



D3.1 Initial consolidated report on the 5G-IANA architecture elements

Dissemination level:	Public (PU)
Work package:	WP3
Task:	T3.1, T3.2, T3.3, T3.4
Deliverable lead:	UBI
Version:	V1.0
Submission date:	27/03/2023
Due date:	28/02/2023
Partners:	



NOKIA



FONDAZIONE
links
PASSION FOR INNOVATION



**Internet
INSTITUTE**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016427

5g-iana.eu

Authors

Authors in alphabetical order		
Name	Organisation	Email
Thanos Xirofotos	UBI	txirofotos@ubitech.eu
Dimitris Klonidis	UBI	dklonidis@ubitech.eu
Edoardo Bonetto	LINKS	edoardo.bonetto@linksfoundation.com
Federico Princiotta	LINKS	federico.princiotta@linksfoundation.com
Matteo Minotti	LINKS	matteo.minotti@linksfoundation.com
Konstantinos Katsaros	ICCS	k.katsaros@iccs.gr
Giorgos Drainakis	ICCS	giorgos.drainakis@iccs.gr
Gabriele Scivoletto	NXW	g.scivoletto@nextworks.it
Matteo Andolfi	NXW	m.andolfi@nextworks.it

Control sheet

Version history			
Version	Date	Modified by	Summary of changes
V0.1	23/01/2023	D. Klonidis	Deliverable structure and ToC
V0.2	02/02/2023	T. Xirofotos	Added Section 2
V0.3	06/02/2023	T. Xirofotos	Added Section 3
V0.4	10/02/2023	K. Katsaros	Added Section 6
V0.5	13/02/2023	G. Scivoletto	Added Section 4
V0.6	16/02/2023	T. Xirofotos, E. Bonetto	Figures added and updated, Added Section 5
V0.7	19/02/2023	All	Module Interconnections Section added
V0.8	23/02/2023	All	Added Section 7
V0.9	27/02/2023	A. Rizk	Review 1
V1.0	28/02/2023	All	Review comments applied
V1.1	28/02/2023	A. Rizk, E. Bonetto	Review 2
V1.2	05/03/2023	K. Katsaros	Review 3
V1.3	10/03/2023	T. Xirofotos	Review comments applied
V1.4	14/3/2023	E. Liotou	Review 4
V1.5	24/3/2023	T. Xirofotos	Addressed review 4 and added Annex C

Peer review		
	Reviewer name	Date
Reviewer 1	Amr Rizk	27/02/2023
Reviewer 2	Edoardo Bonetto	28/02/2023
Reviewer 3	Konstantinos Katsaros	05/03/2023
Reviewer 4	Eirini Liotou	14/3/2023

Legal disclaimer

The information and views set out in this deliverable are those of the author(s) and do not necessarily reflect the official opinion of the European Union. The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any specific purpose. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein. The 5G-IANA Consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Copyright © 5G-IANA Consortium, 2023.

DRAFT

TABLE OF CONTENTS

TABLE OF CONTENTS	4
EXECUTIVE SUMMARY	10
1. INTRODUCTION	11
1.1. Purpose of the deliverable.....	11
1.2. Structure of the deliverable	12
1.3. Relation with other deliverables and tasks.....	12
1.4. Intended audience.....	13
2. OVERVIEW OF THE 5G-IANA REFERENCE PLATFORM	14
3. NETWORK APPLICATION ORCHESTRATION AND DEVELOPMENT MECHANISMS.....	17
3.1. Design details of the nApp Orchestration and Development layer.....	17
3.1.1. nApp toolkit.....	18
3.1.2. Deployment Management.....	18
3.1.3. Lifecycle Management.....	19
3.1.4. Slice Intent Handler.....	19
3.1.5. Policy Execution.....	20
3.1.6. Application Profiling.....	20
3.2. nApp Orchestration & Development layer developments.....	20
3.2.1. nApp Toolkit Developments.....	20
3.2.2. Deployment Management Developments.....	21
3.2.3. Lifecycle Management Developments.....	22
3.2.4. Slice Intent Handling Developments.....	23
3.2.5. Policy Execution Developments.....	23
3.2.6. Application Profiling Developments.....	24
3.3. Planned extensions in the second development cycle	25
3.3.1. Deployment Management planned extensions.....	25
3.3.2. Lifecycle Management planned extensions.....	25
3.3.3. Slice Intent Handling planned extensions.....	25
3.3.4. Policy Execution planned extensions.....	26
3.3.5. Application Profiling planned extensions.....	26
4. SLICE MANAGEMENT AND RESOURCE ORCHESTRATION MECHANISMS	27
4.1. Design details of the Slice Manager and Resource orchestration layer	29
4.1.1. Design details of the E2E Resource Orchestration for Far Edge enabled nApps.....	29

4.1.2. Design details of the OBU/Edge/Cloud Continuum Resource Inventory	30
4.1.3. Design details of the Slice Management.....	31
4.2. Slice Manager and multidomain orchestration layer developments.....	32
4.3. Planned extensions in the second development cycle	33
5. VIRTUALISED INFRASTRUCTURE SEGMENTS	34
5.1. On-vehicle MANO, Edge and Cloud MANO	34
5.2. Information and localization service developments	35
5.3. Resource monitoring agent developments.....	36
5.4. Far-edge devices developments	37
5.5. Future extensions	38
6. CROSS LAYER FUNCTIONALITIES	39
6.1. Distributed AI/ML framework mechanisms	39
6.1.1. AFs and FLOWER integration.....	43
6.1.2. Distributed AI/ML orchestration layer developments.....	45
6.1.3. Planned extensions in the second development cycle.....	45
6.2. Monitoring & Analytics, Distributed Data Collection.....	46
6.2.1. Design Details of Monitoring & Analytics.....	46
6.2.2. Monitoring & Analytics Developments.....	47
6.2.3. Planned extensions in the second development cycle.....	47
7. INTERFACES AND WORKFLOWS	49
7.1. Design Details of the interfaces	49
7.2. Interfacing Developments	50
7.2.1. NOD Interface.....	50
7.2.2. NOD-SM Interface.....	50
7.2.3. MANO Interface.....	53
7.2.4. OVMANO Interface.....	56
8. CONCLUSION	58
ANNEX A	59
ANNEX B	61
ANNEX C	63
REFERENCES.....	67

List of Figures

Figure 1: 5G-IANA Orchestration Layers abstraction	14
Figure 2: 5G-IANA Orchestration Layers and the respective internal modules.....	15
Figure 3: nApp Orchestration and Development layer submodules.....	17
Figure 4: High Level functionalities of Slice Management & Resource Orchestration Layer	27
Figure 5: Internal and External interaction of the Slice Management and Resource Orchestration components	28
Figure 6: Internal Architecture of the E2E Resource Orchestration for Far Edge enabled nApps.....	29
Figure 7: OBU/Edge/Cloud Continuum Resource Inventory internal structure	30
Figure 8: Slice Management internal structure	31
Figure 9: Localization service workflow.....	36
Figure 10: Resource monitoring agent workflow.....	37
Figure 11: Distributed Machine Learning Orchestration (DMLO): Client Selection operation	43
Figure 12: FLOWER framework component architecture	44
Figure 13: AOEP interfaces.....	49
Figure 14: nApp Orchestration & Development – Slice Management & Resource Orchestration interaction flowchart.....	51
Figure 15: NOD-SM-IF OpenAPI specification.....	52
Figure 16: Resource Orchestration - MANO Interaction to allocate k8s quotas and namespaces	54
Figure 17: Resource Inventory - MANO Interaction to receive information about the availability of the Edge and Far Edge devices.....	54
Figure 18: NOD - OBU/RSU MANO Interaction to deploy and to verify the status of a deployed service	56
Figure 19: Visualization of the work done in the NOD layer.....	63
Figure 20: Visualization of the work done in the NOD layer.....	64

List of Tables

Table 1, nApp Toolkit interactions.....	21
Table 2, Deployment Manager interactions	21
Table 3, Lifecycle Manager interactions	22
Table 4, Slice Intent Handling interactions.....	23
Table 5, Policy Execution interactions	24
Table 6, Application Profiling interactions.....	24
Table 7, E2E Resource Orchestration interactions	30
Table 8, OBU/Edge/Cloud Continuum Resource Inventory interactions.....	31
Table 9, Slice Management interactions	32
Table 10, DMLO interactions.....	42
Table 11, Monitoring & Analytics interactions	47
Table 12, NOD-SM-IF API.....	52
Table 13, Resource Inventory - Kubernetes REST Commands	55
Table 14, Resource Inventory - Kubernetes REST Commands	55
Table 15, Description of extensions per module of NOD	63
Table 16, Description of extensions per module of SM& RO layer	65
Table 17: Description of extensions per module of the cross-layer functionalities.....	66

ABBREVIATIONS

Abbreviation	Definition
ACK	Acknowledgement
AF	Application Function
AI	Artificial Intelligence
AOEP	Automotive Open Experimental Platform
API	Application Programming Interface
CAN	Controller Area Network
CCRI	Cloud Continuum Resource Inventory
DML	Distributed AI/ML
DMLO	Distributed Machine Learning Orchestrator
EMBB	Enhanced Mobile Broadband
FL	Federated Learning
GNSS	Global Navigation Satellite System
gRPC	Google's Remote Procedure Call
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IF	Interface
JSON	Javascript Object Notation
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
ML	Machine Learning
NOD	nApp Orchestration and Development
NSI	Network Slice Instance
OBU	On Board Unit
REST	Representation State Transfer

RI	Resource Inventory
RO	Resource Orchestration
RTK	Real Time Kinetic
RSU	Road Side Unit
SDR	Software Defined Radio
SDN	Software Defined Network
SLA	Service Level Agreement
SM	Slice Manager
SME	Small and Medium Enterprise
URLLC	Ultra Reliable Low Latency Communication

DRAFT

EXECUTIVE SUMMARY

The 5G-IANA Automotive Open Experimental Platform (AOEP) aims to provide an open and flexible experimentation platform to third-party developers (e.g., SMEs) that want to develop new 5G-based services devoted to the Automotive vertical. The developments that are reported here concern the first release of 5G-IANA AOEP modules which are planned to be used in the first round of integrations towards the initial platform release. These developments follow the layered structural approach of 5G-IANA architecture as defined in D2.1. Specifically, for each of the two layers comprising the architecture, namely **nApp Orchestration and Development (NOD)** layer and the **Slice Management and Resource Orchestration** layer, this document reports the design and the developments for each of the functional blocks that are embedded in each layer. Emphasis is given on describing also the interconnections of all the functional blocks. Thus, for each of the functional blocks, the document provides a description of the functionality or the functionalities that are provided or linked with, the baseline technologies for the development, the interconnections with other functional blocks, as well as the extensions that are planned for the second development cycle that starts on M25.

The separation of the 5G-IANA orchestration platform functionalities between the two aforementioned layers serves the need to operate between the two different administrative domains. Each of these layers has a precise focus in terms of functionalities and supported procedures, implementing a clear separation of roles and responsibilities across the different platform's stakeholders. **nApp Orchestration and Development (NOD) Layer** covers the Application Domain providing mechanisms and procedures for the deployment and the real-time management of the application only (**covered in Section 3**), while the **Slice Management and Resource Orchestration** layer covers the interplay with the programmable resources incorporation mechanisms for registering and deregistering the various edges, either moving (OBU) or static (RSU), and it also possesses the logic to ensure specific performance requirements (**covered in Sections 4, 5**). Complementary, for each layer, a description is provided indicating the integration points and interfaces between other layers or entities of the platform (**covered in Section 7**).

The document also covers the developments for the cross-layer functionalities that are provided as feature of the 5G-IANA Platform, and specifically the Distributed ML/AL framework and the Monitoring and Analytics accompanied by the Distributed Data Collectors that integrate with the two aforementioned layers as well as its planned extensions (**covered in Section 6**).

1. INTRODUCTION

1.1. Purpose of the deliverable

This deliverable (D3.1) reports on the outcomes of the design and development activities carried out in the scope of WP3 and relative to the 5G-IANA Automotive Open Experimental Platform (AOEP). The document describes the developments that have been carried out from M12-M21 (February 2022) for the building blocks and interfaces of the 5G-IANA architecture that will comprise the first consolidated version (Version A) of the Platform (in M24 after finishing the first round of integrations). The reported developments stem and comply with the 5G-IANA general architecture and requirements, as they have been defined in D2.1.

More in details, with reference to the 5G-IANA overall architecture outlined in D2.1, this deliverable elaborates on the chosen design and technological solutions for the implementation of the 5G-IANA Orchestration Platform and their respective building blocks. It provides technical view on the design of each of the layers that are the: a) nApp Orchestration and Development Layer (excluding the developments of the nApp Toolkit that are reported extensively in Deliverable 4.1), b) the Slice Management and Resource Orchestration Layer including the virtualized infrastructure segments as well as c) the Distributed Machine Learning and the Monitoring and Analytics features of the Platform. To complement the technical view, this document provides a) the baseline technical description and the implemented extensions for each functional block of each layer of the architecture, and b) the interconnections between the various software modules accompanied by the feature they are linked. Descriptions of the interfaces per layer are provided accompanied by the respective developments. Deliverable 3.3 will provide and complement the technical description and developments on all the interfaces of the AOEP that are not included in the current deliverable.

This specific document puts in evidence the developments and extensions that have been introduced during the First Development Cycle of the project (M12-M21) as well as the delta with regards to the inherited work from other projects. Additionally, it includes the planned features to be addressed in the following release of the platform.

1.2. Structure of the deliverable

This deliverable follows the next structure:

- Section 2 provides an overview of the 5G-IANA Platform highlighting the Orchestration Layers providing high level information, and clarifies their respective role in the overall Platform.
- Sections 3, 4, 5, 6 provide technical descriptions with regards to the design, the implementation and the interactions between the software elements that reside in a particular layer or to others.
- Section 7 provides workflows and technical descriptions of the interfaces.

Annex A exhibits a nApp descriptor as this has been defined and implemented. This is the input that triggers **the orchestration phase that follows the nApp Definition phase**. For the sake of clarification, nApp definition phase is provided by the functionalities offered by the nApp Toolkit (described in D4.1) while orchestration phase involves all the entities and layers described in this deliverable.

Annex B demonstrates the descriptor that facilitates the slice negotiation phase between the two layers of the AOEP (NOD-SM&RO) as it has been defined and modelled.

Annex C provides an aggregated picture for all the developments and the work-done per module.

1.3. Relation with other deliverables and tasks

This deliverable reports the developments achieved during the first cycle of development (M12-M21) of the 5G-IANA. This deliverable is mapped and complies with Deliverable 2.1 and the specification of the 5G-IANA AOEP.

This deliverable collects and reports the activities of WP3 and specifically from:

- T3.1 that leads the developments in the nApp Orchestration Layer (Section 3) including also the Monitoring and Analytics part (Section 6.2).
- T3.2 & T3.3 that lead the developments in Slice Management and Resource Orchestration Layer including On-Vehicle and Road-side MANO (Section 4, Section 5).
- T3.4 that leads the developments for the Distributed ML framework that is a feature of the 5G-IANA AOEP (Section 6.1).

In accordance with the implementation strategy and plan described in the project's DoA and the project's workplan described in D5.1, WP3 presents the current developments that will be consolidated in M24 on the first release (Version A) of the 5G-IANA Orchestration Platform. D3.1 reports the active developments and the key design features that the upcoming version of the Platform will realize.

The developments of the features of the 5G-IANA Orchestration Platform (and in extension, the first consolidated version of the Platform), described in D3.1 will foster the advancement of key WP4, WP5 and WP6 activities, as they:

- provide a consolidated view of current and planned features on WP4 that will lead the aspects and the new features that the nApp Toolkit can adopt and implement.
- provide guidelines for the integration and the features supported by the overall delivered 5G-IANA AOEP.

On the other hand, the outcomes of the aforementioned activities will drive the planned future development of the Platform and define the actual content of D3.2, D3.3 and D3.4.

As already stated above, D3.1 is linked to the deliverable D2.1 (from WP2), which defines the requirements and the general architectural design approach for the 5G-IANA Platform: it is therefore recommended that the reader goes through D2.1 first, in order to obtain a better overview of the overall design, before going into the implementation details described herein.

1.4. Intended audience

The dissemination level of this deliverable is "public" (PU). It is primarily aimed to be the reference document to be used by the 5G-IANA Consortium Members during the development and integration phases of the 5G-IANA project. Furthermore, this deliverable is addressed to any interested reader (i.e., public dissemination level) who wants to be informed about the 5G-IANA AOEP architecture. It is expected to be particularly useful also for the SMEs that will be selected to experiment with 5G-IANA Platform, following the project's Open Calls.

REMARK: Deliverable 3.1 is tagged as confidential in the DoA. However, given that there is an upcoming amendment in order to apply to the following instruction "dissemination level needs to change from Confidential to Public, as requested by the project interim review." Deliverable 3.1 dissemination level is public.

2. OVERVIEW OF THE 5G-IANA REFERENCE PLATFORM

The 5G-IANA platform is specifically conceived for simplifying and automating the management of Network Applications - referred as nApps - onto programmable infrastructures, including 5G. At a glance, the proposed platform aims to mostly hide the complexity of programmable infrastructure and 5G environment to service developers and providers, and to make the development, deployment and operation of 5G-ready applications (nApps) similar to the well-known corresponding processes applied to cloud-native applications in cloud computing environments.

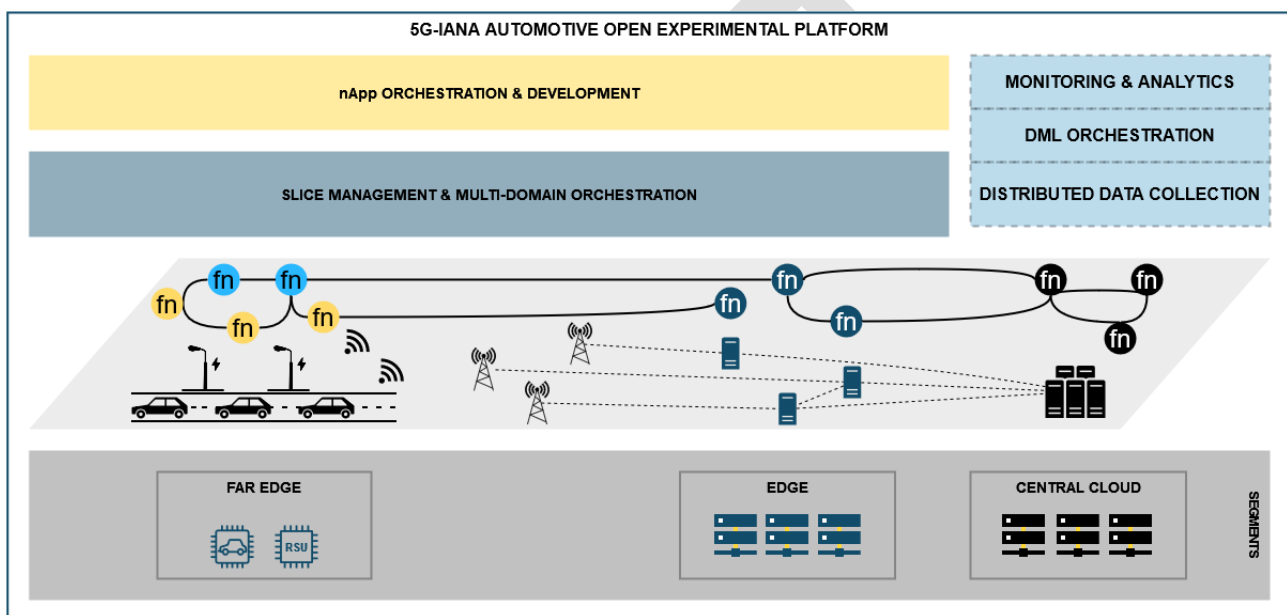


Figure 1: 5G-IANA Orchestration Layers abstraction

Figure 1 shows the 5G-IANA conceptual architecture (Figure 1 is high level view of the 5G-IANA AOEP, as it is defined in the deliverable D2.1) and highlights the two-layered Orchestration stack, whose features and implementations are extensively described in the following parts of this deliverable: the **nApp Application Orchestration and Development (NOD) (layer 1)**, the **Slice Management & Multi-Domain Orchestration**, the **virtualized infrastructure segments (layer 2)** along with the cross layer supported functionalities: The **Distributed AI/ML framework (cross-layer)**, the **Monitoring & Analytics and the Distributed Data Collection (cross-layer)**. These are described together in Section 6 for the sake of readability and given the complementarity of their features.

Figure 2 provides an internal view for each layer indicating the modules behind each provided functionality. Following a **top-down layered approach**, Section 3 provides the functionalities, the design and the developments for the **nApp Orchestration & Development**, while Section 4 and Section 5 provide the functionalities, the design and the developments for **the Slice Management & Resource Orchestration** and the different **Virtualized infrastructure segments**. Section 6 is dedicated to the cross-layer functionalities provided by the **Distributed AI/ ML framework**, and the **Monitoring & Analytics including also the Distributed Data Collectors**. Section 7 describes the **interactions** and the **interfaces** inside the AOEP. Section 7 starts from the interface which facilitates the communication with the nApp developers (technical details are excluded because this is in the scope of D4.1) and goes through the details for the communication between the two layers as well as the virtualized infrastructures. **Annex C summarizes and depicts all the extensions that have been implemented and the planned for implementation with respect to the various modules at different layers .**

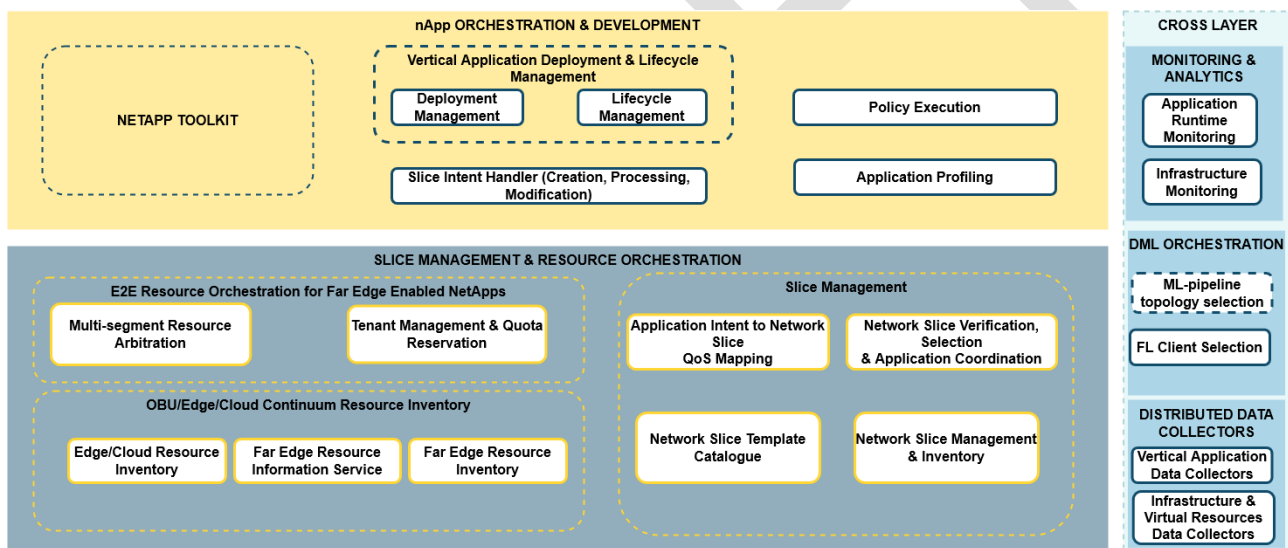


Figure 2: 5G-IANA Orchestration Layers and the respective internal modules

The separation of the 5G-IANA orchestration platform functionalities between the two aforementioned layers serves the need to operate between the two different administrative domains. The two administrative domains the platform spans are: the **Application Domain (in yellow)** and the **Infrastructure Domain (in blue)**. The distinction of layers targets the different “work-burden” that has to be achieved and managed. This way, the tools of the orchestration are targeting two lifecycles and specifically a) of the application and b) of the programmable infrastructure and network services. In this sense, **the 5G-IANA Platform is comprised by a set of orchestration tools with each set devoted to its specific (applicative or network)**

administrative domain. Each administrative domain is operated by a specific stakeholder: for the Application Domain the stakeholders are nApps developers of various automotive vertical industries, while for the **Infrastructure Domain** the stakeholders are Programmable infrastructure owners including 5G network operators. For the sake of clarity, **the reader must know that Slice Management & Resource Orchestration Layer handles the communication with various edges including the on vehicle MANO. Given that the on-board units (OBU) and road side units (RSU) are part of the programmable resources, the specific work described is undertaken by the Slice Management & Resource Orchestration Layer.**

The cross-layer functionalities (as Figure 2 shows) are **runtime functionalities** supported by the 5G-IANA Platform and they are:

- a. **the Monitoring** which is described in Section 3 and
- b. **the Distributed AI/ML** feature which is described in Section 6.

It is important to mention that the end-to-end deployment of a nApp, including its runtime services, mandates the orchestrator to programmatically dictate the configuration of computing, storage and network resources in a continuous and efficient way.

3. NETWORK APPLICATION ORCHESTRATION AND DEVELOPMENT MECHANISMS

This section goes through the design and the functionalities provided for each individual module of the nApp Orchestration and Development Layer, the developments that took place (Section 3.2) and the planned extensions for each particular module until the release B of the 5G-IANA platform (Section 3.3).

3.1. Design details of the nApp Orchestration and Development layer

The objective of this layer (as already expressed in Deliverable 2.1) is **to undertake the deployment and real-time management of nApps. Under this scope, this layer interfaces the end user** (i.e., the automotive industry application developer) for managing the deployable applications and their features. It decouples the application layer management procedures from the network layer management. As such, it **interfaces** the underlying slice creation and management systems, providing compatibility with any network orchestration solution and their respective slice management subsystems.

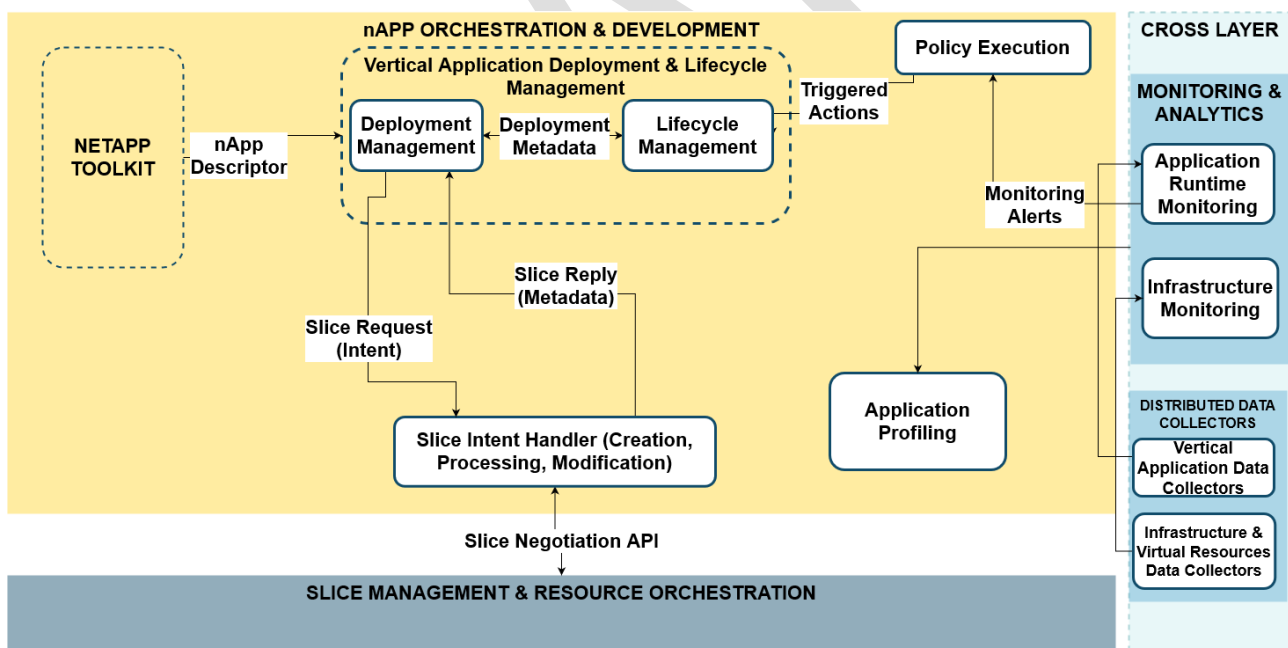


Figure 3: nApp Orchestration and Development layer submodules

More specifically, the **nApp Orchestration and Development Layer** consists of all software modules that are responsible for registering a nApp and all its components, wrapped with the cloud-related and slice-related metadata, authoring deployment and runtime policies, negotiating slices and managing the operational state

of the nApp within the scope of a materialized slice. Figure 3 provides an overview of the **nApp Orchestration and Development layer** architecture including all its modules and the southbound interface. The modules are described in the following subsection while the interface is described in Section 7.

3.1.1. nApp toolkit

The **nApp toolkit** is thoroughly described in Deliverable 4.1 and there will be no technical details of it in this deliverable; however, for the sake of reader's clarity and to provide the connection with the Orchestration layers, a description is provided. Thus, the **nApp Toolkit** offers the abstraction for registering, onboarding nApp components and composing a nApp without interfering with the infrastructure and network configuration details (e.g., the application developers tags one container for OBU deployment without the need to know any technology-specific information like metadata that are required for a deployment in Kubernetes or in Openstack or through docker-compose or Helm charts [1]). It assists the vertical service providers to wrap cloud native nApps in a proper format, so as to be publishable in the Container Registry (see D4.1 - Section 2.1.2). Each nApp consists of multiple containers that are chained in the form of a service graph (Deliverable 2.1). The nApp Toolkit provides design-time validation in the granularity of component model and guarantees that required cloud-native properties are maintained through metadata regarding:

- a. minimum infrastructural requirements
- b. configuration parameters per atomic component
- c. mutable configuration parameters during runtime
- d. exposed and required interfaces – dependencies for a component to be functional
- b. deployment preferences

As it can be seen from Figure 3, the **nApp Toolkit** is one of the functional blocks provided by the **nApp Orchestration and Development Layer** and **provides input to the Vertical Deployment & Lifecycle Management block (see the Annex A).**

3.1.2. Deployment Management

The Deployment Manager is the software module that carries out the task to realise the initial actual deployment of a nApp Graph on top of programmable resources by undertaking the task of communication

with the underlying registered infrastructure. To do so, the Deployment Manager has to be fully aware of the **available resources, their state and directive-indications (declared through the 5G-IANA specific metadata)** for the actual deployment. The knowledge of the resource-state derives from the interaction of the Deployment Manager with the Resource Inventory (see Section 4.1.2). It is important to state that the ability of this module to perform deployments in various virtualization environments derives from a built-in capability to interact with various virtualization endpoints. Hence, **the Deployment Manager provides an abstract interface for basic management virtualization operations (e.g., select/create tenant, create/modify namespaces, etc.) by incorporating some of the industry-dominant virtualization technologies (Openstack [2], Kubernetes [3], OpenShift[4]) and their respective APIs.**

3.1.3. Lifecycle Management

The Lifecycle Manager is a closed control loop that assesses continuously the existing resources, the deployment requests and the reconfiguration requests of the already deployed nApps. This software module encapsulates the orchestration business logic and can be seen as a centralized logical entity that affects the “scheduling” of nApp components that are distributed at various virtualizations levels i.e., at the cloud-edge-OBU level. In a nutshell, **the Lifecycle Manager is responsible to maintain the operational state of the nApp.** The state is mapped to an amount of resources that are allocated per component of the nApp, the slice parameters that have been selected, the geographical constraints of the deployment, etc. Any change on these aspects has to be handled by the control plane that the Lifecycle Manager oversees.

3.1.4. Slice Intent Handler

The purpose of the Slice Intent Handler module is to handle the complete lifecycle of the Network Slice creation process. The conceptual flow of this module includes **a) the slice request phase, b) the slice instantiation, c) the slice operation and d) the slice deprovision.** During the instantiation phase, the creation and verification of the necessary virtualized environment is performed. This process programmatically allocates and assigns resources to virtualized environments to host the lifecycle of the nApp. These environments will be adopted to satisfy the nApp requirements coming from the Service Composition module (which belongs to the nApp Toolkit). The translation of the requirements to a proper slice configuration plan is a primary objective of the Slice Intent Handler **(see a formalized slice intent expressing the user-defined requirements in json in the Annex B).** Upon completion of the instantiation phase, all resources shared/dedicated to the slice should have been created and configured.

3.1.5. Policy Execution

As previously stated, the Lifecycle Manager is responsible to maintain the operational state of all nApp components and to deal with any fluctuations on the different aspects (changes in the operational state that Lifecycle Manager stores – see Section 3.1.3) of an “operational” state that are: the amount of resources that are allocated per component of the nApp, the slice parameters that have been selected, the geographical constraints of the deployment, etc. Some of these aspects can be deliberately amended by policies. Policies are instructions regarding how the overall nApp should behave prior to the deployment and during runtime. **Policies are based on their property to affect the initial deployment or the overall runtime behaviour. In 5G-IANA the Policies are covering only the runtime phase (not the prior to deployment phase) and are expressed as a set of rules and actions.** Each rule is defined as Service Level Agreement (SLA) expressions (formally SLOs [5]) that conceptualize the desired state at the nApp graph-level or at the nApp component-level. In a nutshell, policies are realized through a rule-based framework that attempts to derive execution instructions based on the current set of data and the active rules; rules associated with the deployed service graphs at each point of time. The SLAs can be authored through the **Policy Editor** as rules (which belongs to the functionalities provided by nApp Toolkit described in D4.1) and the data is fed to the Policy Execution through the Monitoring module that is responsible to collect data based on a set of active monitoring probes. The evaluation of the rules according to incoming monitoring data takes place in the **Policy Execution** as well as the triggering of specific actions that the Lifecycle Manager will need to realize.

3.1.6. Application Profiling

The Application Profiling is designed for supporting various profiling aspects at nApp component and nApp graph level. The scope of this software entity is the examination of the behaviour of the application characteristics (e.g., resources’ usage efficiency, scaling or potential scaling actions and profiles, identification of bottlenecks, identification of capacity limits and breaking points upon stressing the software processes, reliability aspects taking into account time series data with identified problems and failures) under various conditions. Application Profiling consumes monitoring metrics by the Monitoring & Analytics with specific criteria e.g., fetching nApp ID or nApp component monitoring data, historical data timeslot and more.

3.2. nApp Orchestration & Development layer developments

3.2.1. nApp Toolkit Developments

NApp Toolkit Developments are thoroughly described in Deliverable 4.1. However, **for a nApp deployment, the starting point is when a nApp descriptor is being sent from the nApp Toolkit to the Deployment Manager. During the first development cycle (M12-M21) of the project the development activities focused on the respective integration of the nApp Toolkit with the Deployment Manager.** In particular, the definition of the descriptor of a nApp (see in the Annex) and the modelling in order to proceed to implementation accompanied with front-end extensions of these features is what mainly took place over the first development phase.

Given that the nApp Toolkit is the persistency layer for: a) the registered nApps, b) the deployment and slice constraints and c) the authored policies, there will be no technical descriptions for these functionalities on this Deliverable but in D4.1.

Module's Interactions

The nApp Toolkit interacts with the following software modules.

Table 1, nApp Toolkit interactions

Interacting Module	Description
Deployment Manager	Exposed API for retrieval (POST) of the nApp Descriptor before proceeding for an actual deployment

3.2.2. Deployment Management Developments

The Deployment Manager is being developed as a microservice^[6] mainly in Spring framework^[7] and partially in Quarkus framework^[8], for optimizing the memory footprint and the spin-up times (of the microservices that runs inside containers). Since the Deployment Manager will operate on top of Kubernetes (including all the flavors that are described in Section 5.1) it is implemented as a Kubernetes controller responsible for service graphs. All the microservices are developed in Java 11^[9] and many aspects of development, like the development of RESTful interfaces^[10], concurrency on service multiple requests, uniform reporting and analytics during operation, among others, are considered granted. Besides REST, the communication among some of the microservices happens through a Kafka^[11] bus.

Module's Interactions

The Deployment Manager interacts with the following software modules.

Table 2, Deployment Manager interactions

Interacting Module	Description
Lifecycle Manager	Receives from the Deployment Manager all the necessary data and settings for the nApp deployment. The data regard the current and previous orchestration loop states and are stored internally.
nApp Toolkit	Connects to the Deployment Manager for sending the nApp descriptor before a deployment.
Slice Intent Handling	After the selection of a nApp, through this interface the Deployment Manager propagates three distinct 'sets' of information, i.e. cloud constraints, network constraints and location of the deployment of that particular nApp. After that, the slice negotiation process with the Slice Manager takes places.

3.2.3. Lifecycle Management Developments

Lifecycle Manager developments are on the same track of work with Deployment Manager. These two are tied together with the only exception that Lifecycle Manager executes the policies. It is important to note here that, service instances management is provided natively by Kubernetes. Lifecycle Manager integrates the Kubernetes API for programmatically triggering actions under the actions inferred by the Policy module.

Module's Interactions

The Lifecycle Manager interacts with the following software modules.

Table 3, Lifecycle Manager interactions

Interacting Module	Description
Deployment Manager	The Deployment Manager triggers a new instance deployment. The Lifecycle Manager implements the orchestrator loop, supervising all the steps and continuously checking the operational state of all the deployments and notifies the Deployment Manager for a re-deployment and maintenance procedure.
Policies Execution	Consumes the actions produced by the Policy Execution module and triggers the Deployment Manager (e.g., scaling action).

3.2.4. Slice Intent Handling Developments

Slice Intent Handling is a Quarkus [6] microservice purely developed in Java 11 with sole purpose **to support the complete lifecycle of Network Slice creation for the nApp Orchestration layer (see Section 3.1.4)**. It is also responsible for parsing and evaluating the realised slice (sent back by the Slice Manager) before signalling the Deployment Manager to start the deployment (see section 7.2.2). Slice instantiation and slice operation phases are totally controlled by the Slice Manager and Resource Orchestration Layer. Currently, there are ongoing developments for realizing the explicit Edge selection given by the end-user i.e., tag a nApp component to be deployed on top of a moving Edge. This type of information has to be reflected inside the resource metamodel (communication details for connecting on the specific Edge, virtualization technology e.g. microk8s, etc.), formulated and communicating to the underlying layer (the E2E Resource Orchestration for Far Edge enabled nApps).

Module's Interactions

The Slice Intent Handling interacts with the following software modules.

Table 4, Slice Intent Handling interactions

Interacting Module	Description
Deployment Manager	<p>Deployment Manager's interface is used to obtain the set of information, i.e. cloud constraints, network constraints, location for the deployment of that particular nApp in order to formulate that into a specific format.</p> <p>Through different resource of the same interface, it signals the Deployment Manager to start deployment of nApp after slice reply has been sent by the Slice Manager (see cell below).</p>
E2E Resource Orchestration for Far Edge enabled nApps (SLICE MANAGEMENT & RESOURCE ORCHESTRATION layer)	<p>The slice negotiation signalling is performed through a Northbound interface of the Slice Manager (see NOD-SM-IF section 7). This is the connection point with the underlying layer for exchanging slice requests.</p>

3.2.5. Policy Execution Developments

Policies are developed in plain Java code and the policy creation is based on Drools [12]. Policies are realized through a rule-based framework that attempts to derive execution instructions based on the current set of

data and the active rules; rules are associated with the deployed application graphs at each point in time. Drools Engine faces a problem with regards to the bottleneck of performance when rule analysis takes places^[13]. During the usage of Drools in 5G-IANA project the aforementioned problem was identified and in order to overcome this an optimization technique has been implemented. Specifically, to minimize the execution time the optimization was to **translate the Drools Flow process straight into Java code and executing that Java code instead**. With that, and as confirmed by discussion threads in the KIA community ^[14], Drools are performing adequately fast. Policies are not new. **In 5G-IANA the developments regard the extensions on the optimization to deal with the scalability and the performance bottleneck of Drools.**

Module's Interactions

The **Policy Execution** interacts with the following software modules:

Table 5, Policy Execution interactions

Interacting Module	Description
Monitoring	Monitoring is publishing alerts while the Policy Execution consumes them to evaluate a policy
Lifecycle Manager	It concurrently executes activated policies to infer corrective actions

3.2.6. Application Profiling Developments

The baseline technologies used for the development of the profiler are Python 3.8, Kafka [9] (for streaming data pipelines) and OpenCPU (for data analysis) ^[15] and Tensorflow (for machine learning) ^[16]. **Application profiling is not new. Extensions for the profiling are not started yet (See the Planned extensions section).**

Module's Interactions

The **Profiling** interacts with the following software modules:

Table 6, Application Profiling interactions

Interacting Module	Description
Monitoring	Profiler consumes specific timeseries metrics by applying filtering queries to the Monitoring

3.3. Planned extensions in the second development cycle

This section goes through the planned developments and extensions as identified and validated by the project's objectives.

3.3.1. Deployment Management planned extensions

Currently the Deployment Manager is refactored in order to be generic, with the right levels of abstraction to support both OpenStack and K8s APIs, while there are still features in the K8s API that will be implemented next. Specifically, in the second development circle, there will be extensions to enable support for Kubernetes Configmaps^[17]. Given that, Configmaps are a way to store configuration data in Kubernetes they can be used to decouple Kubernetes configuration from application code, making it easier to manage and update application configurations without changing the underlying code. Thus, this can be used to provide elevated reconfigurability taking into account application-specific configuration properties for a nApp graph.

3.3.2. Lifecycle Management planned extensions

The planned extensions relate with scaling strategies of a nApp and the migration strategy that will realize the process of moving services from one node to another. **Another extension is the integration with the DMLO.** Specifically, the definition and incorporation of specific rules that will produce an action, as instructed by the DMLO (see Section 5) is currently under investigation.

3.3.3. Slice Intent Handling planned extensions

Extensions on adaptive slices (i.e., with inflated/deflated cloud) through a "sliceUpdate" resource in the **Slice Intent Handling-Slice Manager API** is being examined. The sliceUpdate will take place after the initial deployment of the nApp and will cover changes on the already allocated and provisioned slice. Specifically, the requirements for implementing a slice update are currently under investigation, focusing on updates which:

- a. will pro-grammatically increase/decrease the computing resources needed for the application during runtime.
- b. will handle a request for transferring an already deployed nApp component from one (edge) cluster/node to another.

3.3.4. Policy Execution planned extensions

Policies will be updated and integrated with the rest of the platform in order to be able to insert more rules that will trigger actions on other software modules and specifically the DMLO.

3.3.5. Application Profiling planned extensions

The Profiler is planned to introduce its function during the second cycle of development after the monitoring engine is fully integrated in the 5G-IANA Platform. The developments will be mainly the integration with the Overall platform.

DRAFT

4. SLICE MANAGEMENT AND RESOURCE ORCHESTRATION MECHANISMS

The Slice Manager and Resource Orchestration layer (introduced in Section 2 and depicted in Figure 1, Figure 2) implements the **mechanisms to manage and orchestrate the available Edge and Far-Edge resources**, along with the (Network Slice Instances (NSIs) that can be used to support the system.

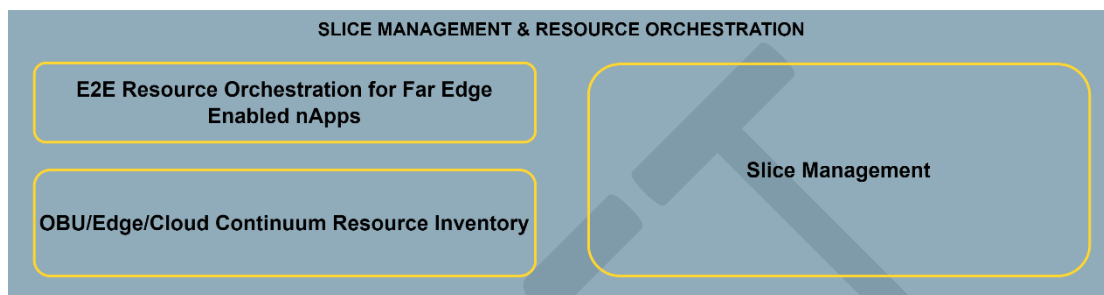


Figure 4: High Level functionalities of Slice Management & Resource Orchestration Layer

The layer is composed of three main functionalities implemented by its subcomponents:

- **The E2E Resource Orchestration for Far Edge enabled nApps:** it provides the functionalities to manage and coordinate the resource quotas at the target host, considering the received provisioning requests.
- **OBU/Edge/Cloud Continuum Resource Inventory (CCRI):** it implements a continuously updated inventory of available Edge and Far Edge computing resources to be targeted when a service provisioning action is addressed by the Slice Intent Handler (see Figure 3). It exposes an interface to retrieve the current list of the available Edge and Far Edge devices, along with their **static capabilities** (computing resources, presence of GPUs, etc.). Moreover, it can rely on external platforms (monitoring platforms, information services) to query its internal device registry based on filtering criteria (for instance, location-based queries): the Far Edge Resources Information Service internal component is able to periodically query an external monitoring entity (like a Prometheus-based monitoring platform) to hold dynamic information (like the location of a device) to enrich its internal registry and expose an interface to be queried based on filtering criteria.
- **Slice Management:** It provides the functionalities to map the Service Intent request (provided by the Slice Intent Handler component that is part of the nApp Orchestration and Development layer) with the most suitable NEST [18] according to a set of translation rules. It relies on a set of pre-defined 5G Network Slice instances to be chosen according to the requested QoS indicator.

An overall picture of the internal and external interactions of the Slice Management & Resource Orchestration Layer software components is depicted in Figure 5 and described in the sections following.

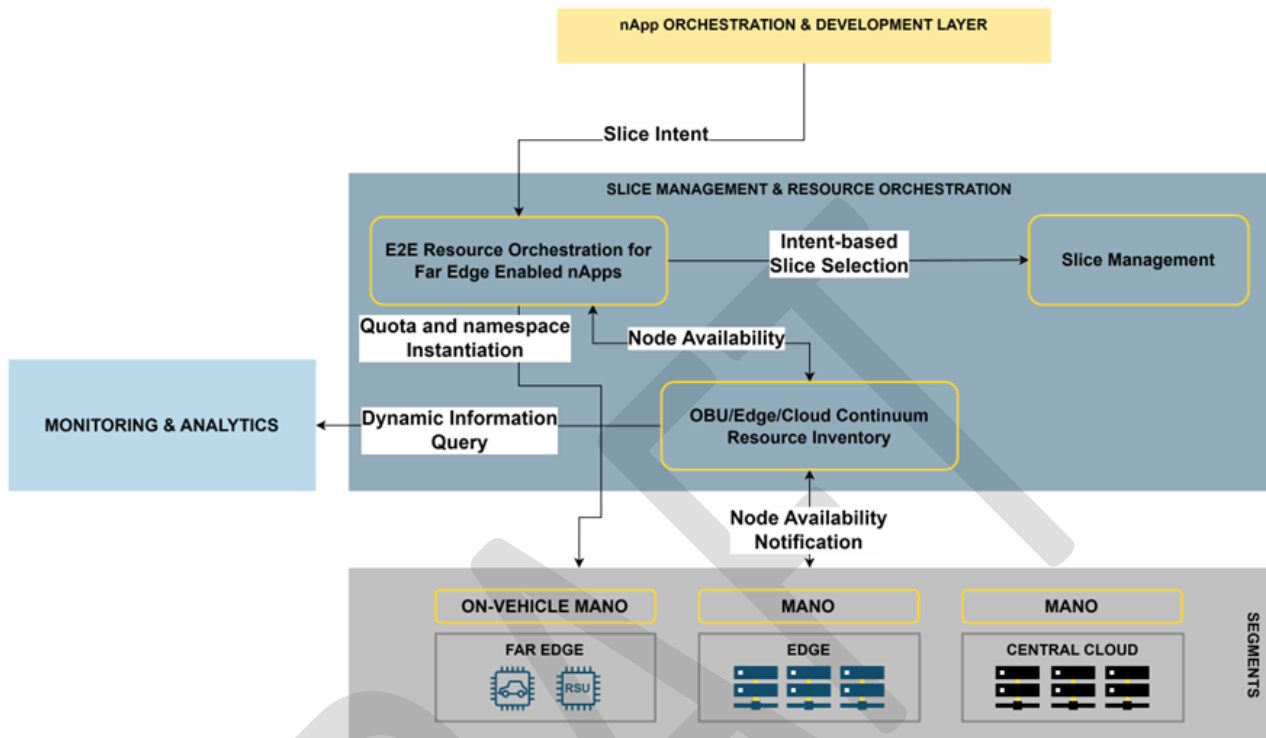


Figure 5: Internal and External interaction of the Slice Management and Resource Orchestration components

Figure 5 shows how the Slice Management & Resource Orchestration layer requests information from the external components.

- A “slice intent” request from the nApp Orchestration and Development Layer triggers the slice selection and quota allocation functions described above, through the internal mechanisms to select the most suitable Network Slice Template from the NEST catalogue implemented in the E2E Resource Orchestration for Far Edge Enabled nApps.
- The integration with the MANO Layer is implemented through the combination of the node availability notification (to keep track of the devices that can be used to instantiate the services) and the quota and namespace allocation when a slice intent request contains a set of computing constraints.

4.1. Design details of the Slice Manager and Resource orchestration layer

4.1.1. Design details of the E2E Resource Orchestration for Far Edge enabled nApps

The E2E Resource Orchestration for Far Edge enabled nApps (RO) takes care of the resource allocation across the Far Edge to Cloud continuum. It is fed with the Service Intent request from the Slice Intent Handler (see Figure 5) and it manages the request by allocating the most appropriate namespaces and quotas to the Far Edge devices according also to the resource availabilities provided by the OBU/Edge/Cloud Continuum Resource Inventory (CCRI or RI). It works in conjunction with the Slice Manager to return to the nApp Orchestrator & Development layer (NOD) a full description of the computing and networking resources to be used to instantiate the service on the target segment and host.

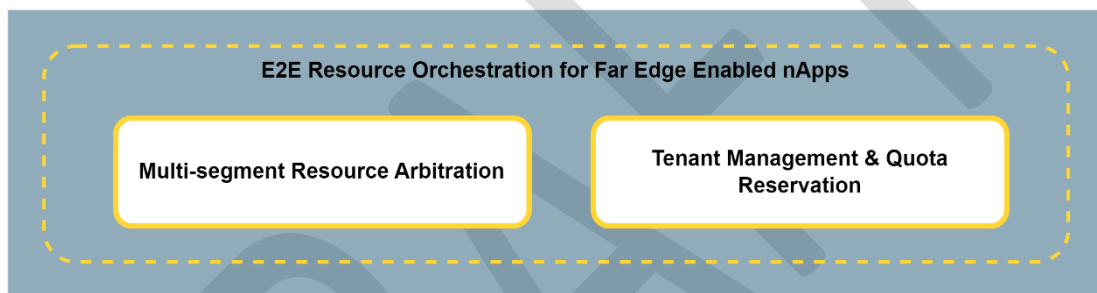


Figure 6: Internal Architecture of the E2E Resource Orchestration for Far Edge enabled nApps

The E2E Resource Orchestration for Far Edge enabled nApps is comprised of two software submodules:

- **Multi-segment Resource Arbitration:** This function is responsible to coordinate the provisioning of computing resources by translating the information received from the nApp Orchestrator & Development Layer (specifically from the Slice Intent) into a set of computing constraints. Starting from the computing requirements and combining this information with the set of available Edge and Far Edge resources obtained from the CCRI, the Multi-segment Resource Arbitration is able to build a “Quota Reservation Request” to the Tenant Management & Quota Reservation component to apply the operation to the actual devices.
- **Tenant Management & Quota Reservation:** It is responsible to implement the client interface to the MANO layer and then to apply the action produced by the Multi-segment Resource Arbitration. Multitenancy allows to deploy multi-vendor services to the same Edge and Far Edge resources, enabling the possibility to exploit the platform by multiple automotive vendors.

Module's Interactions

Table 7, E2E Resource Orchestration interactions

Interacting Module	Description
Slice Intent Handler (nApp Orchestration and Development Layer)	Translation of Intent high-level QoS requirements and constraints into the most suitable 5G Service Profile
MANO	Instantiation of the quotas and namespaces on the Far Edge devices cluster

4.1.2. Design details of the OBU/Edge/Cloud Continuum Resource Inventory

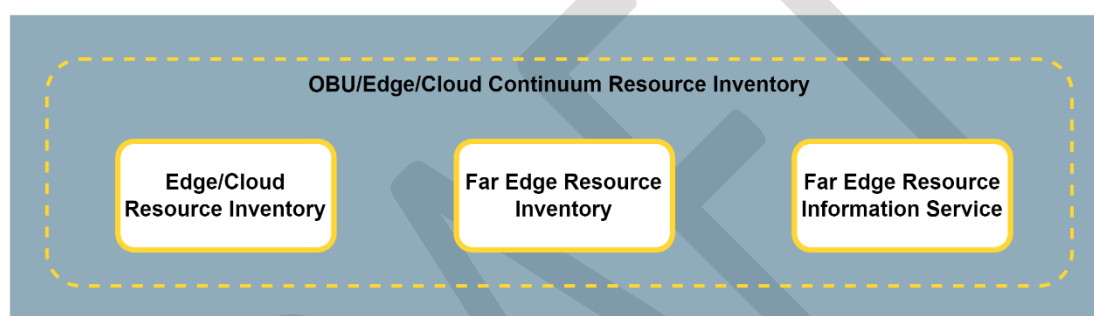


Figure 7: OBU/Edge/Cloud Continuum Resource Inventory internal structure

The OBU/Edge/Cloud Continuum Resource Inventory (CCRI or RI) internal structure is reported in Figure 7. It is based on three internal software modules, whose main functionalities can be explained as follows:

- **Far Edge/Edge/Cloud Resource Inventory:** These two components which implements a continuously updated registry of the available resources (from the Far Edge to the Cloud) and their static capabilities. It could be queried by the components who aim to receive a complete view of the status of the 5G-IANA devices ready to host nApp or vertical services. Moreover, it could work in conjunction with the Far Edge Resource Information services to enrich the device information with dynamic data (like the location of the device itself or the remaining computing resources). This enrichment makes the Resource Inventory able to expose filtering mechanisms to the query received by the external components.
- **Far Edge Resource Information Service:** It relies on external monitoring platforms to enrich the information exposed by the Far Edge/Edge/Cloud Resource Inventory with dynamic data. The component acts as a client for the Monitoring Service (described in Section 3.1.6) to correlate the availability information stored in the Resource Inventory with dynamic information (like the location of the Far Edge

resource) to expose to the external components (mostly the DMLO) a filtering-based interface for the Far Edge Resources. This could be used to select the most appropriate OBUs based on some refined criteria (for instance, location based or computing resource available based criteria).

Module's Interactions

Table 8, OBU/Edge/Cloud Continuum Resource Inventory interactions

Interacting Module	Description
Monitoring & Analytics	To constantly be updated about dynamic Far Edge Resource's information (like the location)
MANO	To receive notifications about the availability of the Edge and Far Edge resources over the time
DMLO	Periodically responding with information about the candidate node set for the next Federated Learning client selection process (see Section 6).

4.1.3. Design details of the Slice Management

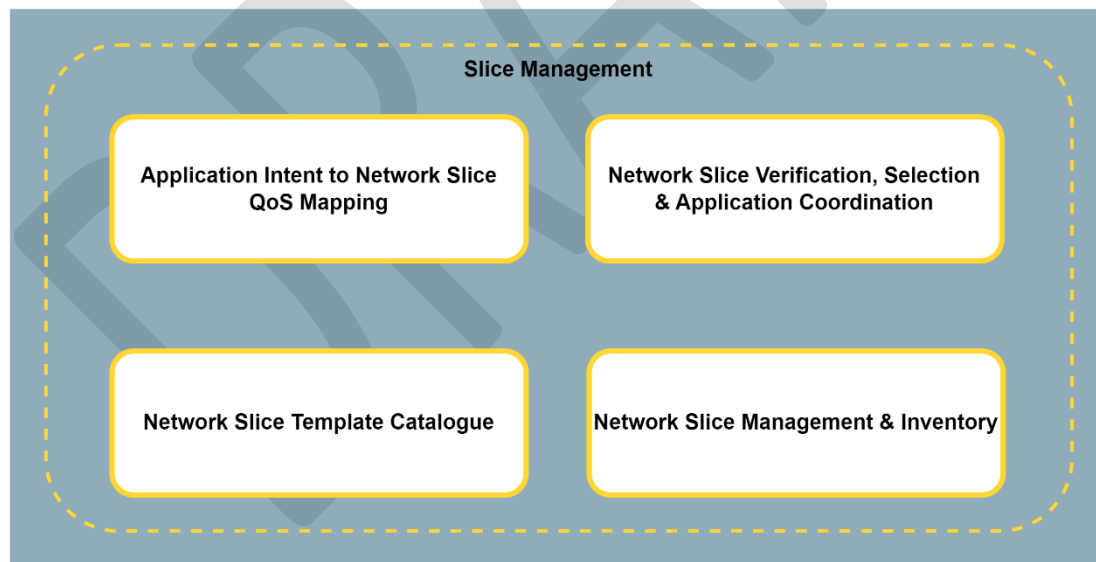


Figure 8: Slice Management internal structure

The Slice Management component implements the functionalities for storing the available Network Slice Instances (NSIs) and validating Service Provider requests for Vertical Services in terms of compatible 5G service profiles. According to Figure 8 it is composed of four internal functions:

- **Application Intent to Network Slice QoS Mapping:** It receives the Slice Intent from the Slice Intent Handler (depicted in Figure 5) and, relying on a set of translation rules, it is able to convert this information to some recommendations about the most suitable Network Slice Instance available on the system. This module, in particular, uses the Networking Constraints specified in the intent-based request to select the most appropriate NEST for the instantiation of the 5G Network Slice, through the 5G Network Slice Management Function, for the requested Vertical Application Slice. To select the NEST, the module interacts with the NEST catalogue of the 5G Network Slice Management Function module to get the NEST available and then maps the Networking Constraints to retrieved NESTs to select the most appropriate one.
- **Network Slice Verification, Selection & Application Coordination:** It is able to choose the 5G Network Slice from the Network Slice Management & Inventory based on the set of recommendations received by the Application Slice Intent to Network Slice QoS Mapping component. Verification is performed based on the networking requirements received by the Slice Intent Management module: the module verifies whether there exists an appropriate Network Slice Template that is able to meet all the requirements.
- **Network Slice Template Catalogue:** The component which contains a static inventory with the Network Slice Templates available on the system.
- **Network Slice Management & Inventory:** It implements a static database with a set of static 5G Network Slice Instances to be selected for the instantiation of the Network Services.

Module's Interactions

Table 9, Slice Management interactions

Interacting Module	Description
E2E Resource Orchestration for Far Edge Enabled nApps	Translation of Intent high-level QoS requirements and constraints into the most suitable 5G Slice Instance

4.2. Slice Manager and multidomain orchestration layer developments

The E2E Resource Orchestration for Far Edge enabled nApps and the Slice Management components have been developed and extended from the Slice Inventory & Resource Orchestrator module developed during the Int5Gent EU Project. The component is a python-based software application that leverages the Flask framework to implement a REST-full web server in order to expose all the endpoints needed by the nApp

Orchestration and Development (NOD) and implements the logic to manage the lifecycle management of Vertical Application Slices and so to manage the lifecycle management of 5G Network Slices through the 5G Network Slice Management Function module and to manage the resource quotas in various Kubernetes clusters/VIMs for the Vertical Application components. The modules have been extended to be compliant with a new set of requirements of the 5G-IANA platform. Planned extensions involve:

- the enhancement of the mapping mechanism to support additional QoS parameters, the implementation of a verification mechanism to assess the presence of a suitable running NSI that matches the selected NEST,
- the implementation of a RESTful interface to enable the management of NSIs by the Network Operator
- the modelling of tenants' SLAs and profiles to accommodate constraints related to the usage of Far-edge resources (i.e., OBUs, RSUs).

The OBU/Edge/Cloud Continuum Resource Inventory, instead, has been developed from scratch to fulfil the 5G-IANA requirements. The first release of the components implements all the main functionalities exposed above except the mechanisms to query the inventory based on some filtering rules, that are yet to be defined according to the requirements collected from the other platform components (see Section 6).

4.3. Planned extensions in the second development cycle

The OBU/Edge/Cloud Continuum Resource Inventory will be extended according to the requirements that will be collected during the development and implementation of the monitoring mechanisms in conjunction with the ones related to the Distributed Machine Learning Orchestrator. The main extensions will be related to the implementation of filtering mechanisms on the Resource Inventory query interface, as well as the internal communication with the Monitoring & Analytics (Table 7) to keep the Far Edge Resource Information Service updated over the time.

5. VIRTUALISED INFRASTRUCTURE SEGMENTS

5.1. On-vehicle MANO, Edge and Cloud MANO

The use of Kubernetes has been judged the most suited for the orchestration of virtualized applications in the far-edge segment (i.e., OBU, RSU) and the edge and cloud segments. This choice satisfies the On-vehicle MANO (OVM) requirements (from SFR-OVM-1 to SFR-OVM-7) and Edge and Cloud MANO requirements (from SFR-EAC-1 to SFR-EAC-5) that have been introduced in the 5G-IANA deliverable D2.1 “Specifications of the 5G-IANA architecture”.

Kubernetes provides several features, such as scalability, resilience and portability, that match with the requirements of the 5G-IANA context. Kubernetes grants scalability to applications, allowing them to scale up or down the number of working nodes (i.e., the number of container’s replica) based on the runtime requirements (e.g., CPU load). This allows to never lose performances or waste resources allocating more working nodes than needed. **The far-edge devices are hardware-constrained devices that require a lightweight solution to be capable to work in a distributed architecture.** Another key feature of Kubernetes is the resilience that permits the applications, that have been deployed in a far-edge device, to keep running even if the device experiences a failure in the network connection. Furthermore, the portability of Kubernetes allows to use it in different environments such as far-edge and edge/cloud segments. This aspect can simplify the management as just one MANO solution can be used.

Kubernetes is available in several distributions. In 5G-IANA, the MicroK8s distribution,^[19] which is a Kubernetes distribution certified by Canonical, has been selected. It provides the main features of Kubernetes and it is designed to be lightweight and easy to install. These specific characteristics made MicroK8s the optimal choice for the MANO in 5G-IANA as devices with limited hardware capabilities (e.g., ARM-based) are the main target of the project. It is indeed preferable to have a solution that does not exhaust the available computational resources.

The main Kubernetes distribution was not selected since its implementation is expected to require more hardware resources with respect to MicroK8s. Other distributions of Kubernetes are minikube ^[20] and K3s ^[21]. Both of them have been analysed, but they have been reputed as less suitable with respect to MicroK8s. Minikube was discarded because it is designed to be used mainly for testing/developing purposes and is not a complete product. K3s could be a valid alternative. It is very similar to MicroK8s in terms of computational

resource requirements and features exposed. MicroK8s was preferred over K3s because it seems to provide more mature and complete software.

The deployment of MicroK8s on the devices needs to perform some preliminary steps before installing the MicroK8s that can be done following the available documentation. The MicroK8s, and in general Kubernetes, orchestrates containers, thus the far-edge device needs to be configured to support containerized applications. In the case that the far-edge device is provided with a GPU, both the container engine and MicroK8s need to be configured for the use of the GPU.

The deployment and the management of the deployed services on the onboard / roadside units is handled by the nApp Orchestration & Development layer through its built-in ability to perform deployments in various virtualization environments. Hence, the Deployment Manager relies on an **abstract interface for basic management operations for the Microk8s APIs**.

5.2. Information and localization service developments

The information and localization service provides far-edge device's hardware capabilities and its current position to the 5G-IANA platform (addresses SFR-GVI-3).

The information about the hardware capabilities (e.g., total CPU, total memory, GPU availability) of the far-edge device is retrieved by the 5G-IANA platform from the APIs that MicroK8s makes available. This solution is then to rely on the already available interfaces of MicroK8s. The far-edge device and the Resource Inventory interact by exploiting these interfaces.

The position of the far-edge device is provided to the 5G-IANA platform from a service that has been developed within the project that is named **localization service**. Figure 9 illustrates the workflow of this service.

The localization service interacts with the GNSS receiver of the OBU for retrieving the current vehicle's position, while this information on the RSU is given as an input parameter based on the RSU installation position. The localization service provides the position information to a Prometheus instance on the edge server. The Resource Inventory retrieves the position information of far-edge devices by using the interfaces that are made available by the Prometheus instance on the edge server.

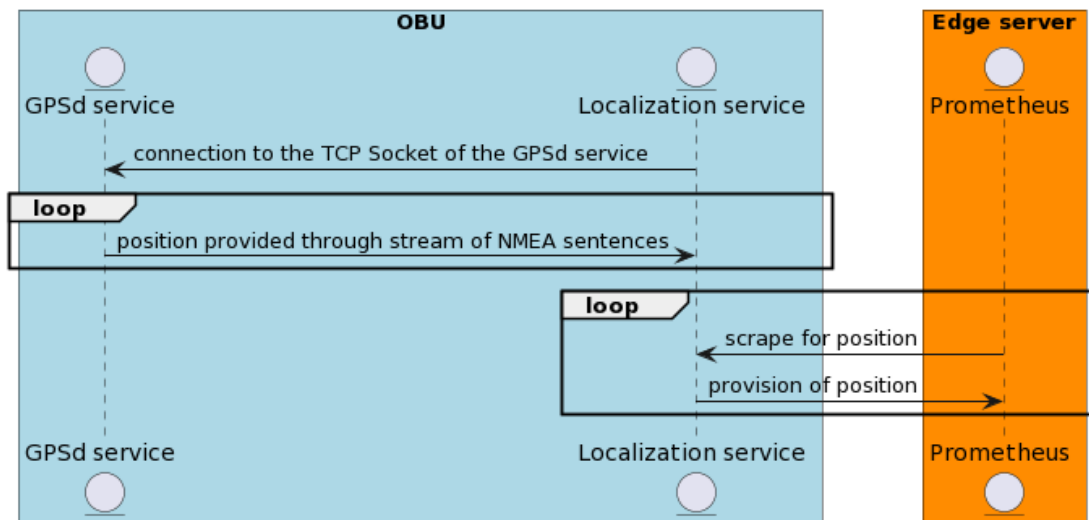


Figure 9: Localization service workflow

The interaction of the localization service with the GNSS receiver is performed using the GPSd service [22]. The localization service is deployed as a system Linux service on the far edge devices. The localization service retrieves the position from the GPSd using a TCP socket. GPSd supports two different operating modes. The monitor mode exploits the GNSS receiver for providing real data, while the simulated mode uses pre-recorded or manually crafted tracks saved in NMEA files. By default, if a GNSS receiver is plugged in and available, the localization service retrieves the true position information. The simulated mode is also made available to third-party experimenters thanks to an interface implemented with RESTful APIs. The third-party experimenters can exploit these APIs for configuring the simulated mode of GPSd and uploading, deleting, starting, or stopping the simulations of vehicle movement. The localization service provides the simulated position when the GPSd service is activated in the simulated mode. This mode can be used on the physical OBUs for testing some nApp without requiring having a real vehicle to drive around.

5.3. Resource monitoring agent developments

The Resource Monitoring Agent (SFR-GVI-3) has been implemented to collect and provide the information about the status of virtualized resources in the far-edge devices. The Resource Monitoring Agent is made of two software modules: the Hardware Monitoring Agent and the Data and Metrics Exporter. The workflow of the Resource Monitoring agent is depicted in Figure 10.

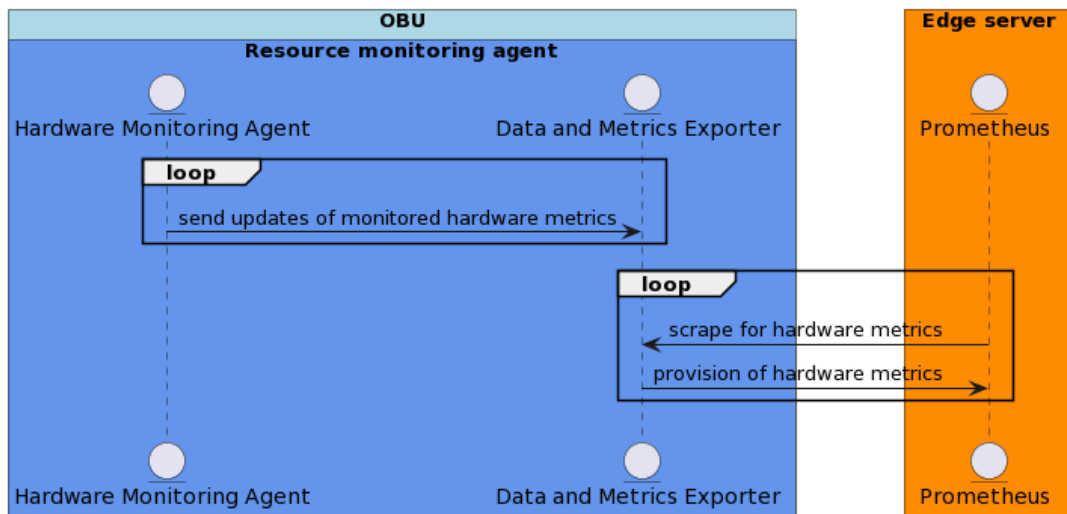


Figure 10: Resource monitoring agent workflow

The Hardware Monitoring Agent is a software component that periodically checks in runtime the utilization of the far-edge resources (e.g., CPU utilization, memory usage, storage used). Different versions of this agent can be used depending on the hardware of the far-edge devices. The main difference is the presence of a GPU on the far-edge device. If a GPU is present, specific commands need to be used for retrieving metrics that are related to the GPU utilization. This software agent provides the collected information using a JSON formatted message to the Data and Metrics Exporter using a RESTful APIs that has been implemented.

The Data and Metrics Exporter is in charge of processing the received metrics and providing them to a Prometheus instance running on the edge server. This software module makes also available the RESTful APIs to AFs that want to collect data through the monitoring platform of 5G-IANA. The Exporter sends the AFs data and metrics to the Prometheus instance on the edge server that forwards them to the monitoring platform in the cloud. The Resource Monitoring Agent interacts in particular with the Distributed Data Collection component of the 5G-IANA platform through the interfaces that Prometheus makes available.

5.4. Far-edge devices developments

The far-edge devices (i.e., the OBUs and the RSUs), that have been developed in the 5G-IANA project, are based on embedded devices (i.e., ARM-based) hardware. These devices have been equipped with an Operating System (OS) that supports virtualization technology and GPU abstraction. The virtualization support makes feasible containerization approach that is essential for the orchestration operations based on Kubernetes.

The boards have been selected (addressing SRNF-GVI-1) to provide an amount of resources (e.g., availability of GPU, storage) that satisfies the needs of the use cases to be demonstrated within 5G-IANA. In particular, a GPU is available on OBU, RSU and at the edge server. The GPU availability makes feasible to process sensors' data.

The far-edge devices can provide a network communication channel (requirement SFR-GVI-4, SFR-OBU-1 and SFR-RSU-1) since both OBUs and RSUs are equipped with a 5G modem providing the Uu network interface.

The OBUs and RSUs boards are provided with a set of external physical interfaces to support external sensors as required by the 5G-IANA use cases (i.e., SFR-OBU-7 and SFR-RSU-2 requirements). The RSU provides three Ethernet interfaces and a USB interface. The OBU provides an Ethernet interface for connecting with the vehicle network (requirement SFR-OBU-5) and it is equipped with a Wi-Fi interface that can create a Wi-Fi hotspot to be used by mobile devices of the vehicle occupants (requirement SFR-OBU-3). The OBU does not provide a Bluetooth interface (requirement SFR-OBU-4) and it does not currently have a CAN-bus interface (requirement SFR-OBU-6) as it is not required by the available vehicles in the 5G-IANA project. A CAN-bus interface can be added in the future if needed. Furthermore, the OBU is equipped with a GNSS RTK receiver (requirement SFR-OBU-2) for retrieving an accurate vehicle position.

5.5. Future extensions

Further extensions of the modules, which have been introduced in the current section, have to be defined after the integration and testing phases. Possible extensions will also address needs that will arise during the experimentation of the platform. For example, a possible extension of the Information and Localization service could be to provide additional metrics such as the speed and the heading of the vehicle.

6. CROSS LAYER FUNCTIONALITIES

6.1. Distributed AI/ML framework mechanisms

The design and development activities for the Distributed AI/ML orchestration layer have focused on the support of a *multi-criteria client selection* operation, for the particular case of **Federated Learning** workloads. As described in D2.1, this corresponds to the support of **FL** applications in dynamically adapting the working set of training nodes i.e., OBUs, subject to a series of dynamically varying parameters including compute/storage resource availability, energy availability, location, data availability and quality. The delivery of this functionality has the following main targets. First, the acquisition of application-agnostic selection criteria information is decoupled from the application at hand, and instead supported by the 5G-IANA platform. This in turn renders the 5G-IANA platform as a FL application enabler, substantially simplifying and therefore facilitating application development. Second, the client selection decisions are realized through life-cycle management and orchestration primitives, essentially allowing OBU resources to be reserved and consumed only when necessary, i.e., a federated learning client AF is not instantiated unless it is selected for the current pool of FL clients. This comes in sharp contrast with the current state of affairs, where a FL client is constantly up and running, regardless of its inclusion in the current pool of FL clients. It is noted that this benefit stems directly from the translation of AI/ML pipelines into nApp service chains (D2.1).

Figure 11 presents the overall process for the support of the Client Selection functionality. The overall functionality is supported through: (i) the Distributed Machine Layer Orchestrator (DMLO): this is a component of the 5G-IANA platform presenting a DML-service specific user interface to the platform user, along with a backend component for the interaction with the various other components of the architecture, as well as the corresponding Application Functions (AFs); (ii) the involved AFs, namely: the AggrNode AF, which presents the FL aggregation node functionality; the Training Node AF, which presents the FL client functionality; and the Data Collection AF, which is responsible for the collection of data to be used for training purposes^[23] The AggrNode and Training Node AFs are containerized versions of the aggregator server and training client components of the FLOWER framework^[24] These AFs are provided by the platform, which will further offer the ability to vertical service providers to customize e.g., by uploading their ML model, configuring the overall service (see next, as well as Section 6.1. and Section 6.3).

In all, the set of OBU devices that corresponds to the current set of FL training clients is identified based on the following distinct categories of criteria / characteristics:

Static criteria: this category refers to non-varying characteristics of the candidate OBU/devices namely: (i) access rights, that essentially identify the rights of the vertical service provide on the device e.g., vehicle is part of an OEM fleet; (ii) Hardware capabilities, which refers to both processing capabilities e.g., GPU availability, CPU/RAM/Storage capacity, and sensing capabilities i.e., LIDAR availability, access to CAN bus information, etc. As discussed in the following, these selection criteria are applied at the preliminary (AI/ML-agnostic) deployment (placement) face of the overall orchestration process.

Dynamic ML-agnostic criteria: this category refers to dynamically varying status information of the device at hand, namely: (i) utilization of CPU/GPU, RAM, storage; (ii) energy availability^[25]; (iii) location information. This information is not directly related to the ML-nature of the orchestrated services and it is collected and managed mainly by the Monitoring and Analytics component of the platform, directly from the UEs (OBUs). The Compute Continuum Resource Inventory (CCRI) is responsible for retrieving all monitoring information related to the OBUs at hand, as described below.

Dynamic ML-specific criteria: this category refers to criteria expressing the likelihood of the device at hand to efficiently contribute to the ML training process. This corresponds to two types of information:

Dynamic ML-specific (Data) criteria: data availability information, expressing the volume of data available at the node for training purposes; optionally this can be augmented with data quality information, expressing the completeness of the (often poly-parametric) training data records i.e., availability of all desired training features;

Dynamic ML-specific (KPI) criteria: ML-performance information, expressing ML KPIs such as observed convergence rate, achieved (local) accuracy, etc. Data availability information is collected and maintained by the DMLO, since, as explained above, service chaining allows the decoupling from the training operation. ML-performance information is maintained and used by the AggrNode, as this is tightly related to the application/training task at hand (see also next).

The first step (**Step 1**), corresponds to the initial configuration and deployment of the main AFs of the FL service. The DMLO user interface is employed to collect ML-specific configuration information and artefacts including the ML model, training scheme i.e., synchronous vs. asynchronous, number of clients per training round. In this step, the baseline (ML-agnostic) functionality of the platform is employed to deploy the AggrNode and Data Collection AFs. The placement of these AFs follows general placement capabilities (see section 6.1) i.e., it is not associated with client selection. In practice, this means that the AggrNode is

instantiated at a (server) node with suitable resource availability, while the Data Collection AF is instantiated at the set of vehicles designated through the *Static criteria*. Step 1 completes with the configuration of the AggrNode AF, using the input provided to the DMLO (see above). It is noted that no Training Node AF is instantiated at this stage.

In **Step 2**, the DMLO communicates with all deployed Data Collection AFs in order to collect *Dynamic ML-specific criteria* information. Currently, the design of this step foresees periodic, blocking requests from the DMLO to the individual Data Collection AFs. It must be noted that this information is currently collected by the DMLO itself, since it is related to ML-specific workloads i.e., it is not expected to be generated by non-ML-related applications, and as such the project does not currently foresee the integration of such training data monitoring functionality in the broader monitoring capabilities of the platform.

In **Step 3**, the DMLO contacts the CCRI so as to request the current set of suitable nodes/devices according to the *Dynamic ML-agnostic criteria*. The CCRI is then responsible to match these criteria against the continuously updating monitoring information, which is retrieved from the Monitoring & Analytics component, as shown in Figure 11. Again, the current default operation involves a periodic pull operation of the updated node set currently matching the selection criteria. With the completion of Step 3, the DMLO holds up to date information regarding suitable nodes with respect to non-ML-specific criteria.

Step 4 is responsible for the joint node/device filtering subject to both ML-agnostic and ML-specific criteria i.e., by combining the collected information, the DMLO identifies the set of nodes fulfilling all client selection criteria. This step follows each update of the information received in steps 2 and 3.

The following steps correspond to the synchronous operation mode of FL, where the training process proceeds in iterations, as described in D2.1. In **Step 5**, the AggrNode contacts the DMLO so as to retrieve the most up-to-date set of nodes fulfilling the *Static criteria*, the *Dynamic ML-agnostic criteria* and the *Dynamic ML-specific (Data) criteria*. Using potentially available information regarding *Dynamic ML-specific (KPI) criteria*, the AggrNode makes the final selection in **Step 6**. The selected set of nodes is communicated to the Policy Execution (through the DMLO), which is responsible for instantiating the corresponding Training Node AFs at the selected OBU. At this stage, and upon confirmation of the successful completion of this step, **Step 7** commences, which corresponds to the default training round of FL i.e., the model is dispatched to all (selected and instantiated) clients, trained locally and finally collected back and aggregated at the AggrNode. **Step 8** focuses on visualizing status reports at the DMLO user interface level, by collecting statistics related

to the ML process i.e., achieved accuracy/convergence rate, number of completed epochs, etc. (retrieved from the Aggr Node), as well as to the node resource utilization e.g., CPU/GPU utilization (retrieved from the Monitoring & Analytics). At the end of each round the AggrNode checks whether it should terminate the training process, subject to application-specific criteria e.g., convergence rate/ training accuracy achieved. Service termination is notified to the DMLO (**Step 9**), which propagates it to the Slice Manager (not shown).

Module's Interactions

The Distributed Machine Learning Orchestration module interacts with the following software modules of the 5G-IANA platform.

Table 10, DMLO interactions

Interacting Module	Description
Slice Manager	Notified that ML-specific configuration information has been collected by the DMLO and the service can be deployed.
Compute Continuum Resource Inventory	Periodically contacted by the DMLO to provide information about the candidate node set for the next client selection process.
Policy Manager	Instructed by the DMLO to instantiate a Training Node AF at the designated OBUs.
Monitoring & Analytics	Contacted to provide resource consumption monitoring information for display purposes (combined with ML-level statistics)

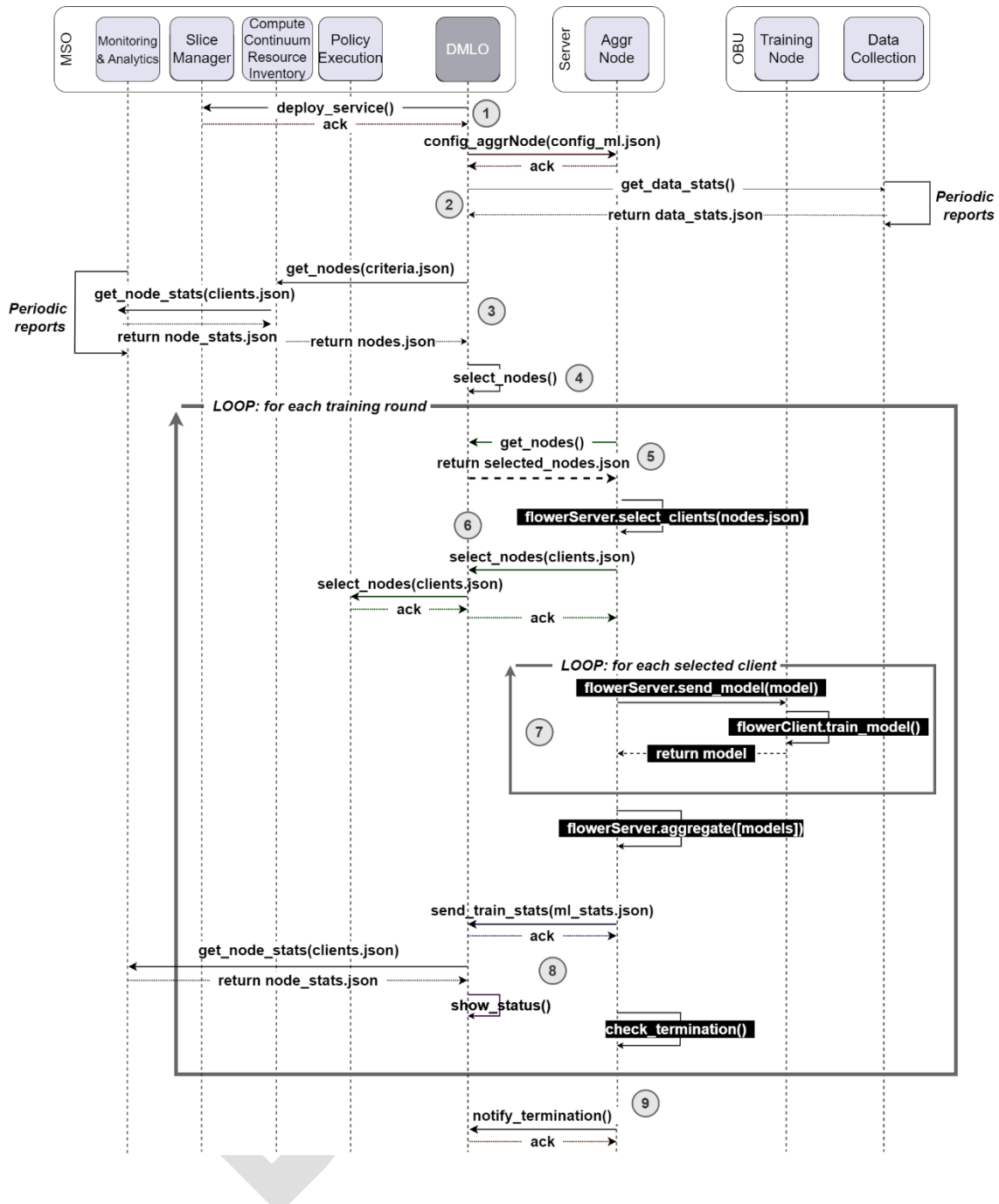


Figure 11: Distributed Machine Learning Orchestration (DMLO): Client Selection operation

6.1.1. AFs and FLOWER integration

The FLOWER framework handles all FL task specific operations included in FL aggregation node i.e., the FLOWER server (implemented in the AggrNode AF) and the FLOWER client (implemented in the Training Node AF), along with their communication (functions marked with black in Figure 11).

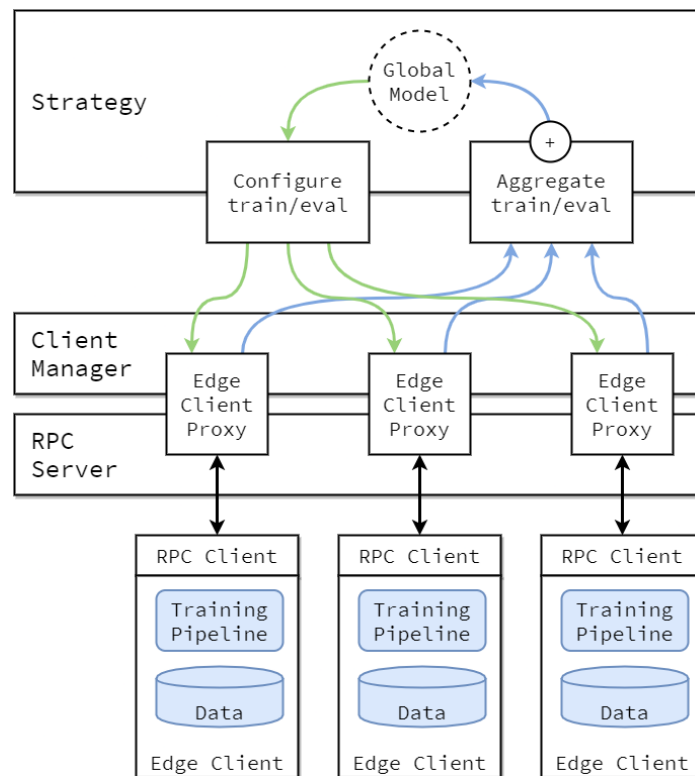


Figure 12: FLOWER framework component architecture

The FLOWER server consists of the following components (see Figure 12) the FLOWER strategy, which is an abstraction that allows for full customization of the ML process i.e., configuring the respective FL-related parameters e.g., client selection method, number of per round participating clients, model aggregation method, accuracy evaluation metric, etc. 2) the FLOWER client manager, responsible for monitoring the FLOWER clients at all times and 3) the FLOWER RPC server that handles server-client communication using Google’s Remote Process Calls framework (gRPC) over Hyper Text Transfer Protocol (HTTP). These components are instantiated and configured in the FLOWER server during FL-task configuration (see step 1, Figure 11), according to the configuration file (see config_ml.json Figure 11) of the DMLO request.

Upon configuration, the FLOWER initiates the training process. In each training round, the FLOWER server queries the DMLO (via a REST API) in regards to the available nodes/devices (see Step 5, Figure 11). Note that the DMLO keeps track of the nodes, according to dynamic ML-agnostic criteria, as described in Sec. 6.1. The DMLO’s (REST) response is a list of node IPs. The FLOWER server performs client selection thereafter (see Step 6, Figure 11). Client selection is an operation that is internal to the FLOWER framework, which is controlled by ML-specific criteria, set during the configuration phase, described above. The selected clients list is sent back to the DMLO (see Step 6, Figure 11) via another REST call, so that the respective FLOWER

clients (implemented in the Training Node AF) can be activated via the Policy Manager. During that phase, the FLOWER server is waiting for an acknowledgment. Upon receiving the acknowledgment (API response) from the DMLO, the FLOWER server activates the training process in the clients (see Step 7, Figure 11). The activation of the training process is an internal FLOWER process, which only requires the list of the involved client IPs. Server-client communication is handled by the FLOWER RPC server, control of the process by the FLOWER strategy and finally local client training by the FLOWER client component.

The FLOWER client (referred as Edge Client in FLOWER – see Figure 12) includes: 1) the training pipeline, which accounts for the (local) model training in each client, 2) the data used for training and 3) the RPC client-side, to communicate with the FLOWER server. These components are set during service deployment in the respective Training Node AF (see function “deploy_service” in Figure 11). Note that in our case, data does not reside in the FLOWER client; it is acquired in a real-time manner from the Data Collection AF in each OBU (see Figure 11).

FLOWER provides a baseline FL framework but does not implement the involved FL functionalities e.g., training, aggregation, model monitoring, etc. It is rather used as a facilitator, allowing for flexible development and integration (dockerization in our case) thereafter. Specifically, FLOWER implements a wide array of wrappers that control the above-mentioned functionalities as well as the underlying communication medium between FLOWER server and the respective clients. By adjusting, configuring (or even overriding) these wrappers, we are able to control the respective training process, according to the DMLO’s input. It is worth mentioning that FLOWER operations are limited to ML-specific tasks such as training and does not account for more generic tasks such as client connectivity, client data availability, etc., which are performed by the DMLO.

6.1.2. Distributed AI/ML orchestration layer developments

The development of the DMLO component and corresponding functionality have been planned for Release 2 of the 5G-IANA platform. As such, development is currently in progress.

6.1.3. Planned extensions in the second development cycle

The functionality described in Section 6.1 constitutes the baseline for the Distributed AI/ML framework. A series of extensions are planned on top of this baseline, as described in the following.

Asynchronous operation: currently Steps 2 and 3 are performed with blocking, pull-based calls performed by the DMLO. The project considers the refactoring of this operation to an asynchronous mode realized by a Pub/Sub (push-based) interface. This would allow the CCRI and Data Collection components to provide their information subject to actual updates matching the selection criteria, triggered by the actual change of the monitored state, rather than the call by the DMLO (non-blocking). This can include notifications regarding specific events e.g., suitable OBU entering a designated area, in a fashion similar to 3GPP SEAL²⁶. In a similar fashion, considered extensions further include the support of asynchronous FL operation (see also D2.1).

ML-scheme/topology selection: as investigated in [23],[24],[25], the selection of the ML training scheme e.g., centralized training vs. federated learning, is a multifaceted task associated with aspects such as resource/energy consumption, training performance (achieved accuracy), privacy constraints. While strict privacy constraints dictate a federated learning approach, privacy elasticity may leave room for other schemes e.g., centralized learning, where all data is aggregated at a central location where training takes place. Based on recent work [27], the project will explore the possibility of providing decision support in what concerns ML-scheme/topology selection.

6.2. Monitoring & Analytics, Distributed Data Collection

This section describes the design behind the **Monitoring & Analytics including also data collectors** that are used to extract and report metrics from the different hardware and the virtualization technologies used. Furthermore, Monitoring & Analytics section includes the Distributed Data Collection design and development details.

6.2.1. Design Details of Monitoring & Analytics

The aim of Monitoring & Analytics is to collect and make readily available monitoring data regarding resource utilization from the underlying virtualized infrastructure including the on-vehicle MANO (e.g., compute, memory, network - see Section 5.3) and deployed nApp's behaviour from tailored application-level metrics (e.g., throughput, active users). The Monitoring & Analytics enables core services (e.g., Orchestration, Policies, Profiling) and is responsible to collect data based on a set of active monitoring probes, as well as to support a set of data management operations (e.g., calculation of average values in specific time windows).

It also provides access to historic and real-time monitoring data that can be centrally accessible by tenants through the 5G-IANA Dashboard, and subsequently through the respective API (Prometheus-API [28]). The

"metrics collection" model of Monitoring Engine follows a distributed approach by fetching and processing monitoring data/requests regarding a specific deployed nApp over the overlay mesh network that interconnects the respected nodes of the running nApp.

6.2.2. Monitoring & Analytics Developments

The monitoring engine relies on an industry-grade telemetry system: Prometheus [28]. To populate the Prometheus database with monitoring metrics, Netdata [29] is used as a monitoring agent. Netdata provides real-time streams that are collected and managed by the Prometheus monitoring engine.

The reported developments concern the different monitoring instances (cloud, edge) for collecting both on-vehicle MANO metrics (see section 5.3) as well as application metrics. The far-edge devices interact with the Prometheus instance at the edge server. Further details about development of the monitoring at the far edge are provided in Section 5.3.

Module's Interactions

The **Monitoring & Analytics** interacts with the following software modules.

Table 11, Monitoring & Analytics interactions

Interacting Module	Description
Policies	Consumes the Monitoring HTTP API for collecting specific metrics under the user selection and uses them to construct rules.
Profiling	Consumes application-related data acquiring specific timeseries by applying filtering queries
Resource Monitoring Agent (on-vehicle see section 5.3)	Pulls metrics from two sources included in the OBU/RSU Monitoring Agent: a. the Hardware Monitoring Agent and b. Data and Metrics Exporter.
Resource Inventory	Resource Inventory scrapes metrics from Monitoring reported by the Resource Monitoring Agent (see above)
Distributed AI/ML Framework	Consumes resource monitoring information

6.2.3. Planned extensions in the second development cycle

Monitoring & Analytics hosts a) application metrics and is extended to also host b) infrastructure metrics (including also OBU/RSU related metrics, see Section 5.3). This extension will likely imply an enlargement of monitored data set that can be correlated with specific policies. As it is now, Monitoring & Analytics is mostly limited to cloud-based parameters (such as the usage of virtual computing and storage resources), while until the second release of 5G-IANA developments it is expected to host **cloud-based, custom use case specific as well as infrastructure-based parameters (e.g., actual QoS measures on networking services, mobility)**.

DRAFT

7. INTERFACES AND WORKFLOWS

The integration of the overall AOEP requires a number of interfaces between the layers described in the previous sections. These interfaces are described in the following section.

7.1. Design Details of the interfaces

Figure 13 depicts the interfaces between each particular layer. The functionality of each interface is described below.

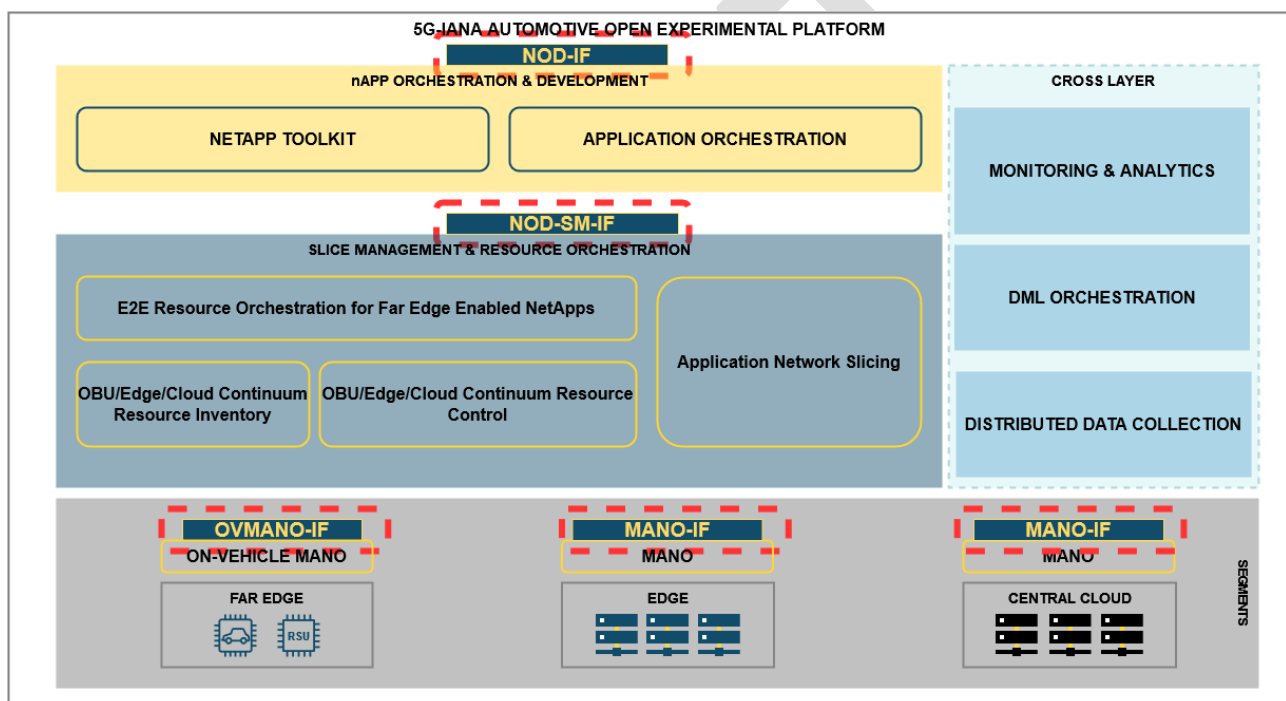


Figure 13: AOEP interfaces

NOD-IF: the interface between the end users of the platform and the nApp Toolkit.

NOD-SM-IF: the interface between the nApp Orchestration & Development layer and the Slice Management & Resource Orchestration and describes the programmatic actions supported between these two layers.

ONMANO-IF: describes the interface that is used to access the services deployed on the on-board / roadside units from the nApp Orchestration & Development layer.

MANO-IF: describes the programmatic actions supported between Slice Management & Resource Orchestration layer and the control plane of Kubernetes running on cloud and edge static (static refers to location static i.e., not moving) servers.

7.2. Interfacing Developments

7.2.1. NOD Interface

NOD-IF: This interface is hosting the actions supported by the nApp Toolkit with the end-user and they are described in Deliverable 4.1.

7.2.2. NOD-SM Interface

NOD-SM-IF: This interface hosts the **Slice Negotiation phase** (described in Section 3.2.1.4) between the **nApp Orchestration – Slice Management and Resource Orchestration layers** whose purpose is to handle the complete lifecycle of Network Slice creation. As described in Section 3.1.4, the lifecycle of a slice includes a) the slice request phase, b) the slice instantiation, c) the slice operation and d) the slice deprovision. Initially, the Slice request phase is triggered from the NOD layer in the form of a **Slice Intent**. **As already mentioned, the Slice Intent is a metamodel for transferring information such as the set of declared computational, networking (in terms of network services) and QoS requirements per nApp on behalf of the nApp provider.** The aforementioned set of requirements have to be fulfilled by the infrastructure in order to guarantee specific KPIs. During the instantiation phase, the creation and verification of the “proper” environment is performed, which will be used to support the lifecycle of the nApp. The term “proper” is defined as the tuning of the programmable resources both computational and networking (access or transport network, SDR, FPGAs, SDN switches) in order to accommodate the user-given QoS values. The translation of the requirements to a proper slice configuration plan is a primary objective of the Slice Negotiation. During the instantiation phase, all resources shared/dedicated to the slice should have been created and configured.

7.2.2.1 Slice Negotiation Workflow

The slice negotiation phase is depicted in the following flow diagram. It is important to state that the process only focuses on the communication between the two layers and not in intra-layer communications.

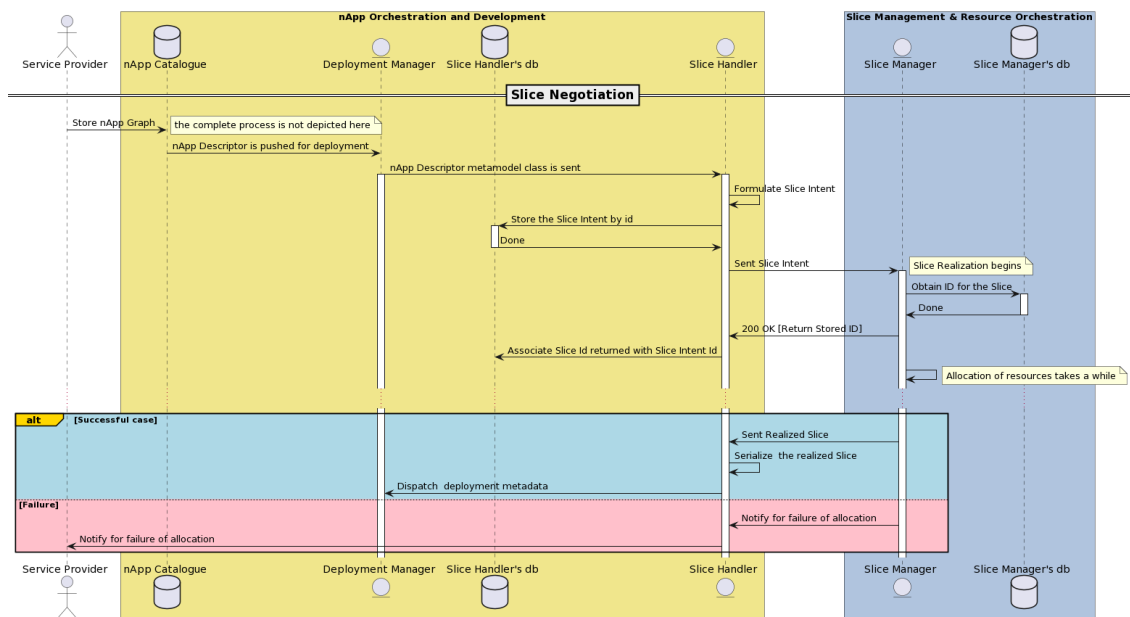


Figure 14: nApp Orchestration & Development – Slice Management & Resource Orchestration interaction flowchart

The nApp Orchestration layer is totally unaware and beyond its programmatic scope to find new programmable resources; it operates only on programmable infrastructure resources allocated and exposed by the Slice Manager (SM) & Resource Orchestration Layer. Moreover, NOD is responsible to gather (from the end user) and handover to the SM the computational and network requirements and the network constraints in the form of Slice Intent. When the SM receives an application slice intent it replies with an assigned ID for the slice that it not yet ready to be used. The reason behind this provide-ID-before-it-is-ready is to handle non-blocking requests by the NOD and serve parallel requests. In the process of realization of a slice, the SM determines the network services required for that nApp graph and asks the Resource inventory for the corresponding resources. Upon a successful creation of the required network slice, the Slice Manager & Resource Orchestration collects all the information related with the allocated computational and network resources for both the application and network functions. Then the slice with the same ID that has been assigned is returned to the NOD (Slice Handler) in the form of a candidate materialized slice.

The Slice Handler receives the realized slice and checks the mapped Slice Intent to this ID. After the evaluation of the received slice, it forwards the deployment metadata (connection details, certificates) to the Deployment Manager and the nApp Deployment is about to begin.

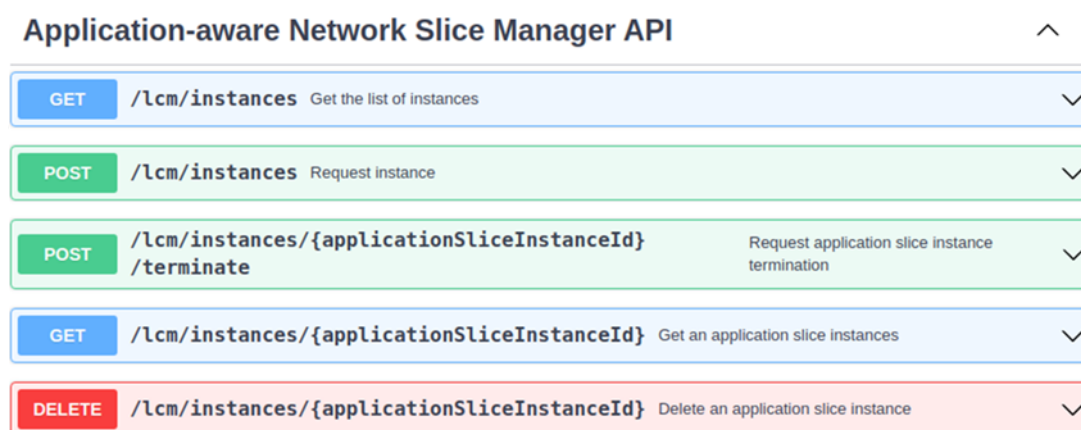
The process above describes the action on a successful realization and evaluation of a slice. In case of failure, there is only one action and that is to notify the end user.

7.2.2.2 API Specification

The current version of NOD-SM interface comprises an API supporting the following actions:

- 1) **Creation of a network slice based on the slice intent:** this request is released by the Slice Handler (NOD) to the SM, in order to start a process of slice creation and proceed with the deployment of the nApp graph when the network slice is available.
- 2) **Provision of a network slice instance:** this notification is sent by the SM to the Slice Handler (NOD) when the requested network slice is ready; then, the Deployment Manager (NOD) may initiate a deployment request.
- 3) **Deprovision of a network slice:** this request is released by the Slice Handler (NOD) to the SM in case that the created network slice is no longer required (e.g., no further nApp graph is going to be deployed over it); based on the request, the allocated resources are released, and the network slice instance is deactivated.

Figure 15 shows the **NOD-SM-IF** interface that has been formally defined using the OpenAPI specification.



Application-aware Network Slice Manager API		
GET	/lcm/instances	Get the list of instances
POST	/lcm/instances	Request instance
POST	/lcm/instances/{applicationSliceInstanceId}/terminate	Request application slice instance termination
GET	/lcm/instances/{applicationSliceInstanceId}	Get an application slice instances
DELETE	/lcm/instances/{applicationSliceInstanceId}	Delete an application slice instance

Figure 15: NOD-SM-IF OpenAPI specification

In 5G-IANA the programmability of this API will be extended with more actions and more attributes per action as stated in 3.2.1.4.

Table 12, NOD-SM-IF API

API	Method	Description
Resource: SM-IP>:<SM-PORT>/lcm/instances	POST	<p>Send a slice(*) intent to the SM with the specific requirements as parameters:</p> <ul style="list-style-type: none"> A. Computing Constraints: for expressing the computational requirements of the nApp graph B. Networking Constraints: for expressing the network high level requirements as type of slice (EMBB, URLLC, MIoT) and related values like latency for an URLLC or bandwidth for an EMBB type, isolation (boolean) C. Location constraints: for indication of a deployment over specific edge resources OBU/CORE. <p>(*) A sample of a slice intent can be found in Annex B</p>
Resource: <SM-IP>:<SM-PORT>/lcm/instances	GET	Returns all the active slices i.e. slices that have been allocated and assigned resources to be used
Resource: <SM-IP>:<SM-PORT>/lcm/instances/{ID}/terminate	POST	<ul style="list-style-type: none"> • Request to deprovision a requested by ID slice. This is a graceful termination request.
Resource: <SM-IP>:<SM-PORT>/lcm/instances/{ID}	DELETE	<ul style="list-style-type: none"> • Request to delete a slice instance by ID. This is a deletion request.

7.2.3. MANO Interface

MANO-IF exposes to the upper-level components (mainly the Slice Manager & Resource Orchestration layer and the nApp Orchestration and Development layer) an interface to interact with the distributed MANO Control plane. This integration can be explained through the following workflows:

- **E2E Resource Orchestration for Far Edge enabled nApps (RO) – MANO Integration:** The RO consumes the MANO-IF interface to allocate k8s quotas and namespaces in reaction once a slice intent request is received from the Slice Intent Handler (see Figure 16).

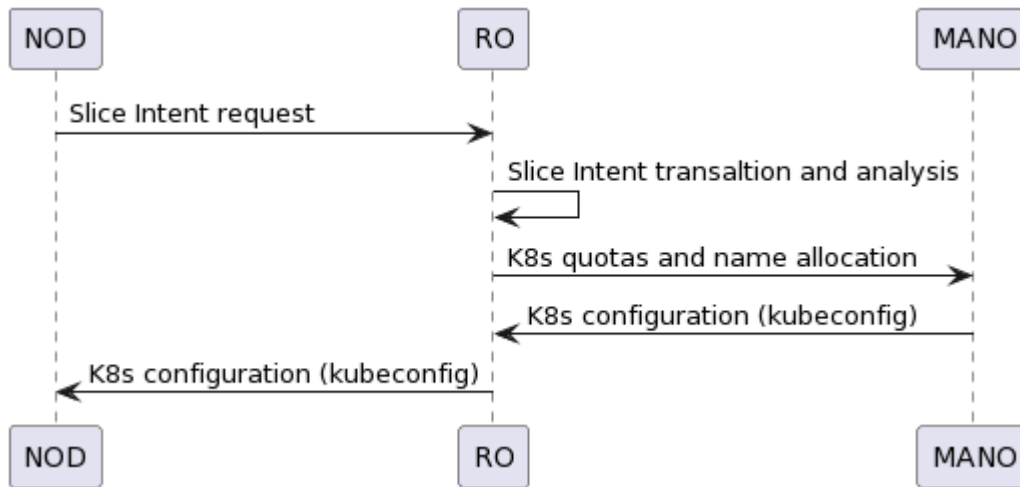


Figure 16: Resource Orchestration - MANO Interaction to allocate k8s quotas and namespaces

- OBU/EDGE/Cloud Continuum Resource Inventory (RI) – MANO Integration:** The RI interacts with the MANO layer (through the MANO-IF interface) to constantly have an updated snapshot of the availability of the Edge and Far Edge resources, leveraging on the subscribe/notify interface provided by k8s (see Figure 17).

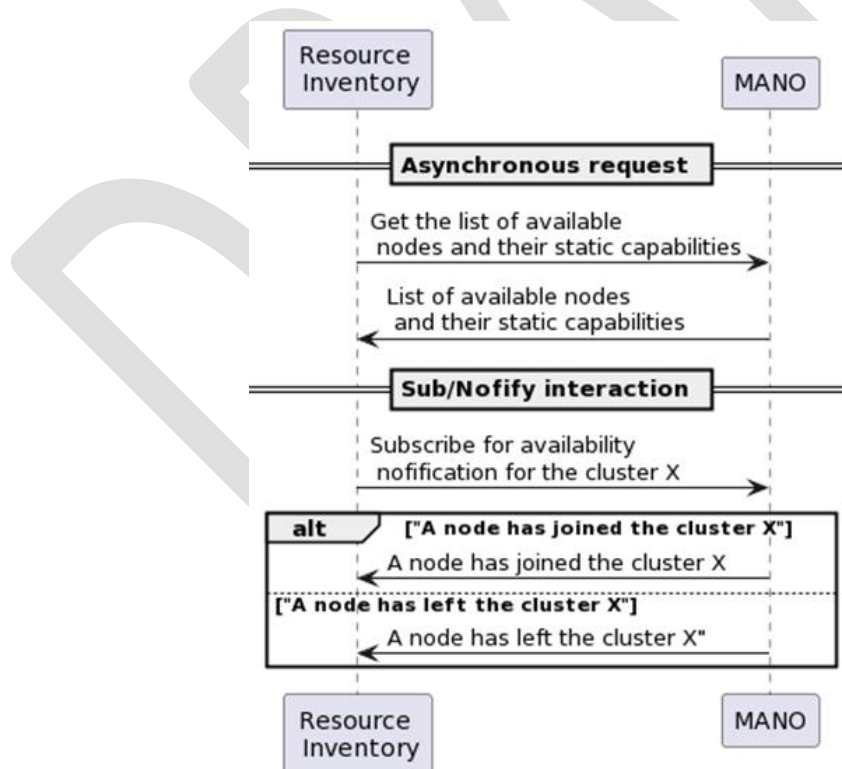


Figure 17: Resource Inventory - MANO Interaction to receive information about the availability of the Edge and Far Edge devices

Table 13, Table 14 shows the MANO-IF REST APIs used to implement the functions described in the integration workflows explained above.

- **Namespace API:** Used by the Resource Orchestration module to create/update/delete a K8s namespace and to allocate computing quotas for a service to be instantiated from the Vertical Application Orchestrator.

Table 13, Resource Inventory - Kubernetes REST Commands

API	Methods	Description
/api/v1/namespaces	GET, POST	Create and get information about the allocated namespaces
/api/v1/namespaces/{name}	GET, PUT, DELETE	Create, update, and get information about a namespace
/api/v1/namespaces/{namespace}/resourcequotas	GET, POST, DELETE	Create, delete, and get information about the quotas allocated for a namespace
/api/v1/namespaces/{namespace}/resourcequotas/{name}	GET, PUT, DELETE	Create, delete, and get information about a quota allocated for a namespace

- **Node API:** Used by the resource inventory to get information about the available Edge and Far Edge resource available on the system.

Table 14, Resource Inventory - Kubernetes REST Commands

API	Methods	Description
/api/v1/nodes	GET, POST, DELETE	Create delete, and get information about the available nodes for a given k8s cluster

/api/v1/nodes/{name}	GET, PUT, DELETE	Create delete, and get information about an available node for a given k8s cluster
/api/v1/watch/events	GET	Subscribe to the events for getting notifications once a node joins/leaves a k8s cluster

7.2.4. OVMANO Interface

The OVMANO-IF describes the interactions of on-vehicle and road side unit with the following components:

- the NOD: the interaction is in between the Deployment and the Lifecycle Manager for a) the first deployment of the service in the cluster running on-top of the OBU/RSU and b) for the lifecycle operations. Both (a) and (b) are realized through the programmatic interface of MicroK8s that is responsible.
- the Monitoring & Analytics (Cross layer) and with the Quota Manager of the Slice Manager and Multi-segment Orchestrator using the MicroK8s available interfaces for the needed OVM operations.

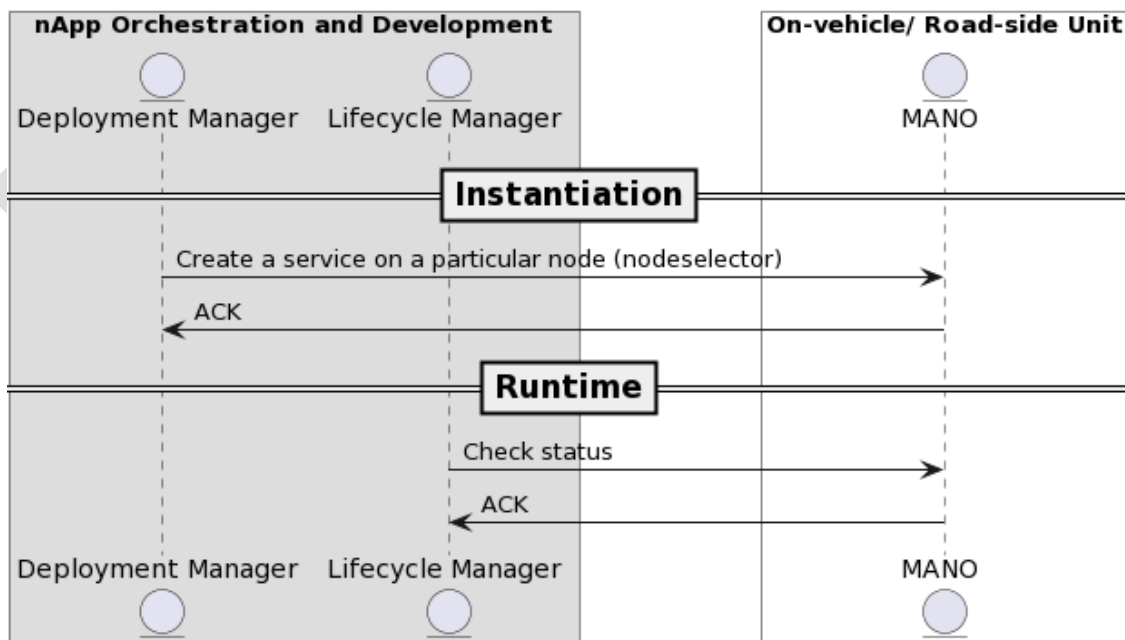


Figure 18: NOD - OBU/RSU MANO Interaction to deploy and to verify the status of a deployed service

- the NOD and the RO for the Information and Localization service; the interaction with the information service is based on the APIs made available from MicroK8s that provide information about the static hardware characteristics of the far-edge nodes; the localization service interacts with a Prometheus instance on the edge server as described in Section 5.2; the Prometheus instance scrapes the localization service for position information that is then provided to it.
- the Distributed Data collection for collecting data and metrics; a module named Data and Metrics Exporter is in charge of collecting data and metrics on the OBU; this module interacts with a Prometheus instance on the edge server as described in Section 5.3; the Prometheus instance scrapes this module for retrieving the available information.

DRAFT

8. CONCLUSION

This deliverable summarizes the developments of the 5G-IANA AOEP that have been carried out from M12 of the project until M21 which marks the completion of the developments for the first development cycle. The developments are being reported in a layered approach and specifically each section presents the developments and the extensions for a particular layer of the 5G-IANA AOEP. In addition, every development or extension described is accompanied by the feature that is linked as well as the interconnection with the other entities of the Platform. Furthermore, for each functional block of each layer of the architecture, an overview of the baseline technical description and the implemented extensions and enhancements are provided excluding only the nApp Toolkit technical description which is provided in Deliverable 4.1.

The description of the layers of the 5G-IANA AOEP starts with the nApp Orchestration and Development mechanisms, which supports the key features for the registration, deployment and management of the lifecycle of a nApp. The description of the developments per functional block (excluding nApp Toolkit) inside nApp Orchestration and Development includes also a nApp descriptor (see Annex A) as it has been agreed. This nApp descriptor shows the metamodel for declaring adequate resources for a nApp deployment.

Similarly for the Slice Manager and Resource Orchestration layer, it describes the extensions in order to support new features like the registration of on-vehicle MANO on a resource inventory.

Following are the cross-layer functionalities:

- a. The Distributed AI/ML mechanism goes through its software architecture and the functionalities that it empowers, such as the support of a multi-criteria client selection operation and provides links to the other elements of the Platform like cross-layer functionalities (e.g. monitoring).
- b. Monitoring and Analytics includes also the Distributed Data Collections, provide insights about the whole monitoring process as well as the mechanics behind that and the integration points with the other layers.

Finally, particular attention is given to specify how the different layers interact to achieve a nApp Deployment as well as the integration and exposure of the different registered edges through the Resource Inventory (Section 7). However, it is important to underline that this deliverable provides the description for some of the interfaces and not all of them (as described in D2.1). Extended description for the other interfaces will be provided in D3.3.

ANNEX A

Below is a sample of a nApp Descriptor as has been defined and modelled by 5G-IANA.

```
{
  "name": "string",
  "description": "string",
  "version": "string",
  "publicApplication": "boolean",
  "hexID": "string",
  "organization": "string",
  "type": ["SERVICE", "COMPONENT"],
  "serviceCategory": ["HAZARD_NOTIFICATION", "VEHICLE_MOVEMENT",
"SMART_TRAFFIC_PLANNING", "INFOTAINMENT", ],
  "packageType": ["HELM"],
  "specLevel": ["VERTICAL_AGNOSTIC", "VERTICAL_SPECIFIC"],
  "accessLevel": ["PRIVATE", "RESTRICTED", "PUBLIC"],
  "useCase": "string",
  "testbed": ["NOKIA", "TS"],
  "softwareLicenses": [{
  }],

  "componentNode": [{
    "componentNodeID": "long",
    "hexID": "string",
    "name": "string",
    "component": [{
      "id": "long",
      "name": "string",
      "hexID": "string",
      "publicComponent": "boolean",
      "architecture": "string",
      "iconBase64": "string",
      "dockerImage": "string",
      "dockerRegistry": "string",
      "dockerCredentialUsing": "boolean",
      "dockerCustomRegistry": "boolean",
      "dockerUsername": "string",
      "dockerPassword": "string",
      "exposedInterfaces": [{
        "interfaceID": "long",
        "name": "string",
        "port": "string",
```

```

        "interfaceType": "string",
        "transmissionProtocol": "string"
    }],
    "requiredInterfaces": [{
        "graphLinkID": "long",
        "friendlyName": "string",
        "interfaceId": "long",
    }],
    "requirement": [{
        "requirementId": "long",
        "CPU": "Integer",
        "Ram": "Float",
    }],
    "healthCheck": [{
    }],
    }
}],
"LinkNodes": [{
    "linkNodeID": "long",
    "componentNodeFrom": "ComponentNode",
    "componentNodeTo": "ComponentNode",
    "graphLink": {
        "linkID": "long",
        "friendlyName": "string",
        "interfaceObj": {
            "interfaceId": "long",
            "name": "string",
            "port": "string",
            "interfaceType": "string",
            "transmissionProtocol": "string"
        }
    }
}],
"required5GCoreService": [{
    "fiveGServiceSpecid": "string",
    "version": "string",
    "function": ["NWDAF", "LCS"],
    "mandatory": "boolean",
    "name": "string"
}]
}

```

ANNEX B

Slice Intent Descriptor with comments.

```
{
  "callbackUrl": "URL-where-the-slice-will-be-sent",
  "locationConstraints": [
    {
      "applicationComponentId": "15",
      "geographicalAreaId": "athens-dc1" //Placement Constraint for a specific
component
    },
    {
      "applicationComponentId": "16",
      "geographicalAreaId": "athens-dc1" //Placement Constraint for a specific
component
    }
  ],
  "computingConstraints": [
    {
      "applicationComponentId": "15",
      "ram": "2Gi",
      "cpu": "1",
      "storage": "20Gi"
    },
    {
      "applicationComponentId": "16",
      "ram": "2Gi",
      "cpu": "1",
      "storage": "20Gi"
    }
  ],
  "networkingConstraints": [
    {
      "applicationComponentId": "15",
      "applicationComponentEndpointId": "15",
      "sliceProfiles": [
        {
          "sliceType": "EMBB",
          "profileParams": {
            "isolationLevel": "NO_ISOLATION",
            "dlThroughput": 150000.0, // Required Bandwidth - downlink
            "ulThroughput": 1000000.0 // Required Bandwidth - uplink
          }
        }
      ]
    }
  ]
}
```

```
}  
}  
]  
}  
]  
}
```

DRAFT

ANNEX C

In this section we refer to Table 13: “Reused Software and Extensions” of Deliverable 2.1 (Specifications of the 5G-IANA architecture, revised version) and we present the specific extensions that are already implemented with respect to the various modules at different layers. Moreover, for each layer, the main developments/extensions are indicated with the dashed boxes.

nApp Orchestration & Development layer developments

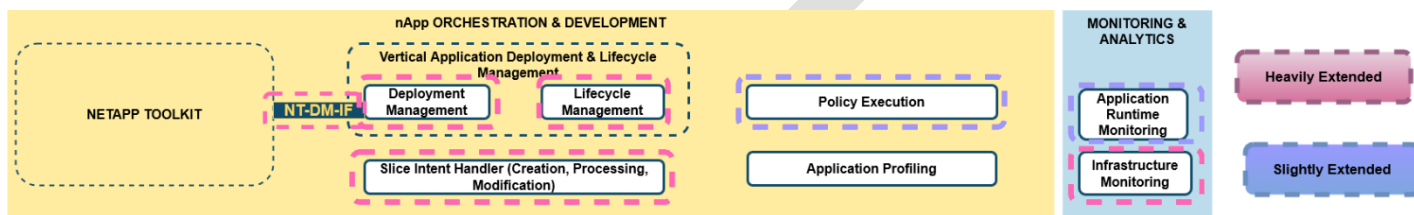


Figure 19: Visualization of the work done in the NOD layer

Figure 19 depicts and Table 15 summarizes the extensions for the nApp Orchestration and Development layer and the work that is planned for Phase B of the project.

Table 15, Description of extensions per module of NOD

nApp Orchestration and Development Layer				
Functionalities	Origin	Implemented Extensions for the phase A	Planned Extensions for the phase B	Responsible Partner
Deployment and Lifecycle Management	H2020 5G-PPP phase II MATILDA	Completed transformed for Openstack based to be fully compatible with the Kubernetes environment and MicroK8s. Both Deployment Manager Microservice and the Lifecycle Manager microservice have been implemented as a Kubernetes controller responsible for service graphs deployments and lifecycle and status operations. Also, the Deployment Manager is integrated with the nApp Toolkit through the NT-DM-IF. The extract of this integration is the nApp Descriptor (see Annex A).	Extensions to support more Kubernetes resources. Kubernetes resources are extensions of the Kubernetes API. Specifically, there will be extensions to support for Kubernetes Configmaps	UBI

Slice Intent Handler	H2020 5G-PPP phase III Int5Gent project	<p>Heavily extended so to support OBU/RSU deployments. Specifically, the Slice Intent metamodel has been extended to support edge deployment selection. This functionality for retrieving the edges and the deployment locations has been implemented as a new microservice.</p>	<p>New API function will be implemented. Specifically, “sliceUpdate” function will facilitate adaptive slices (i.e., with inflated/deflated cloud). The sliceUpdate will take place after the initial deployment of the nApp and will cover changes on the already allocated and provisioned slice (see section 3.3.3)</p>	UBI
Application Profiling	Maestro® (Commercial Product by UBI)	No	The developments concern the integration with the overall platform	UBI
Policy Execution	H2020 5G-PPP phase II MATILDA	Slightly extended. The extensions concern the optimizations to deal with the scalability and the performance bottleneck of Drools.	Policies execution will integrate with the 5G-IANA AOEP and further extensions with regards to the integration with the DMLO are examined.	UBI

Slice Management and Resource Orchestration Layer

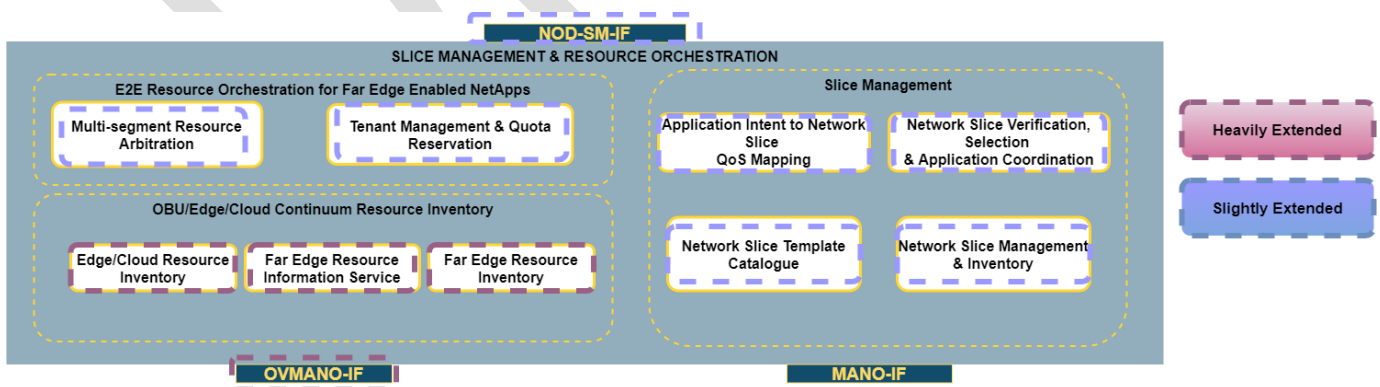


Figure 20: Visualization of the work done in the NOD layer

Figure 20 depicts and Table 16 summarizes the extensions for the Slice Management & Resource Orchestration layer and the work that is planned for Phase B of the project.

Table 16, Description of extensions per module of SM& RO layer

Slice Management and Resource Orchestration				
Functionalities	Origin	Implemented Extensions for the phase A	Planned Extensions for the phase B	Responsible Partner
SM – Application Intent to Slice QoS Mapping	H2020 5G-PPP phase III Int5Gent project	This component maps the intent QoS parameters into a suitable NEST. Planned extensions relate to the enhancement of the mapping mechanism to support additional QoS parameters	Phase B will deliver all the functionalities presented in section 4.1.3	NXW
SM – Network Slice Verification, Selection & Application Coordination	H2020 5G-PPP phase III Int5Gent project	This component coordinates the selection of the NSI and the provisioning of the compute quotas. Planned extensions relate to the implementation of a verification mechanism to assess the presence of a suitable running NSI that matches the selected NEST	Phase B will involve the development of new mechanisms to properly assign a suitable NSI that matches with the selected NEST.	NXW
SM – Network Slice Management & Inventory	H2020 5G-PPP phase III Int5Gent project	This component is an inventory of available NSIs. Planned extensions relate to the implementation of a RESTful interface to enable the management of NSIs by the Network Operator.	Phase B will deliver the all the functionalities described in section 4.1.3, including mechanisms to select the proper Far Edge device based on some dynamic filtering information (location, resources available etc.)	NXW
SM – Network Slice Template Catalogue	H2020 5G-PPP phase III Int5Gent project	This component catalogues the available Network Slice Templates (i.e., NEST and 3GPP-based ones). The component can be eventually extended to update the implemented data models	Phase B will involve a continuous update of the NEST data model according to the evolution of the 3GPP standards	NXW

		according to the evolution of the related standard specification		
--	--	--	--	--

Cross Layer Functionalities

Table 17 summarizes the extensions for the cross-layer functionalities and the work that is planned for Phase B of the project.

Table 17: Description of extensions per module of the cross-layer functionalities

Cross Layer Functionalities				
Functionalities	Origin	Implemented Extensions for the phase A	Planned Extensions for the phase B	Responsible Partner
DMLO	None	DMLO planned for Release B. Developments ongoing according to detailed specifications presented in Section 6. Completed AF and FLOWER integration (Section 6.1.1) and ongoing developments regarding the RI and retrieval of monitoring aspects.	Phase B will deliver the entire functionality described in Section 6.1.	ICCS, UULM
Monitoring & Analytics	Maestro® (Commercial Product by UBI)	The reported developments concern, the different monitoring instances (cloud, edge) for collecting both on-vehicle MANO metrics as well as application metrics.	Extensions concern the addition of use case application specific metrics. Also the infrastructure - based metrics will be enhanced (e.g. actual QoS measures on networking services, mobility)	UBI/NXW/LINKS

REFERENCES

- ¹ Helm Package Management System for using resources in Kubernetes : <https://helm.sh/docs/topics/charts/>
- ² Openstack Infrastructure as A service : <https://www.openstack.org/>
- ³ Kubernetes Container Orchestrator : <https://kubernetes.io/>
- ⁴ Openshift Container Orchestrator: <https://www.redhat.com/en/technologies/cloud-computing/openshift>
- ⁵ Atlassian, "SLA vs. SLO vs. SLI - differences," *Atlassian*. [Online]. Available: <https://www.atlassian.com/incident-management/kpis/sla-vs-slo-vs-sli>.
- ⁶ "Microservices Pattern: Microservice architecture pattern," *microservices.io*. [Online]. Available: <https://microservices.io/patterns/microservices.html>.
- ⁷ Spring Boot Framework: Used for building web applications - <https://spring.io/>
- ⁸ Quarkus Framework: Used for building microservices - <https://quarkus.io/>
- ⁹ Java Programming language: <https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html>
- ¹⁰ Representation State Transfer -API: <https://restfulapi.net/>
- ¹¹ Kafka event streaming technology: <https://kafka.apache.org/>
- ¹² Drools Rule Composition and Management System : <https://www.drools.org/>
- ¹³ T. Kobayashi, "How to find a bottle-neck in your rule for performance analysis," KIE Community, 27-Jul-2021. [Online]. Available: <https://blog.kie.org/2021/07/how-to-find-a-bottle-neck-in-your-rules-for-performance-analysis.html>.
- ¹⁴ <https://blog.kie.org/>
- ¹⁵ Framework for Scientific Computations: <https://www.opencpu.org/>
- ¹⁶ Framework for creating ML models : <https://www.tensorflow.org/>
- ¹⁷ Kubernetes ConfigMap data structure : <https://kubernetes.io/docs/concepts/configuration/configmap/>
- ¹⁸ "Generic Network Slice Template 25 November 2021 - GSMA." [Online]. Available: <https://www.gsma.com/newsroom/wp-content/uploads/NG.116-v6.0.pdf>.
- ¹⁹ Minimized flavor of Kubernetes : <https://microk8s.io/>
- ²⁰ <https://minikube.sigs.k8s.io/docs/start/>
- ²¹ <https://docs.k3s.io/>
- ²² <https://gpsd.gitlab.io/gpsd/>
- ²³ Drainakis, G., Pantazopoulos, P., Katsaros, K. V., Surlas, V., & Amditis, A. (2021, July) On the Resource Consumption of Distributed ML. In The IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN 2021). IEEE
- ²⁴ Georgios Drainakis, Panagiotis Pantazopoulos, Konstantinos V. Katsaros, Vasilis Surlas and Angelos Amditis, " On the distribution of ML workloads to the network edge and beyond," in Proc. of the First International INFOCOM Workshop on Distributed Machine Learning and Fog Networks (INFOCOM WKSHP FOGML) 2021

²⁵ Georgios Drainakis, Konstantinos V. Katsaros, Panagiotis Pantazopoulos, Vasilis Sourlas and Angelos Amditis, "Federated vs. Centralized Machine Learning under Privacy-elastic Users: A Comparative Analysis," in Proc. of 19th IEEE International Symposium on Network Computing and Applications, 2020 (IEEE NCA 2020)

²⁶ <https://www.3gpp.org/news-events/3gpp-news/sa6-verticals>

²⁷ Ippokratis Sartzetakis, Polyzois Soumplis, Panagiotis Pantazopoulos and, Konstantinos V. Katsaros, Vasilis Sourlas and Emmanouel Varvarigos, "Resource Allocation for Distributed Machine Learning at the (Edge-Cloud) Continuum", IEEE International Conference on Communications (ICC): Communication, QoS, Reliability and Modeling Symposium (IEEE ICC'22 - CQR Symposium)", Seoul, Korea (South), May, 2022

²⁸ Prometheus Communication API : <https://prometheus.io/docs/prometheus/latest/querying/api/>

²⁹ Netdata- Infrastructure Monitoring Metric exporter : <https://www.netdata.cloud/>

DRAFT