 DEBUG urllib3.connectionpool: Starting new HTTPS connection (1): 172.29.19.2:3000
2025-04-08 15:02:13.420 DEBUG urllib3.connectionpool: https://172.29.19.2:3000 "POST /api/v1.0/network-slice/slice-instance HTTP/11" 200 79
2025-04-08 15:02:23.460 DEBUG cumucore: Deleting existing Cumucore slice '5480f617-8d26-4ece-8b5a-a811b08f0012'
2025-04-08 15:02:23.461 DEBUG urllib3.connectionpool: Starting new HTTPS connection (1): 172.29.19.2:3000
2025-04-08 15:02:23.517 DEBUG urllib3.connectionpool: https://172.29.19.2:3000 "DELETE /api/v1.0/network-slice/slice-instance/5480f617-8d26-4ece-8b5a-a811b08f0012 HTTP/11" 200 54
2025-04-08 15:02:23.519 DEBUG cumucore: Deleting existing Cumucore slice 'c969da01-609b-4c71-ae96-1bece0ebf6c2'
2025-04-08 15:02:23.521 DEBUG urllib3.connectionpool: Starting new HTTPS connection (1): 172.29.19.2:3000
2025-04-08 15:02:23.584 DEBUG urllib3.connectionpool: https://172.29.19.2:3000 "DELETE /api/v1.0/network-slice/slice-instance/c969da01-609b-4c71-ae96-1bece0ebf6c2 HTTP/11" 200 54
    
```

Figure 77: Test logs of creating and deleting slices in Cumucore

Slice Instances	
Create Network Slice	
Name	SST
Slice1	sst 1 sd 1
slice2	sst 1 sd 2
Slice3	sst 1 sd 3
5480f617-8d26-4ece-8b5a-a811b08f0012	sst 1 sd 2
c969da01-609b-4c71-ae96-1bece0ebf6c2	sst 1 sd 3

Figure 78: Created slices visible in Cumucore GUI

5.3.3 NNA – AI/ML Integration Test

Description: Test that network resource allocation decision is received from AI/ML via a REST API call by submitting per-slice KPI data as a parameter.

Steps: Perform the following AI/ML REST API call:

- POST /allocate_resource

Expected outcome: The AI/ML REST API call returns 200 – OK status code and returns network resource allocation between slices as fractions of one (1 meaning 100 per cent).

Result: The REST API call returned OK status and network resource allocation decision. The execution logs of the test can be seen in Figure 79 showing OK response and returned allocation. PASSED.

```
2025-04-08 13:29:24.555 DEBUG ai: Sending KPIs for AI/ML to process
2025-04-08 13:29:24.556 DEBUG urllib3.connectionpool: Starting new HTTP connecti
on (1): 172.29.6.15:5000
2025-04-08 13:29:24.577 DEBUG urllib3.connectionpool: http://172.29.6.15:5000 "P
OST /allocate_resource HTTP/11" 200 74
2025-04-08 13:29:24.577 DEBUG ai: AI/ML response: {'allocation': {'slice1': 0.41
449254751205444, 'slice2': 0.5855074524879456}}
2025-04-08 13:29:24.578 INFO ai: Slice allocation; Slice1=0.41449254751205444, S
lice2=0.5855074524879456
```

Figure 79: Test logs of the AI/ML integration test

5.3.4 NNA – OSM Integration Test

Description: Test that two virtual machine instances with desired applications (Nginx on both hosts) are instantiated and stopped correctly in OpenStack.

Steps: Perform the following OSM REST API calls:

- POST /osm_create
- DELETE /osm_delete

Expected outcome: All REST API calls return 200 – OK status code and instantiated VMs are visible in OpenStack GUI until they are deleted by the test. While running, Nginx web server can be reached via web browser on both VMs.

Result: The REST API calls returned OK status and VMs were visible in OpenStack until deleted by the test script. Nginx was reachable on both VMs via web browser. The logs of the test script execution can be seen in Figure 80, which shows successful initiation and teardown of OpenStack VMs. PASSED.

```

2025-04-08 14:46:12.516 DEBUG osm: Initializing OSM: Starting VMs
2025-04-08 14:46:12.518 DEBUG urllib3.connectionpool: Starting new HTTP connect
ion (1): 10.50.150.103:5001
2025-04-08 14:46:13.720 DEBUG urllib3.connectionpool: http://10.50.150.103:5001
"POST /osm_create HTTP/11" 200 571
2025-04-08 14:46:13.721 DEBUG osm: VMs started; VMData=[{'details': {'id': '98f
46634-feb2-4d9c-af67-75a3c99fb56c'}, 'id': '98f46634-feb2-4d9c-af67-75a3c99fb56
c', 'instantiate_status': 'success', 'status': 'Success, wait for 1 minute befo
re proceeding as VM is being setup for you', 'vm': 'VM1_Nginx'}, {'details': {'
id': '11e5e9de-84f5-476d-8e4f-aad8629a001c'}, 'id': '11e5e9de-84f5-476d-8e4f-aa
d8629a001c', 'instantiate_status': 'success', 'status': 'Success, wait for 1 mi
nute before proceeding as VM is being setup for you', 'vm': 'VM2_Nginx'}]
2025-04-08 14:46:13.722 DEBUG osm: Initializing OSM: Done
2025-04-08 14:48:13.722 DEBUG osm: Cleaning up OSM: Stopping all running VMs
2025-04-08 14:48:13.724 DEBUG urllib3.connectionpool: Starting new HTTP connect
ion (1): 10.50.150.103:5001
2025-04-08 14:48:13.775 DEBUG urllib3.connectionpool: http://10.50.150.103:5001
"GET /osm_info HTTP/11" 200 28648
2025-04-08 14:48:13.778 DEBUG urllib3.connectionpool: Starting new HTTP connect
ion (1): 10.50.150.103:5001
2025-04-08 14:49:14.924 DEBUG urllib3.connectionpool: http://10.50.150.103:5001
"DELETE /osm_delete HTTP/11" 200 253
2025-04-08 14:49:14.925 DEBUG osm: Cleaning up OSM: Done

```

Figure 80: Test logs of the OSM integration test

5.3.5 NNA – OVS Integration Test

Description: Test that ingress policy rates and burst values can be set for N3 and N6 interfaces in Open vSwitch responsible for handling the rate limiting of slices.

Steps: Perform the following JSON-RPC queries to OVS listening on port 6640:

- Send a JSON-RPC message with ingress rate/burst values set for each interface
- Send a JSON-RPC message with zero ingress rate/burst values for each interface

Expected outcome: All JSON-RPC messages return a response without error and all interfaces have desired policy rates and burst values set accordingly until set back to zero by the test script.

Result: All queries to OVS return error value “None” (success) and rate and burst values could be verified to have changed by using ovs-vsctl command line tool at the OVS host. Log of test run altering ingress rates and burst values on N3 and N6 interfaces is shown in Figure 81. PASSED.

```

2025-04-08 15:14:37.845 DEBUG ovs: Setting DL rate/burst for slice '5480f617-8d
26-4ece-8b5a-a811b08f0012': 10000/5000 (cumucore-upf5-ranup-vhost)
2025-04-08 15:14:37.849 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:37.849 DEBUG ovs: Setting UL rate/burst for slice '5480f617-8d
26-4ece-8b5a-a811b08f0012': 20000/10000 (cumucore-upf5-n6-vhost)
2025-04-08 15:14:37.851 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:37.851 DEBUG ovs: Setting DL rate/burst for slice 'c969da01-60
9b-4c71-ae96-1bece0ebf6c2': 30000/15000 (cumucore-upf4-ranup-vhost)
2025-04-08 15:14:37.853 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:37.853 DEBUG ovs: Setting UL rate/burst for slice 'c969da01-60
9b-4c71-ae96-1bece0ebf6c2': 40000/20000 (cumucore-upf4-n6-vhost)
2025-04-08 15:14:37.854 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:57.855 DEBUG ovs: Setting DL rate/burst for slice '5480f617-8d
26-4ece-8b5a-a811b08f0012': 0/0 (cumucore-upf5-ranup-vhost)
2025-04-08 15:14:57.861 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:57.862 DEBUG ovs: Setting UL rate/burst for slice '5480f617-8d
26-4ece-8b5a-a811b08f0012': 0/0 (cumucore-upf5-n6-vhost)
2025-04-08 15:14:57.866 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:57.867 DEBUG ovs: Setting DL rate/burst for slice 'c969da01-60
9b-4c71-ae96-1bece0ebf6c2': 0/0 (cumucore-upf4-ranup-vhost)
2025-04-08 15:14:57.871 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}
2025-04-08 15:14:57.871 DEBUG ovs: Setting UL rate/burst for slice 'c969da01-60
9b-4c71-ae96-1bece0ebf6c2': 0/0 (cumucore-upf4-n6-vhost)
2025-04-08 15:14:57.875 DEBUG ovs: OvS response: {'id': 1, 'result': [{'count':
  1}], 'error': None}

```

Figure 81: Logs of the OVS integration test

5.3.6 North Node Web Portal – Qosium Measurement Integration Test

Description: In this section, the user should view and analyse the generated results related to the experiment. The information is presented to the user through the Grafana dashboard.

Steps: To test and receive measurement results, the user first sends the NST to NNA. Then, the measurements generated by Qosium are stored in the database by the North Node web portal. A request is sent to Qosium every 50 seconds to retrieve the latest measurements, which are then saved in the database. The user can select four KPIs to view the related measurements and, if desired, download the results. Additionally, the user can terminate the experiment in Qosium when needed by sending a request to NNA to stop running measurements in Qosium.

- POST /home/api/grafana/
- POST /home/api/downloadcsvview/
- DELETE /home/api/terminate/<id>/

Expected outcome: The user should be able to easily select their preferred KPIs, clearly view the results in real time, download the results, and terminate the experiment if needed.

Result: During the experiment, the user can select their desired KPIs on the dashboard and monitor the corresponding results in real time. The result download and experiment termination functionalities were also successfully tested. PASSED.

Figure 82 illustrates the integration between the North Node Web Portal and Qosium Storage.

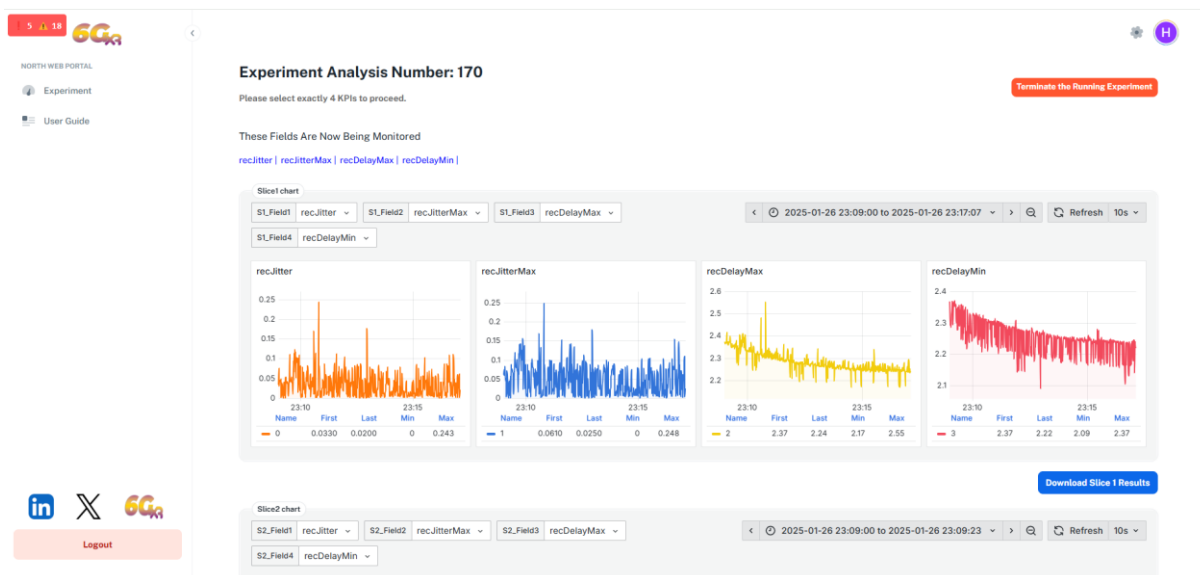


Figure 82: Qosium measurement results in North Node Portal

5.4 NORTH NODE ENERGY FRAMEWORK ENABLERS

This section includes the integration tests carried out on the North Node UOULU side of energy measurement framework and sustainability experimentation enablers. At the UOULU site, the OAIBOX comprising the gNB, core, radio unit, and user equipment, serves as the main enabler for carrying out integration and validation of WP5 experimentations. During the development phase, module integration tests were conducted to validate the KPIs defined in D1.1 [2], including the cost counter, CO₂ counter, and active energy counter. These tests include 66-hour ahead energy weather forecast, 24-hour ELSHOT electricity pricing, and real-time FINGRID's CO₂ emission estimates. They also enabled data exchange between VTT and UOULU including the real time energy consumption of individual network components OAIBOX gNB, core, USRPs and UEs. This section presents the results of these tests, covering the integration of data exchange between VTT and UOULU (North Node gNB sites).

5.4.1 Data exchange between VTT and UOULU gNB sites using MQTT bridge broker integration

Description: To test and validate successful data exchange between the VTT and UOULU gNB sites, real-time energy monitoring was performed using the MQTT bridge broker (Mosquitto). During the test, MQTT clients subscribed to all relevant topics and consistently received live data from energy monitoring equipment deployed at both sites as shown in Figure 83. This included readings such as voltage, current, energy, power factor, and solar charger power yield from VTT-Oulu-Zencon solar charger and the UOULU external meter Netio-PowerBOX-77. In addition, data from forecasting APIs, PV production inverters, and on-site sensors were also successfully transmitted between the North Node sites.

Steps: Perform the following steps:

- **Start the MQTT client with command:**

```
mosquitto_sub -t "#" -i "emf" -p <port> -h <broker_address> -u <username> -P <password>
```

- **Subscribe to Power and Energy Topics**

VTT-Oulu-Zencon/solarcharger/258/Yield/Power

Values: {"value": 1366.77001953125}, {"value": 140.0}

VTT-Oulu-Zencon/battery/512/Dc/0/Voltage

Values: {"value": 49.19999694824219}

Voltage, current, power factor, energy, and load across multiple outputs (e.g., output/1/voltage, output/2/current, etc.) including active energy counter for individual network components such as OAIBOX gNB, core USRPs and UEs.

Result: The MQTT-based data exchange worked as expected. All relevant KPI data was received in real time from both VTT and UOULU sites without any interruptions or message loss. The continuous flow of active energy counter values confirms that the MQTT bridge broker is functioning reliably. The results of the message flow and KPI updates can be seen in Figure 83.


```

Last login: Mon Apr 21 11:28:40 2025 from 193.166.32.12
hamid@xr-emf:~$ mosquitto_sub -t '#' -d -u "emf" -P "Satellite@5gtn"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) received PUBLISH (d0, q0, r1, m0, 'N/38d2693a4962/battery/512/Dc/0/Voltage', ... (29 bytes))
{"value": 49.540000915527344}
Client (null) received PUBLISH (d0, q0, r1, m0, 'VTT-Oulu-Zencom/battery/512/Dc/0/Voltage', ... (29 bytes))
{"value": 49.189998626708984}
Client (null) received PUBLISH (d0, q0, r1, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (16 bytes))
{"value": 140.0}
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (28 bytes))
{"value": 1366.1700439453125}
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/load', ... (2 bytes))
47
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/voltage', ... (3 bytes))
231
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (28 bytes))
{"value": 1366.8299560546875}
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/current', ... (3 bytes))
735
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/load', ... (3 bytes))
153
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/POWER_FACTOR', ... (4 bytes))
0.90
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/3/voltage', ... (3 bytes))
231
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/3/current', ... (2 bytes))
11
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/3/load', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/3/POWER_FACTOR', ... (4 bytes))
0.39
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (27 bytes))
{"value": 1366.260009765625}
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/POWER_FACTOR', ... (4 bytes))
0.88
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/energy', ... (3 bytes))
164
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/load', ... (3 bytes))
148
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/current', ... (3 bytes))
231
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/voltage', ... (3 bytes))
231
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/energy', ... (3 bytes))
164
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/energy', ... (2 bytes))
52
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (27 bytes))
{"value": 1366.800048828125}
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/battery/512/Dc/0/Voltage', ... (28 bytes))
{"value": 50.529998779296875}
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/total/energy', ... (3 bytes))
221
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (26 bytes))
{"value": 1366.77001953125}
Client (null) received PUBLISH (d0, q0, r0, m0, 'VTT-Oulu-Zencom/solarcharger/258/Yield/Power', ... (28 bytes))
{"value": 1368.6600341796875}
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/2/load', ... (2 bytes))
48
Client (null) received PUBLISH (d0, q0, r0, m0, 'netio/PowerBOX-77/output/1/voltage', ... (3 bytes))

```

Figure 83: Data exchange validation between VTT and UOULU

5.4.2 North Node UOULU forecasting APIs integration

Description: This section covers the use of forecasting APIs, including the site-specific FMI energy weather forecast (up to 66 hours ahead), Elspot electricity pricing (24-hour ahead), and real-time CO₂ emissions estimates from Fingrid. The data is stored in a central database and visualized for the user through a Grafana dashboard.

Steps: The following steps were performed to integrate and validate the forecasting APIs into the energy measurement framework.

- **FMI Energy Weather Forecast:** The main.py script was executed as shown in Figure 85 via a nohup background process, storing outputs in fmigitoutput.log. The script successfully simulated 3-day PV generation forecasts.
- **Elspot Electricity Pricing:** The ELSPOTrequests.py script fetched hourly electricity prices using the API ² as shown in Figure 86.
- **Fingrid CO₂ emissions estimates:** The Fingrid_Open_data.py script fetched real-time grid linked CO₂ estimates using the API ³ as shown in Figure 87.

Result: All three forecasting APIs (FMI energy weather forecast, Elspot pricing, Fingrid CO₂) were successfully called, parsed, and validated. Output data is correctly formatted, time-aligned to Finnish local time, stored in central database, and visualized in Grafana as real-time dashboards for energy budgeting shown in Figure 84. The central controller publishes this data using MQTT broker, allowing each component (energy and network) to subscribe to these forecasting APIs.

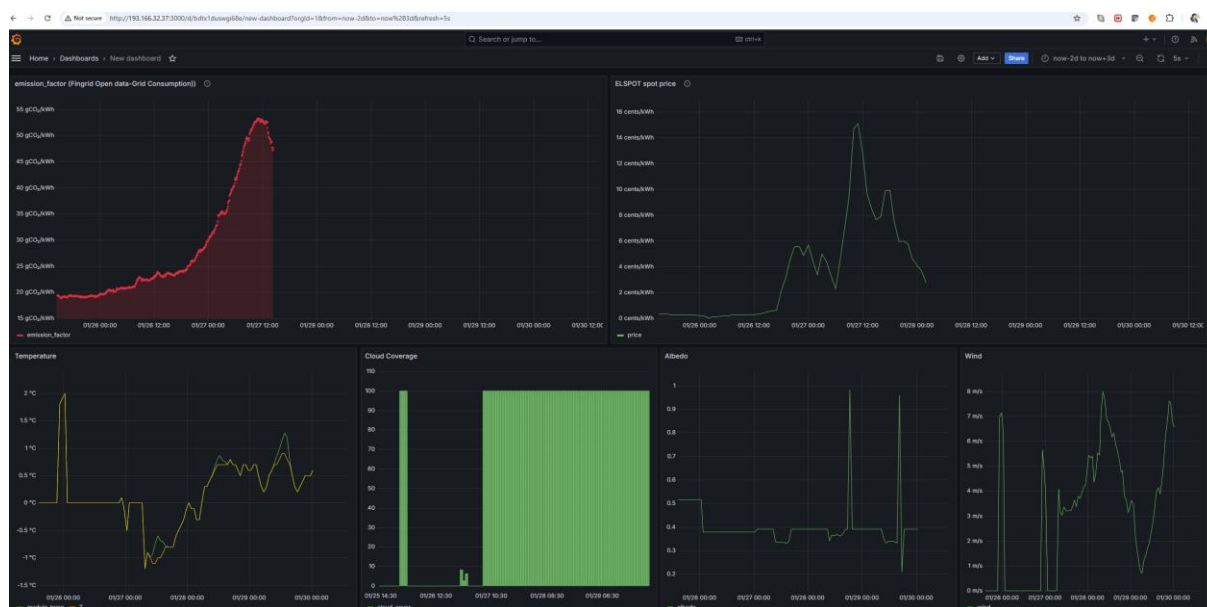


Figure 84: Screenshot of Grafana visualization of integrated forecasting APIs

² <https://api.porssisahko.net/v1/latest-prices.json>

³ <https://data.fingrid.fi/api/datasets/266/data>

```

hamid@xr-omf:~$ nohup /bin/python3 /home/hamid/fmi-open-pv-forecast/main.py > /home/hamid/fmi-open-pv-forecast/fmigitoutput.log 2>&1 &
[1] 184381
hamid@xr-omf:~$ tail -f /home/hamid/fmi-open-pv-forecast/fmigitoutput.log
nohup: ignoring input
Simulating clear sky and weather model based PV generation for the next 3 days.
Index(['time', 'dni', 'dhi', 'ghi', 'dir_hi', 'albedo', 'T', 'wind',
      'cloud_cover', 'dni_poa', 'dhi_poa', 'ghi_poa', 'poa', 'dni_rc',
      'dhi_rc', 'ghi_rc', 'poa_ref_cor', 'module_temp', 'output'],
      dtype='object')
Time
2025-04-21 00:00:00      0.00
2025-04-21 01:00:00      0.00
2025-04-21 02:00:00      0.00
2025-04-21 03:00:00      0.00
2025-04-21 04:00:00      0.00
2025-04-21 05:00:00      0.00
2025-04-21 06:00:00      0.00
2025-04-21 07:00:00      0.00
2025-04-21 08:00:00      0.00
2025-04-21 09:00:00      0.00
2025-04-21 10:00:00      0.00
2025-04-21 11:00:00    6,074.05
2025-04-21 12:00:00    5,825.10
2025-04-21 13:00:00    5,331.40
2025-04-21 14:00:00    4,402.19
2025-04-21 15:00:00    3,096.52
2025-04-21 16:00:00    1,482.34
2025-04-21 17:00:00     247.72
2025-04-21 18:00:00      73.48
2025-04-21 19:00:00       0.00
2025-04-21 20:00:00       0.00
2025-04-21 21:00:00       0.00
2025-04-21 22:00:00       0.00
2025-04-21 23:00:00       0.00
2025-04-22 00:00:00       0.00
2025-04-22 01:00:00       0.00
2025-04-22 02:00:00       0.00
2025-04-22 03:00:00      19.68
2025-04-22 04:00:00     180.32
2025-04-22 05:00:00     541.75
2025-04-22 06:00:00    1,269.21
2025-04-22 07:00:00    2,044.21
2025-04-22 08:00:00    2,912.04
2025-04-22 09:00:00    3,241.44
2025-04-22 10:00:00    2,337.25
2025-04-22 11:00:00    1,426.80
2025-04-22 12:00:00    3,042.83
2025-04-22 13:00:00    2,669.42
2025-04-22 14:00:00     867.20
2025-04-22 15:00:00     509.65
2025-04-22 16:00:00     496.22
2025-04-22 17:00:00     190.66
2025-04-22 18:00:00      58.20
2025-04-22 19:00:00       0.00
2025-04-22 20:00:00       0.00
2025-04-22 21:00:00       0.00
2025-04-22 22:00:00       0.00
2025-04-22 23:00:00       0.00
2025-04-23 00:00:00       0.00
2025-04-23 01:00:00       0.00
2025-04-23 02:00:00       0.00
2025-04-23 03:00:00      3.84
2025-04-23 04:00:00     39.28
2025-04-23 05:00:00     51.05
2025-04-23 06:00:00     64.35
2025-04-23 07:00:00     43.93
2025-04-23 08:00:00     71.51
2025-04-23 09:00:00    158.07
2025-04-23 10:00:00    232.44
2025-04-23 11:00:00    662.16
2025-04-23 12:00:00   1,430.39
2025-04-23 13:00:00   1,103.24
2025-04-23 14:00:00   1,780.11
2025-04-23 15:00:00   1,183.25
2025-04-23 16:00:00    483.88
2025-04-23 17:00:00    255.22
2025-04-23 18:00:00    114.87
2025-04-23 19:00:00       0.00
2025-04-23 20:00:00       0.00
2025-04-23 21:00:00       0.00
2025-04-23 22:00:00       0.00
2025-04-23 23:00:00       0.00
Name: output, dtype: float64
Data inserted successfully into fmigit.
Simulation plot saved as '/home/hamid/UOULU-2025-04-21 13-30.png'
-----
Simulating clear sky and weather model based PV generation for the next 3 days.
Index(['time', 'dni', 'dhi', 'ghi', 'dir_hi', 'albedo', 'T', 'wind',
      'cloud_cover', 'dni_poa', 'dhi_poa', 'ghi_poa', 'poa', 'dni_rc',
      'dhi_rc', 'ghi_rc', 'poa_ref_cor', 'module_temp', 'output'],
      dtype='object')

```

Figure 85: FMI energy weather forecast 66h API validation

```

Welcome | ELSPORequests.py X
home > hamid > ELSPORequests.py > fetch_and_publish_elspot_data
9 def fetch_and_publish_elspot_data():
10     # Step 1: Send a request to the API to fetch the pricing data
11     url = "https://api.porssisahko.net/v1/latest-prices.json"
12     response = requests.get(url)
13
14     # Step 2: Parse the JSON response
15     data = response.json()
16
17     # Step 3: Extract the relevant information (timestamps and prices)
18     prices = data['prices']
19
20     # Define the Finnish timezone (EET or EEST depending on DST)
21     finnish_tz = pytz.timezone("Europe/Helsinki")
22
23     # Convert the timestamps from UTC to Finnish time
24     times = [
25         datetime.strptime(item['startDate'], "%Y-%m-%dT%H:%M:%S.000Z")
26         .replace(tzinfo=pytz.utc) # Set the timezone to UTC first
27         .astimezone(finnish_tz) # Convert to Finnish time
28         for item in prices
29     ]
30
PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | SPELL CHECKER
'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 9, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 8, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 7, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 6, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 5, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 4, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 3, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 2, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>), datetime.datetime(2025, 4, 21, 1, 0, tzinfo=<DstTzInfo 'Europe/Helsinki' EEST+3:00:00 DST>)] [13.723, 15.839, 19.739, 34.534, 30.621, 35.14, 31.374, 24.963, 19.573, 19.855, 21.742, 30.618, 37.654, 37.654, 37.661, 33.094, 0.949, 0.963, 1.093, 1.795]
^CTraceback (most recent call last):
  File "/home/hamid/ELSPOTrequests.py", line 73, in <module>
    time.sleep(1)
KeyboardInterrupt

hamid@xr-emp:~$ /bin/python3 /home/hamid/ELSPOTrequests.py
    
```

Figure 86: ELSPOT electricity pricing 24 ahead API validation

```

Welcome | Fingrid_Open_data.py
home > hamid > Fingrid_Open_data.py > ...
1 import requests
2 from datetime import datetime
3 import mysql.connector
4 import schedule
5 import time
6
7
8 def fetch_and_publish_fingrid_data():
9     # Define the API URL for the Emission Factor dataset
10    url = "https://data.fingrid.fi/api/datasets/266/data"
11    api_key = "91c0f4395581425cac08d68ccde326aa"
12
13    # Set the headers
14    headers = {
15        "Cache-Control": "no-cache",
16        "x-api-key": api_key
17    }
18
19    # Send the GET request to the API
20    response = requests.get(url, headers=headers)
21
22    # Check if the request was successful
23    if response.status_code == 200:
24        data = response.json()
25
PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | SPELL CHECKER
hamid@xr-emp:~$ /bin/python3 /home/hamid/Fingrid_Open_data.py
Success
2025-04-21 13:18:00 28.58
Success
2025-04-21 13:21:00 28.598
Success
2025-04-21 13:21:00 28.598
Success
2025-04-21 13:21:00 28.598
^CTraceback (most recent call last):
  File "/home/hamid/Fingrid_Open_data.py", line 90, in <module>
    time.sleep(1)
KeyboardInterrupt

193.166.32.37 | Launchpad | 0 | 0 | 2 | 0 | Sourcery
    
```

Figure 87: Fingrid CO₂ estimates API validation

6 CONCLUSIONS

Overall, the partners fulfilled their objectives on the WP2 enablers. Minor delays occurred mostly on the development phase and could be easily absorbed by speeding up the deployment phase.

For UC1 *Resolution Adaptation or Quality on Demand* and UC2 *Routing to the Best Edge*, the Edge platforms are ready at Barcelona and Madrid to be able to run the applications. The federation between both platforms allows to use Barcelona Edge as a secondary Edge controlled by Madrid Edge orchestrator, allowing to start and stop application instances at Madrid or Barcelona depending on service requirements. The Northbound Interface (NBI) APIs, combining CAMARA and Network Exposure Function (NEF) APIs, enable the Application Functions (AFs) to dynamically find which is the UE location to adapt the path to the closest UPF and to change the applied QoS for the service. Additionally, there is a NEF API to collect metrics from the network.

For UC3 *Control plane innovations*, the novel IMS Data Channel solution has been developed and later deployed at the South Node for the project's use. This solution allows to start the XR service by dialling the receiver and selecting the application in the mobile phone.

For UC4 *Collaborative 3D Digital Twin-like Environment*, the component of North Node Adapter (NNA), which serves as orchestrator for the North Node, is operational and integrated with the other components such as the resource optimization module and the 5G Core. The AI-driven resource optimization enabler follows a reinforcement learning approach to make decisions based on real-time observations. This enabler calls the Slice Management API offered by the 5G Core to create and delete the required network slices.

For UC5 *Energy Measurement Framework for Energy Sustainability*, the infrastructure for the energy management framework was enhanced in the North Node. New sensors and methods are used to measure the energy consumption of a newly deployed open source based 5G network and then elaborate a consumption forecast.

The enablers developed in WP2 are ready to be used for UC validation in WP6.

7 REFERENCES

- [1] 6G-XR, "D2.2 Orchestration AI techniques End to end slicing and signalling for the core enablers Implementation," 30 October 2024. [Online]. Available: https://6g-xr.eu/wp-content/uploads/sites/96/2024/10/D2.2-Orchestration-AI-techniques-End-to-end-slicing-and-signalling-for-the-core-enablers-Implementation_3.0.pdf.
- [2] 6G-XR, "D1.1 "Requirements and use case specifications"," 30 September 2023. [Online]. Available: <https://www.6g-xr.eu/wp-content/uploads/sites/96/2023/10/D1.1-Requirements-and-use-case-specifications-V1.0.pdf>.
- [3] "Multi-Access Edge Computing," [Online]. Available: <https://www.etsi.org/technologies/multi-access-edge-computing>.
- [4] "Fishy project," [Online]. Available: <https://fishy-project.eu/>.
- [5] "GSMA Open Gateway," [Online]. Available: <https://www.gsma.com/solutions-and-impact/gsma-open-gateway/>.
- [6] "CAMARA project," [Online]. Available: <https://camaraproject.org/>.
- [7] "Flask software project," [Online]. Available: <https://pypi.org/project/Flask/>.
- [8] "Waitress software project," [Online]. Available: <https://pypi.org/project/waitress/>.
- [9] "Requests software project," [Online]. Available: <https://pypi.org/project/requests/>.
- [10] 6G-XR, "D5.1 Description of sustainability experimentation framework," 30 June 2024. [Online]. Available: <https://www.6g-xr.eu/wp-content/uploads/sites/96/2024/09/D5.1-Description-of-sustainability-experimentation-framework.pdf>.