A Holistic, Innovative Framework for the Design, Development and Orchestration of 5G-ready Applications and Network Services over Sliced Programmable Infrastructure

Co-funded by the Horizon 2020 Framework Programme of the European Union

DELIVERABLE D4.2

NETWORK AND COMPUTING SLICE

Due Date of Delivery:	M24 *Mx* (31/05/2019 *dd/mm/yyyy*)
Actual Date of Delivery:	04/09/2019 *dd/mm/yyyy*
Workpackage:	**WP4 – Network and Computing Slice Deployment Platform**
Type of the Deliverable:	OTHER
Dissemination level:	PU
Editors:	CNIT
Version:	1.0

Call:
H2020-ICT-2016-2

Type of Action:
IA

Project Acronym:
MATILDA

Project ID:
761898

Duration:
35 months

Start Date:
01/06/2017 *dd/mm/yyyy*

Project Coordinator:
Name: Franco Davoli
Phone: +39 010 353 2732
Fax: +39 010 353 2154
e-mail: franco.davoli@cnit.it

Technical Coordinator:
Name: Panagiotis Gouvas
Phone: +30 216 5000 503
Fax: +30 216 5000 599
e-mail: pgouvas@ubitech.eu

List of Authors	
CNIT	CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI
Roberto Bruschi, Franco Davoli, Chiara Lombardo, Jane Frances Pajo	
UBITECH	GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS
Anastasios Zafeiropoulos	
ERICSSON	ERICSSON TELECOMUNICAZIONI
Orazio Toscano	
INC	INCELLIGENT IDIOTIKI KEFALAIOUCHIKI ETAIREIA
Panagiotis Demestichas, Kostas Tsagkaris, Athina Ropodi, Nikos Stasinopoulos.	
COSM	COSMOTE KINITES TILEPIKOINONIES AE
Ioanna Mesogiti	
NCSRD	NATIONAL CENTER FOR SCIENTIFIC RESEARCH “DEMOKRITOS”
Kourtis Michail-Alexandros, Themis Anagnostopoulos	
ATOS	ATOS SPAIN SA
Fernando Díaz, Javier Melian	
AALTO	AALTO-KORKEAKOULUSAATIO
Ibrahim Afolabi, Miloud Bagaa, Tarik Taleb	
ININ	INTERNET INSTITUTE Ltd.
Luka Koršič, Dušan Mulac, Janez Sterle	
ORO	ORANGE ROMANIA SA
Catalin Brezeanu, Jean Ghenta, Marius Iordache, Cristian Patachia, Bogdan Rusti, Horia Stefanescu	

Disclaimer

The information, documentation and figures available in this deliverable are written by the MATILDA Consortium partners under EC co-financing (project H2020-ICT-761898) and do not necessarily reflect the view of the European Commission.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.

Copyright

The MATILDA Consortium consists of:

CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI (CNIT)

ATOS SPAIN SA (ATOS)

ERICSSON TELECOMUNICAZIONI (ERICSSON)

INTRASOFT INTERNATIONAL SA (INTRA)

COSMOTE KINITES TILEPIKOINONIES AE (COSM)

ORANGE ROMANIA SA (ORO)

EXXPERTSYSTEMS GMBH (EXXPERT)

GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)

INTERNET INSTITUTE, COMMUNICATIONS SOLUTIONS AND CONSULTING LTD (ININ)

INCELLIGENT IDIOTIKI KEFALAIOUCHIKI ETAIREIA (INC)

NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS" (NCSRD)

UNIVERSITY OF BRISTOL (UNIVBRIS)

AALTO-KORKEAKOULUSAATIO (AALTO)

UNIVERSITY OF PIRAEUS RESEARCH CENTER (UPRC)

ITALTEL SPA (ITL)

BIBA - BREMER INSTITUT FUER PRODUKTION UND LOGISTIK GMBH (BIBA)

SUITE5 DATA INTELLIGENCE SOLUTIONS LIMITED (S5).

Table of Contents

Table of Acronyms

Acronym	Definition
API	Application Programming Interface
APN	Access Point Name
CIoT	Cellular IoT
eNB	E-UTRAN Node B
ENM	Ericsson Network Manager
EPC	Evolved Packet Core
EPS	Evolved Packet System
e-RAB	EPS Radio Access Bearer
E-UTRAN	Evolved – Universal Terrestrial Radio Access Network
GRE	Generic Routing Encapsulation
GTP	General Packet Radio Service Tunnelling Protocol
HSS	Home Subscriber Server
IaaS	Infrastructure as-a-Service
IoT	Internet of Things
LAN	Local Area Network
MANO	Management and Orchestration
MCO	Multi-Cluster Overlay
MME	Mobility Management Entity
NAT	Network Address Translation
NFV	Network Functions Virtualization
NFVCL	Network Functions Virtualization Convergence Layer
NFVO	NFV Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSI	Network Slice Instance
ODL	OpenDayLight
OS	OpenStack
OSM	Open Source MANO
OSS	Operations Support System
OTT	Over-the-Top
PCE	Path Computation Engine
PCRF	Policy and Charging Rules Function
PDN	Public Data Network
P-GW	PDN Gateway
PLMN	Public Land Mobile Network
PNF	Physical Network Function
QCI	QoS Class Identifier
QoS	Quality of Service
REST	Representational State Transfer
RSO	Resource Selector Optimizer

Acronym	Definition
SBA	Service Based Architecture
SLA	Service Layer Agreement
SDN	Software Defined Networking
SIM	Subscriber Identity Module
SLA	Service-Layer Agreement
SLM	Slicing Lifecycle Manager
S-GW	Serving Gateway
TAC	Tracking Area Code
TLP	Telecom Layer Platform
UE	User Equipment
vApp	Vertical Application
VAO	Vertical Application Orchestrator
VIM	Virtual Infrastructure Manager
VIMCL	VIM Convergence Layer
VLD	Virtual Link Descriptor
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFM	VNF Manager
VNFI	VNF Instance
VNFM	VNF Manager
VXLAN	Virtual Extensible Local Area Network
WAN	Wide-Area Network
WIM	Wide-area Infrastructure Manager
WIMCL	WIM Convergence Layer
WSC	Wide-Area SDN Controller
YAML	Yet Another Markup Language

1 Executive Summary

The scope of MATILDA is to deliver a holistic and innovative 5G framework to undertake the lifecycle of design, development and orchestration of 5G-ready applications and 5G network services over programmable infrastructure. To achieve this goal, a radically new Telecom Layer Platform is required, able to support functional and performance requirements, as well as the effective lifecycle management of 5G-ready Vertical Applications.

Activities undertaken within WP4 have dealt with the network layer of the MATILDA framework, which also includes an applicative and an infrastructure layer. In particular, the previous deliverable report has paved the way for the architectural design of the network platform by outlining the main design principles driving the development, such as the usage of microservice meshes to improve the flexibility of the framework and speed up the provisioning and deployment processes.

The final outcome of WP4 activities presented in this deliverable is the completed development of the Telecom Layer Platform, which supports the whole lifecycle management of 5G-ready Vertical Applications while fulfilling their functional and performance requirements. This documents presents details on the final release, including design choices, algorithms and mechanisms for all the platform building blocks, as well as highlighting progress beyond the first release.

2 Introduction

The goal of WP4 activities has been to build and operate slices of network and computing resources for providing network services to the vertical applications to be managed through their Vertical Application Orchestrator (VAO). In this respect, a radically new Telecom Layer Platform prototype has been implemented to manage the entire lifecycle of 5G-ready Vertical Applications (vApps), by leveraging on pools of network services, spanning from the 4/5G radio all the way to the wide-area network, and of computing resources directly accessible by the VAO.

The main building blocks composing the platform, the stakeholders and the related domains have been defined at the very beginning of the project: the overall MATILDA framework shown in Figure 1 was initially presented in the D1.1 [1]; the Telecom Layer Platform is located in the middle and exposes northbound interfaces towards multiple diverse vertical applications, and southbound interfaces towards multiple network and computing infrastructures.

The finalized MATILDA 5G Telecom and Infrastructure Platform is fully compliant with this vision as well as with the first release of the framework presented in the D4.1 report [2]. The role of each building block and their deployment are outlined in Section 3 and exhaustively described in the remaining of the document, with details on the progress beyond the first release, and the additional solutions that have been introduced for supporting functionalities, mainly related to 5G-ready applications as well as monitoring. In order to further map WP4 outcomes against the objectives in the Grant Agreement, the following sub-section provides a brief summary of the activities performed for each task.

2.1 Work Performed per Task

The aim of **Task T4.1** has been to extend the functionalities of a traditional OSS in order to support the MATILDA architecture at the Telecom administrative layer. The nine main components initially defined in the D4.1 report have been developed as a suite of five cloud-native, parallelizable software services compatible with the service-mesh paradigm.

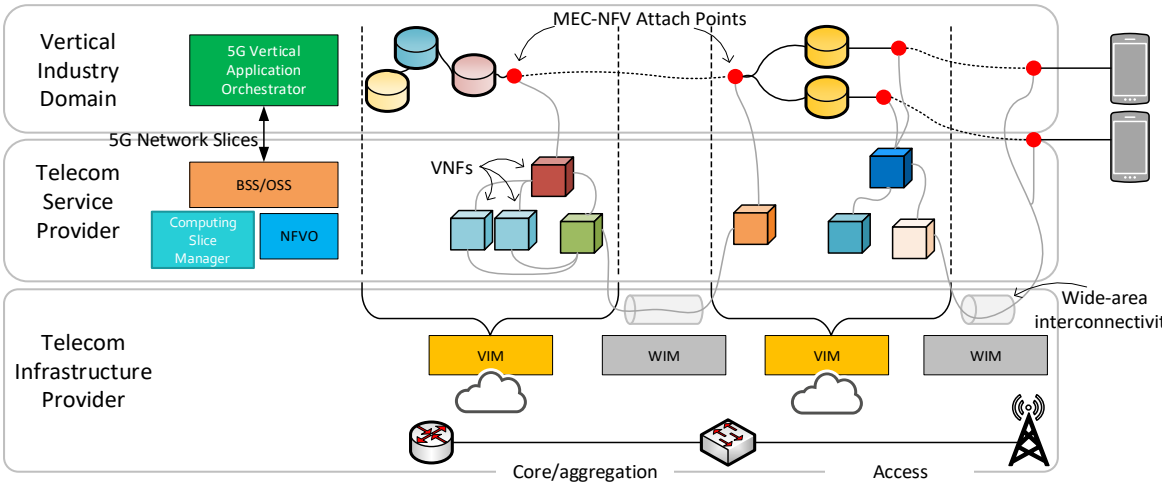


Figure 1: Key architectural building blocks and main stakeholders involved in the deployment of a vertical application onto a 5G infrastructure.

Table 1 provides a mapping between the initial functionality and the final software, along with introducing the newly added functionalities. The table can be seen as an update of Table 4 from the D4.1, which listed the OSS functions and implementation status at the time of delivery (M18).

Table 1: Mapping between OSS functionalities and corresponding software implementation in the final release.

Functionality	Corresponding software module	Details (Section)
Slicing Lifecycle Manager	OSS core module	- Finalization of the APIs for interaction with the convergence layers (Sect. 5) - Finalized integration of the VIM (Sect. 5) and WIM (Sect. 8) convergence layers - Realization of the attach points between OS instances (Sect. 5) - Addition of the monitoring framework (Sect. 10)
VIM Convergence Layer		
WIM Convergence Layer		
NFV Convergence Layer	Network Service Manager and NFV Convergence Layer	- Finalized integration of the convergence layer with OSM (Sect. 7) - Completed implementation of network elements post-boot configuration, monitoring, and dynamic management. (Sect. 7)
Network Service Manager		
Slicing Northbound Module	Slicing Northbound Module	- Finalization of the APIs for exchange of materialized slice messages. (Sect. 4)
Resource Selection Optimizer	Resource Selection Optimizer	- Finalization of the algorithm for internal graph representation model (Sect. 6) - Added feature allowing to change algorithms and policies for resource optimization and slice deployment in a plug-and-play fashion (Sect. 6).
Persistency Layer	Persistency Layer	- Separation of the layer into two different databases (MongoDB and Prometheus) for configuration and monitoring, respectively (Sect. 5, Sect. 6)

Task T4.2 deals with the design and implementation of the multi-site NFV Orchestrator, which is in charge of the lifecycle management of both 3GPP and non-3GPP network services. Since the latest version of Open-Source MANO has been chosen as NFVO, most of the activities of this task have regarded the development of the NFV Convergence Layer (NFVCL).

In the final release of the MATILDA 5G Telecom and Infrastructure Platform, the NFVCL has maintained the previously defined structure, composed of the NFV Service Mapper (NFVSM), which interacts with the RSO to select the most appropriate blueprint and determine the amount of resources to be used in each VIM, and the NFV Service Setup (NFVSS), which performs the lifecycle operations (instantiation, operation and deprovisioning) on the network services.

Activities undertaken after the first release have focused on the finalization of the interfaces and APIs between the NFVCL and OSM and between the NFVCL and the RSO. Development details can be found in Section 7. Another relevant outcome of this task is represented by the blueprints, which were defined in the D4.1 as data structures allowing the fields to be presented as wildcards used for composing the Network Service Descriptor (NSD) and has been completely developed for the final release. The blueprint metamodel allows selecting the most suitable network services according to the desired performance and operational requirements, as well as including the information needed to create the configuration file for proper initialization of the Virtual Machine running a VNF. Further details on the blueprint selection, along with how its information brings to the creation of the NSD, are reported in Section 7 and Annex 1.

Activities of the **Task T4.3** provide the support of the Edge Computing functionalities, that is exposing the resources available in the wide-area to the VAOs and providing the attach points between the NFVO and Vertical Applications domains at the edge VIMs. The main contributions provided for the final release of the platform have exploited the outcomes of T4.2 and T4.4 to develop the required, ad-hoc solutions for handling specific aspects of OS and OSM and guarantee their cooperation.

To provide the wireless connectivity, a Public Land Mobile Network (PLMN) has been designed to be instantiated by the NFVO at bootstrap. Throughout the activities of WP4, the selection of the individual services has been made by leveraging on the levels of flexibility and the offered features. In the final release, the PLMN is composed of *i*) a single, monolithic VNF for the EPC, *ii*) an eNode PNF for each Tracking Area Code (TAC), with *iii*) an associated bypass VNF (see D4.1) deployed in each edge VIM connected to an eNodeB.

The attach point interconnecting the vApp and NFVO domains in OS, as described in the D4.1, is realized by means of Rule-Base Access Control (RBAC); moreover, an OS Neutron router has been designed for the final release to handle the IP addressing of the VNFs composing the PLMN.

Regarding the wide-area connectivity, it was decided to rely on VXLAN/GRE tunnels for isolating the data shared between couples of eNodeB PNF and bypass VNF, and also for the interconnection of components of a same vApp deployed across multiple VIMs.

Finally, the **Task T4.4** realizes the network services required for handling wide-area interconnectivity and for supporting the use case applications. Table 2 in Section 9 lists all the 3GPP and non-3GPP P/VNFs that have been implemented in MATILDA. Such implementation, for the accomplishment of the MATILDA Telecom Layer Platform as well as the support of the demonstrators, has entailed, for each network service, the realization of the NS package by appropriately filling the blueprint and adding the Juju charm, which was designed ad-hoc on a per-service basis as well. The packages have then been onboarded in the OSM catalogue to be available for instantiation, upon which the charm allows managing Day-2 configurations.

While the first release of the Telecom platform only included a preliminary implementation of the 4G network services, the final prototype has been enhanced by adding an alternative 4G service, active network monitoring, along with non-3GPP services such as firewall, overlays, routers etc. that can be provided to vApps on request. The complete description of the available services can be found in Section 9.

2.2 MATILDA End-to-End Story

MATILDA offers a framework that allows software developers to create applications following a simple and conventional microservices-based approach where each component can be independently orchestratable. Based on the conceptualization of metamodels (application component and graph metamodels), they can formally declare information and requirements - in the form of descriptor- that can be exploited during the deployment and operation over programmable infrastructure. Such information and requirements may regard capabilities, envisaged functionalities and soft or hard constraints that have to be fulfilled and may be associated with an application component or virtual link interconnecting two components within an application graph. The produced application is considered as 5G-ready application.

Service Providers are able to adopt the developed 5G-ready applications (published to the Marketplace or created internally) and specify policies and configuration options for their optimal deployment and operation over programmable infrastructure. Based on the provided application descriptor, service providers are able to design operational policies and formulate a **slice intent**. These operational policies describe how the application components should adapt their execution mode in runtime. On the other hand, the slice intent includes a set of constraints that have to be fulfilled during the placement of the application and a set of envisaged network functionalities that have to be provided. This information is used by the Vertical Application Orchestrator to request from the Telecommunication Infrastructure Provider the creation of an appropriate **application-aware network slice**.

While the instantiation and management of the application-aware network slice (including the set of network functions) is realised by the Network and Computing Slice Deployment mechanisms (managed by the telecommunications infrastructure provider), the deployment and runtime management of an application is realised by the vertical application orchestrator (managed by the service provider), following a service-mesh-oriented approach. In order to instantiate and manage the application-aware network slice during the overall lifecycle of the 5G-ready application, Telecommunication Infrastructure Providers rely on the concept of network slice to fulfil the vertical application needs. A **network slice is a logical infrastructure partitioning allocated resources and optimized topology** with appropriate isolation, to serve a particular purpose of an application graph.

The Network and Computing Slice Deployment mechanisms includes an OSS/BSS system, a NFVO and a resources manager for managing the set of deployed WIMs and VIMs. Based on the interpretation of the provided slice intent, the required network management mechanisms are activated and dynamically managed. The Telecommunication Infrastructure Provider is responsible to realise the instantiation of the slice over the programmable infrastructure. The reserved resources for this slice combine both network and compute resources. A Telecommunication Infrastructure Provider may deliver all these resources based on his own infrastructure or come into an agreement with a Cloud Infrastructure Provider and acquire access to additional compute resources (e.g. in the edge of the network).

These actions are realized in an agnostic way to application service providers. However, through a set of open APIs, requests for adaptation of the slice configuration may be provided by the Vertical Applications Orchestrator to the Network and Computing Slice Deployment Platform.

The materialization of the network slice requires the instantiation of network services (NSs) that are composed of virtual network functions (VNFs) chains. These NSs and VNFs can

imported into the telecommunications infrastructure provider's catalogue from MATILDA marketplace. A summary of the described overall lifecycle of an application created with the MATILDA framework is represented in Figure 2 below, highlighting the interaction among the different stakeholders and the usage of the metamodels.

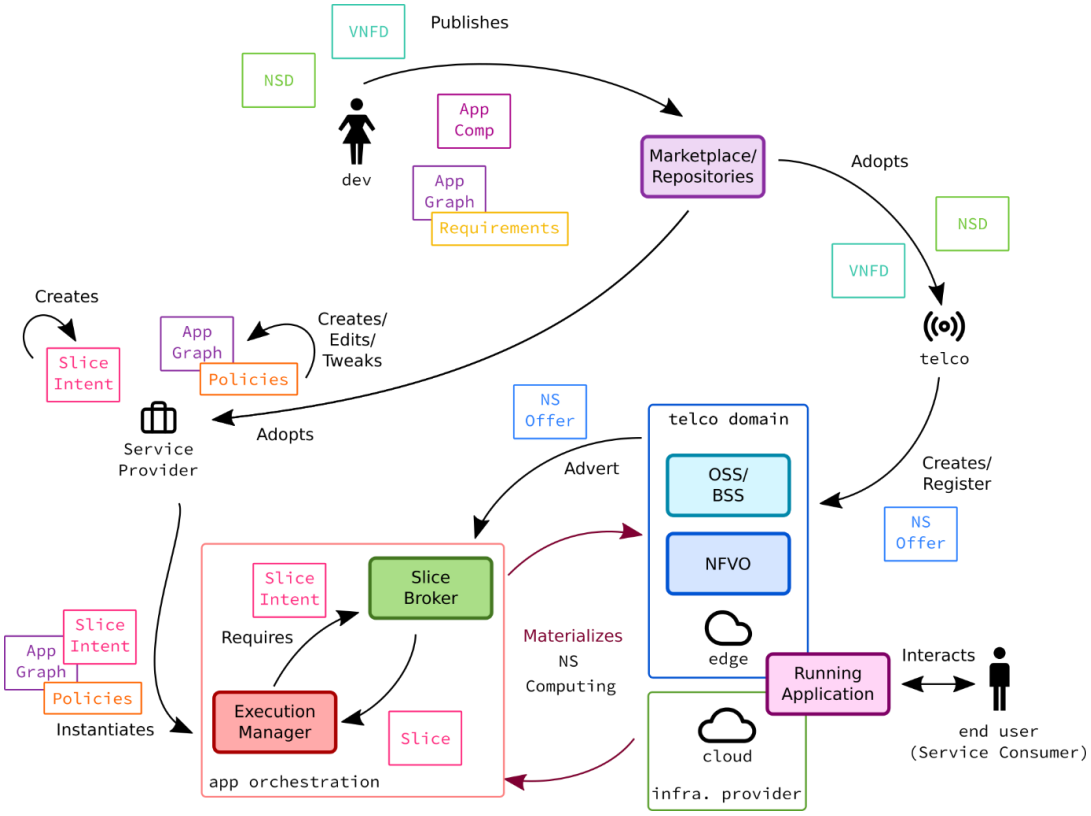


Figure 2: MATILDA workflow highlighting the different stakeholders and metamodels.

2.3 Deliverable Structure

After the Executive Summary in **Section 1** and the Introduction in **Section 2**, this deliverable is organized as follows.

Section 3 presents a high-level view of the Telecom Layer Platform, including its building blocks and the resources they deal with. Moreover, this section provides an example of platform deployment to better understand the role of each module within the architecture.

Section 4 focuses on the Slicing Northbound module, which is in charge of the deployment and lifecycle management of the application graph components over the created network slice. Attention is put on the interfaces supporting the interaction between the VAO and the OSS for requesting the creation of a new network slice, and provisioning/deprovisioning of an existing slice instance.

Section 5 provides some development details on the design of the OSS Core module, which represents a mapping of the OSS functionality outlined in the D4.1 with the actual software code realized for the final release of the platform. This section also deals with the retrieval and management of the information exchanged among the other building blocks and their temporary storage in the persistency layer.

The Resource Selector Optimizer, which is responsible for the optimization and reinforcement policies for selecting the most appropriate blueprint and determine the amount of resources to be used in each VIM, is described in **Section 6.** The design choices that have driven the finalization of this module in its current structure are discussed as well.

Section 7 covers the module that has incorporated the NFV Convergence Layer and the Network Service Manager features defined in the D4.1. This section also introduces the definition of the Network Service Blueprints, as well as presenting the complete set of operations driving the setup of a network service. The complete reference of the APIs used by the NFVSM and NFVSS components is reported in **Annex 1**.

Section 8 is about the Wide-Area Infrastructure Manager, which is based on the Ericsson Network Manager and allows for the management and monitoring of the resources in the wide-area network that are available for the creation of network slices according to vertical application requirements. Examples on the request and response APIs designed for the communication between the WIM and the OSS, and for the actual routing setup, are provided in **Annex 2**.

Section 9 draws to a close the description of the lifecycle orchestration of network services by reporting the description of the main network services supported by the NFVCL+NSM module. Such services are classified as "base 4G/5G network services", which are the ones providing base 3GPP 4G/5G network connectivity to User Equipment (UE), and "vApp/slice-specific network services", which provide additional functionalities dedicated to a specific vApp.

Section 10 introduces the OSS Monitoring Framework, which was the final addition to the platform and allows monitoring the performance metrics related to the VIMs (such as vCPU, RAM and disk utilization), to OpenFlow (e.g., active flows, number of packets looked-up and matched, etc.), as well as active network monitoring by means of the qMON NFV-based solution.

The conclusions of the document are drawn in **Section 11**.

Finally, **Annex 3** presents the current status of a joint activity between the MATILDA (specifically, ORO with the Smart City Intelligent Lighting System use case) and the SliceNet projects, in which the network and computing slicing framework of the former is replaced by the one of the latter. This collaboration allows showcasing how the MATILDA ecosystem can also be modularly exploited, fostering a synergic collaboration and potential future partnerships within EU research activities.

3 Overall Vision and Work Flow

The foundations of the MATILDA 5G architecture are based on the "*separation-of-concerns*" paradigm, where vertical industries are expected to orchestrate and manage their applications through the **Vertical Application Orchestrator** (VAO). Vertical Applications are meant to acquire custom projections of 5G communications and edge computing services, specifically tailored on vertical application requirements. Such projections, called **5G network and computing slices** [3], consists of a coordinated pool of network services, provided by the **Telecom Layer Platform** and including wide-area connectivity and radio resources, and computing resources to be exposed to vertical applications. It is worth noting that the resources and services, requested in the form of a **Slice Intent** [4] by diverse applications, potentially have heterogeneous performance, programmability, mobility and isolation/sharing needs [5] [6].

The objective of the MATILDA WP4 has been to design and develop a radically new prototypic framework of the Telecom Layer Platform able to support functional and performance requirements, as well as the effective lifecycle management of 5G-ready Vertical Applications. To this end, as earlier specified in [2], [3], the MATILDA WP4 framework consists of a highly modular architecture aimed to dynamically control, abstract, and suitably expose the 5G network infrastructure resources and services to vertical applications in terms of network and computing slices.

In detail, the 5G infrastructure, managed by the MATILDA Telecom Layer Platform through its southbound interfaces, deals with the following main types of resources:

- **Computing resources** (e.g., micro- and/or pico-datacenters), attached to the edge or placed in the core of the Telecom Operator network. Each computing resource is meant to be managed by a **Virtual Infrastructure Manager** (VIM).
- **Wide-area Network resources**, composed of network devices connecting radio access devices, computing resource pools and network peerings towards other provider networks. These network devices are meant to support modern protocols and paradigms (like software defined networking and segment routing) for the rapid provisioning of wide-area overlay networks in an as-a-Service fashion.
- **Physical Network Functions**, geographically deployed as radio access devices, such as g/eNodeBs, Radio Heads, etc.

Within the MATILDA framework, all the above resources might belong to third-party players (i.e., Infrastructure Providers), and be acquired as-a-Service by the Telecom Platform Provider [1].

These infrastructure resources are used by the MATILDA Telecom Platform to provide a complete and efficient lifecycle management of:

- **Base 4/5G network services**, in order to provide connectivity to User Equipment (with or without any vertical application slice present in the system);
- Suitable **computing domains** in a number of geographically-distributed VIMs to host vertical application components within the 5G infrastructure;

- **Application/slice-specific network services** and **WAN overlay networks** to connect the above components to UEs and/or among themselves with the needed functional and performance requirements.

As suggested in [7], to accomplish these main platform functionalities, the MATILDA Telecom Platform has been realized through the composition of five main architectural building blocks represented in Figure 3 and herein listed:

- The **Operations Support System** (OSS), in charge of receiving the slice intents from vertical applications (and more precisely from the VAO), and of coordinating the work of all the other building blocks in the Telecom layer platform to set up and to properly configure base 4/5G network services, network slices, and edge computing resources. The OSS also acts as the main configuration/interfacing point for Telecom Platform Providers. The prototyping of the MATILDA OSS can be considered as one of the most significant outcomes of WP4.
- The **NFV Orchestrator** (NFVO), in charge of managing the lifecycle of the network services composing the base 4/5G services, and of the ones provided to slices in a shared or isolated fashion. The NFVO is also in charge of Day-2 operations for PNFs (e.g., g/eNodeBs). In the MATILDA demonstration pilot, Open Source MANO [8] has been selected as reference NFVO.
- The **Wide-area Infrastructure Manager** (WIM), devoted (*i*) to manage and monitor the wide-area communication resources, (*ii*) to create network overlays to be used in a shared or isolated fashion by vertical applications and base telecommunication services, as well as (*iii*) to provide information on which resources (e.g., VIMs, PNFs, etc.) can be selected in the distributed 5G infrastructure to create slices/services in order to satisfy vertical application performance requirements (e.g., end-to-end latency, bandwidth, etc.). On its southbound, the WIM can be interconnected to networking devices either in a direct fashion or through SDN controllers. In the MATILDA demonstration pilot, the WIM has been realized through an extended version of the Ericsson Network Manager [9] (forked from the official product mainstream in order to support a number of MATILDA solutions).
- The **Virtual Infrastructure Manager** (VIM – one instance per each distributed computing facility) is mainly devoted to abstract and expose computing, storage, and networking capabilities of datacenters within the 5G infrastructures. It has the key role of isolating the various tenant domains (i.e., NFV domains and Vertical Applications' ones), as well as of creating shared resources to properly "attach" these domains [3]. In the MATILDA demonstration pilot, the VIMs have been realized through the OpenStack software suite (mainly with the Rocky and Queen official releases).
- The **Wide-Area SDN Controller** (WSC), in charge of interconnecting the control agents of the SDN devices in the wide-area network for monitoring and configuration purposes. It exposes a northbound interface mainly towards the WIM and Telecom layer monitoring frameworks. Within the MATILDA WP4 pilot, the OpenDaylight project has been chosen as reference WSC [10].

Even if the VIM and the WSC are defined to be part of the 5G infrastructure layer, they are explicitly considered here for a sake of completeness.

As already stated in previous deliverable reports [1] [2], one of the key and challenging objectives targeted by this Consortium has been to demonstrate the feasibility and the short-term applicability of the MATILDA approach and ideas on emerging 5G infrastructures and tools.

To this end and as shown in Figure 3, the Consortium decided to rely, as much as possible, on the vanilla versions of existing and emerging reference projects/software suites in the 5G field. As previously reported, the NFVO, the VIM and the WSC have been realized through the latest (unmodified) versions of Open-Source MANO, OpenStack and OpenDaylight, respectively.

This choice allows to focus the research and development innovations mainly on the MATILDA OSS, whose prototype has been conceived and completely developed within the MATILDA WP4.

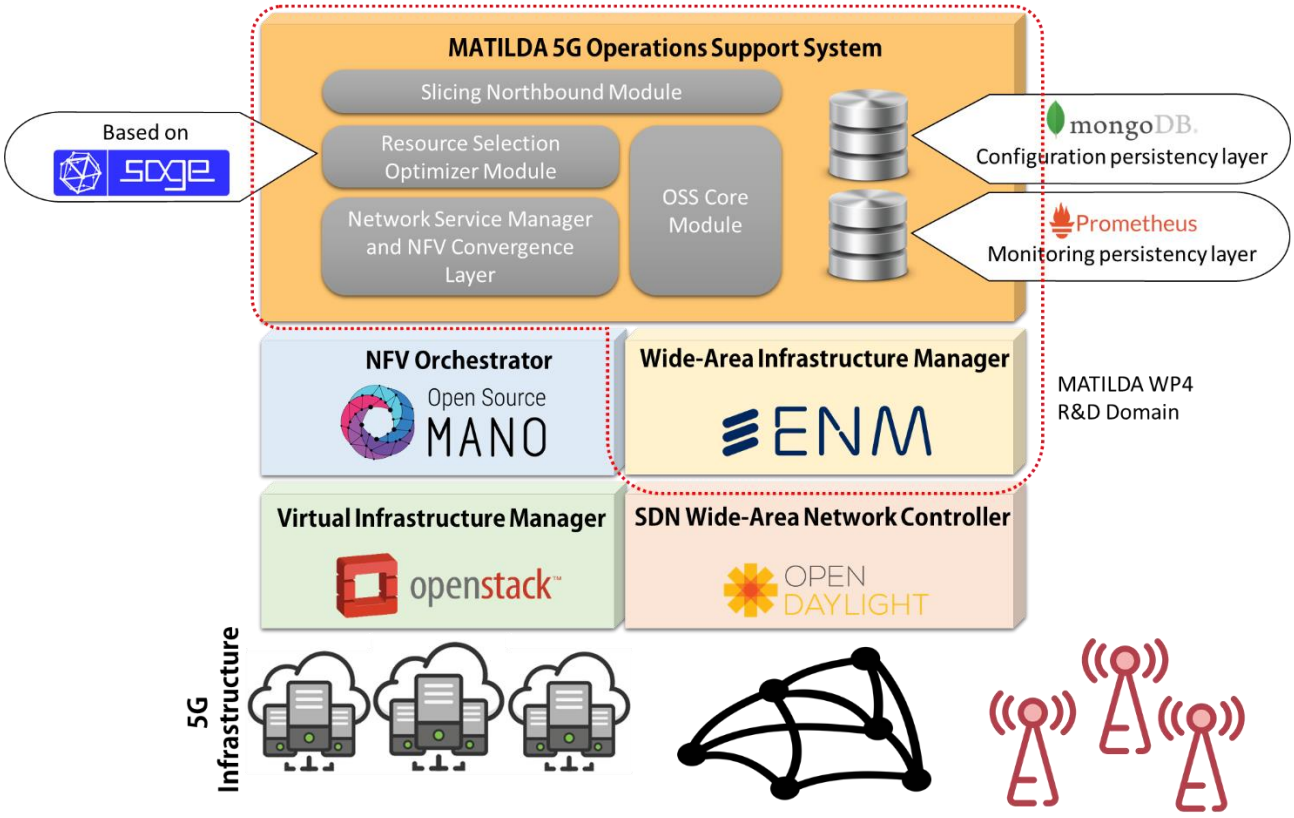


Figure 3: Main building blocks composing the MATILDA 5G Telecom and Infrastructure Platforms.

3.1 Design and Development of the MATILDA Telecom Layer Platform

The MATILDA OSS has been defined in the D1.1 report. As represented in Figure 3, its functional specification identified nine main components, each one with a precise role in the 5G slice creation procedure. In detail, thirteen different steps were defined for replying to the VAO slice intent message with a materialized slice proposal, and further thirteen steps were needed for its instantiation.

The final OSS prototype has not only been developed in a fully compliant fashion with respect to the aforementioned architecture and procedure, but has also been designed to fulfil/support a number of additional functionalities, extremely precious for modern 5G-ready platforms. For

instance, the support for a Telco-layer performance monitoring framework has been added to the original design.

As shown in Figure 3, the prototype development activities resulted in the creation of a suite of five main software services, including all the components originally specified in the OSS definition, and two different databases (MongoDB [11] and Prometheus [12]) acting as persistency layer.

In more detail:

- The OSS Core Module includes the three components of the D1.1 functional specification, namely the Slicing Lifecycle Manager, the VIM and WIM convergence layers.
- The NFV Convergence Layer has been merged with the Network Service Manager into a single service.
- The Slicing Northbound Module and the Resource Selection Optimizer (RSO) are univocally mapped between the D1.1 specification and the final software prototype.

This mapping has no implications on the OSS functional behaviour, but it is rather due to software-level rationalization and implementation decisions. For example, the RSO was designed as an independent service in order to give the possibility to:

- change algorithms and policies for resource optimization and slice deployment in a plug-and-play fashion.
- rapidly and easily develop new algorithms through the provision of a suitable software environment. To this purpose, the RSO is built on top of an extended version of *SageMath* [13], an exhaustive and very advanced open-source mathematics software system, which can be programmed in Python or with a domain language fully compliant with MatLab. The SageMath environment has been integrated by adding a number of agents to communicate with the other OSS components (e.g., to send and receive REST messages to/from the OSS Core, to retrieve configuration parameters from the MongoDB, as well as to retrieve performance and status metrics from the infrastructure and/or services from the Prometheus database).

It is worth noting that all the software services composing the OSS have been designed as state-of-the-art cloud native software, i.e., as stateless services (or more precisely services with a state maintained in an external database), inherently parallelizable, and ready to be integrated with the service-mesh paradigm.

In addition to the design and the development of the Telecom Layer Platform, the MATILDA WP4 was also in charge of designing and producing a set of network services and related network functions to be used for setting up the network slices needed in the demonstration pilots. Given that commercial and open-source 5G access and core devices/software were still in early prototyping during WP4 activities, the MATILDA Consortium decided to use 4G technologies with a number of solutions, studied ad-hoc within the WP4 for emulating network slicing capabilities. Every physical and virtual network function (P/VNF) has been released as a package to be on-boarded on OSM, and provided with a V/PNF Manager (in the form of a Juju charm [14], exposing homogenous interfaces to the Network Service Manager for Day-2 configurations. In total, the WP4 produced sixteen P/VNFs to enable the composition of five main types of services including two different 4G Enhanced Packet Core (EPC) implementations

(one monolithic with extended capabilities, and one distributed but with a more limited set of supported capabilities).

Network Services Descriptors (NSD) [15] have not been provided as static packages in OSM, but they have been rather generalized into "**network slice blueprints**." Such blueprints are meant to be defined by the Telecom Operator, and consist of a metamodel representing how many and which types of network services should be created and deployed in order to cope with performance and operational requirements of the various base telecom service or slice intent requests. These metamodels obviously include a reference on which types of VNFs should be applied (on the basis of their nature and of the supported features), and of their Day-2 configuration (i.e., to be properly parsed to produce the configuration file of the software processes realizing the network function in the Virtual Machine).

Upon the peculiarity of the incoming network service request, the NFV Convergence Layer selects the most appropriate blueprint, and on the basis of the metamodel and instantiation requirements, builds on the fly the NSD, onboards it onto OSM, and triggers its (multi-site) instantiation through OSM.

Finally, when all the Virtual Machines and the Juju charms of all the VNFs have been successfully deployed, the Network Service Manager pushes to each of them (through the aforementioned Juju charms) a proper Day-2 configuration, and starts the all the needed processes.

3.2 Deployment of the MATILDA Telecom Layer Platform

Before entering in the details of the different components, the example in Figure 4 provides a simple and high-level outlook on the final deployment to be produced by the MATILDA Telecom Layer Platform, that can help to better and more intuitively understand the role, the relation, and the objective of each sub-entity against the MATILDA WP4 architecture.

In the example of Figure 4, the 5G infrastructure is composed of a wide-area network interconnecting three VIMs and two (4G) eNodeB PNFs. VIMs 1 and 3 are supposed to be "edge" VIMs, in the sense that they are placed geographically closer to the eNodeBs, while VIM 2 is placed in the "core" of the Telecom infrastructure.

On top of this infrastructure, the MATILDA Telecom Layer Platform (TLP), through the NFVO, instantiated a single Public Land Mobile Network (PLMN) as base network service. The MATILDA TLP supports the deployment of multiple PLMNs (with different PLMN identifiers) in order to emulate isolated 5G slices, and to support Over-The-Top (OTT) players.

The PLMN in Figure 4 is composed of a number of NFV Network Services (NSs), whose VNFs are highlighted with the light blue colour, which are deployed in all the three available VIMs. One service, in the core VIM 2, includes a single monolithic VNF implementing the EPC functionalities. This VNF has two main network interfaces, one towards the E-UTRAN networks (where 4G S1 data- and control-plane sessions are received) and one towards the Internet (the so-called 4G SGi interface).

On each edge VIM, a further NFV NS is created for managing and configuring the eNB PNF and for providing S1 bypass capabilities (i.e., the possibility of steering some predefined traffic flows to the L7 application components deployed in the edge, instead of sending them to the "core" EPC).

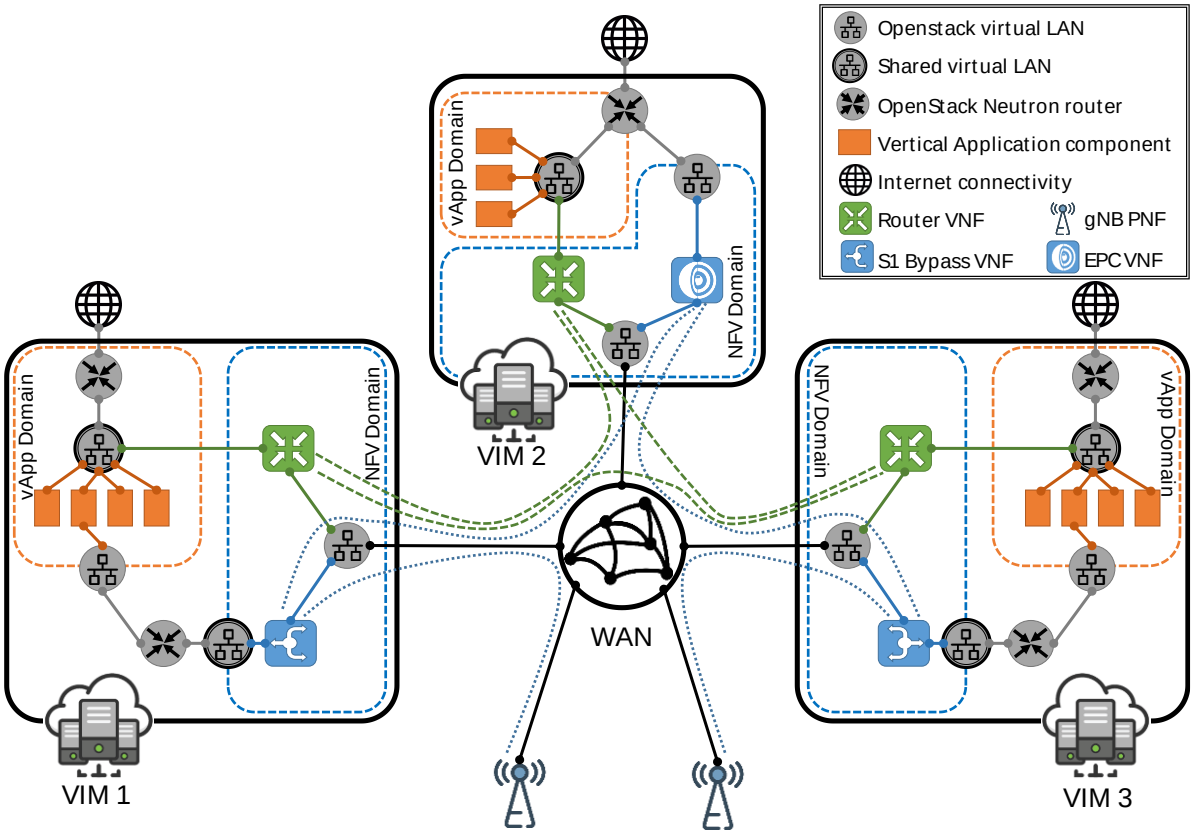


Figure 4: Simple deployment example of a base network service and of a network slice produced by the MATILDA Telecom Layer Platform.

In detail, the S1 bypass VNF has two main network interfaces, one towards the WAN to interconnect with the EPC and the eNBs, and one towards a virtual layer-2 network internal to the VIM. This second network is the one acting as the "attach point" (see Figure 1) between the NFV and Vertical Applications domains at the edge VIMs, and it is shared through a Rule-Base Access Control (RBAC) between the OpenStack projects owned by the VAO(s) and the NFVO.

Since a single PLMN network can host multiple applications, and these applications might require diverse IP addressing in their front-end components towards the UEs, it has been decided to add an OpenStack Neutron Router between the attach point virtual LAN and the one connecting the front-end components of the vApp. Furthermore, this solution also permits to better isolate the front-end components of the various vApps, since it is possible to add to the Neutron Router specific firewalling rules to only allow the connectivity among the vApps and the S1 bypass VNF.

The Neutron Router and the attach point network are created at the bootstrapping of the PLMN network by the OSS (and more precisely through the VIMCL in the OSS Core) before triggering the NFVO. Then, upon the reception of a new vApp slice intent, the OSS creates the virtual LAN for the vApp front-end components, and adds a port to the Neutron Router to connect it with the attach point. The architectural choice to use an OpenStack Neutron Router instead of the more intuitive option of adding a router VNF is due to a current limitation of the ETSI OSM project, which does not allow adding on-the-fly a new port to an instantiated VNF, or reserve a number of VNF ports for future use; in an OSM NSD, all the ports of a VNF should be

explicitly connected to a network, and they cannot be modified without destroying and recreating the NS/VNF instances through new descriptors.

It is worth noting that the WAN is configured to be exposed on all the VIMs as a particular layer-2 network. Therefore, VNFs can attach to this network with standard OpenStack procedures.

The SDN switches composing the WAN are programmed by the WIM and the WSC to enable suitable paths between all the couples of eNB PNFs and S1 bypass VNFs, as well as among all the S1 bypass VNFs and the core EPC. To make this SDN programmability easier, the Consortium decided to apply tunnelling protocols among all the aforementioned P/VNFs.

The use of tunnelling protocols is a common practise in 4G networks, and allows packets crossing the WAN to have the source and destination IP addresses corresponding to the ones of the source and destination VNFs for that path. Therefore, on the same SDN switch, it is possible to discriminate packets exchanged between the eNB and the S1 bypass VNF from the ones between the S1 bypass VNF and the EPC, by only looking at the couple of IP addresses. The instantiated tunnels in the example of Figure 4 are depicted as sketched light blue lines.

Regarding the PLMN EPC, a number of specific Day-2 configurations should be performed by the MATILDA TLP. For instance, at the bootstrap of this VNF, the list of the UE SIMs enabled to attach to the PLMN should be passed to the HSS functionality within the VNF, the proper PLMN identifier and default Access-Point Name (APN) should be created to provide Internet connectivity to all the UEs. Within the MATILDA TLP, a single PLMN can host multiple shared vApp slices.

For each activated vertical application slice, a new APN is created by specifying a default E-RAB (E-UTRAN Radio Access Bearer) with the QoS parameters indicated in the vApp slice intent. These APN should be configured on-the-fly, upon the reception of new slice intent request.

Considerations similar to the ones on the need for specific Day-2 configurations at the service bootstrap, or at the reception of a slice intent, can be also made for all the other VNFs. For instance, in order to build the aforementioned tunnels, each VNF should know the IP addresses acquired by other VNFs in the service. Moreover, eNodeBs should also be configured by making explicit the IP address of the MME acting as anchor point; in order to properly steer the traffic at the edge VIM, S1 bypass VNF should be configured with the list of IP addresses of the enabled vApp front-ends, the list of UEs that can access to that vApp, etc.

In the MATILDA TLP, all these configurations are completely handled by the “Network Service Manager and NFVCL” module in the OSS, and are applied to the VNF instances through the Juju charms (corresponding to the VNFM in the OSM architecture) embedded in their VNFD.

Moving on to the vApp domain, it is an OpenStack project fully dedicated to the vApp and directly controlled by the VAO (and for this reason, these types of projects need a connection to the public Internet to be reached by the VAO). Different vApps in the same VIM will be assigned to different and isolated OpenStack projects.

In Figure 4, the vApp domain is highlighted with a sketched orange box. It is created by the OSS Core through the VIMCL functions, and prefilled with all the needed virtual networks and routers. The identifiers of the created networks are passed back from the OSS to the VAO in the

materialized slice message, by specifying which vApp component should connect to which network identifier.

vApp components might be placed on edge VIMs close to the attach point network, if they have particular performance requirements, or in other VIMs, like the ones in the infrastructure core.

If, like in the case of Figure 4, the vApp components are deployed in multiple VIMs, a further NFV service (whose VNF are coloured in green in Figure 4) should be created and deployed to provide connectivity among the vApp components in the different VIMs. This NFV service is realized by adding a router VNF in each VIMs where the vApp has been deployed, and, similarly to the PLMN case, connecting all the routers through a set of tunnels crossing the WAN. Each router VNF has two main interfaces, one towards the WAN (and used to terminate the inter-VIM tunnels), and one towards the virtual network used to interconnect the vApp components among themselves.

In the presence of this NFV NS, it can be noted that this last virtual network serves as a further attach point between the vApp and the NFV domain, and, as in the previous case, it is shared through the OpenStack RBAC policies.

Day-2 configurations are also needed by router VNFs, not only to properly configure tunnel instances, but also to deal with the IP addressing assigned to vApp components in each VIM. In case of non-overlapping addressing, such VNFs can act as normal IP routers, otherwise 1:1 NAT functionalities would be needed.

It is worth noting that this last NS is application/slice specific, and, consequently, its lifecycle management directly depends on the one of the vApp slice. Different vApp slices will rely on multiple instantiations of this kind of service, and will have diverse deployments (in terms of VIMs selected, number of router VNFs, etc.).

4 The OSS Slicing Northbound Module

The Slicing Northbound Module is responsible for the deployment and lifecycle management of the application graph components over the created network slice. It provides mechanisms for requesting the creation of an application-aware network slice, considering the set of declared computational, networking (in terms of network services) and QoS requirements on behalf of the service provider. Upon the successful creation of the required network slice and the provision of a network slice instance, it supports the lifecycle management of the application graph components. It is also responsible for requesting the deprovision of an established network slice.

The following interfaces are specified for the Slicing Northbound Module, supporting actually the interaction between the VAO and the OSS.

Request for the creation of a network slice based on the slice intent: this request is realised by the VAO to the OSS in order to start a process of slice creation and proceed with the deployment of the application graph when the network slice is going to be available.

URL	**api/v1/oss/sliceintent**
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	• Slice Intent request json, which includes the constraints that are requested by the user.
	
	Example:
	{
	"applicationInstanceID": "580",
	"name": "OSSScenario",
	"callbackURL": "http://localhost:8080/api/v1/callback/slice/580",
	"authenticationDetails": {
	"clientToken": "!telcoprovider!",
	"clientKey": "telcoprovider"
	},
	"componentNodeInstances": [{
	"componentNodeInstanceID": "581",
	"componentNodeInstanceName": "TestCaseMariaDB"
	}, {
	"componentNodeInstanceID": "587",
	"componentNodeInstanceName": "TestCasePhpMyAdmin"
	}],
	"constraints": [{
	"constraintID": "591",
	"interfaceInstanceID": "590",
	"qi": "10",
	"radioServiceType": "1",
	"resourceType": "DELAY_CRITICAL_GBR",
	"allocationRetentionPriorityProfile": 1,
	"minimumGuaranteedBandwidth": 120.0,
	"maximumRequiredBandwidth": 200.0,
	"constraintUnit": "kbps",
	"category": "ACCESS",
	"type": "HARD"
	}],
	"graphLinkNodes": [{
	"graphLinkNodeID": "544",
	"fromComponentNodeInstanceID": "587",
	"toComponentNodeInstanceID": "581",
	"type": "CORE"
	}],
	"dateCreated": "Jun 13, 2018 12:02:04 PM"
	}

Response code	• 200 – OK: Valid response. • 400 – Bad request: The object included in the body is not subscriptable or it is badly formed • 405 – Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. • 503: Service unavailable. In case the OSS is not reachable.

Provision **of a network slice instance**: this request is realised by the OSS to the VAO when the requested network slice is ready. Following, the VAO may initiate a deployment request.

URL	**/api/v1/callback/slice/{ID}**
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	• Slice Response json, which includes the constraints that are requested and the information if they are satisfied or not.
	
	Example:
	{
	"applicationInstanceID": "580",
	"vimDescriptors": [{
	"vimID": "115e1625-da50-4c7d-ba08-d213f6662205",
	"domain": "default",
	"project": "maestro",
	"username": "maestro",
	"password": "!maestro!",
	"endpoint": "http://192.168.3.253:5000/v3/"
	}],
	"componentPlacements": [{
	"vimID": "115e1625-da50-4c7d-ba08-d213f6662205",
	"componentNodeInstanceID": "581",
	"attachmentPoints": [{
	"graphLinkNodeID": "544",
	"attachmentPointIdentifier": "26ee5637-316d-48bd-bcb6-183dccb43444"
	}]
	}, {
	"vimID": "115e1625-da50-4c7d-ba08-d213f6662205",
	"componentNodeInstanceID": "587",
	"attachmentPoints": [{
	"graphLinkNodeID": "544",
	"attachmentPointIdentifier": "33ae68d9-4543-46eb-be00-653e1288af27"
	}]
	}],
	"constraintSatisfactions": [{
	"constraintID": "591",
	"satisfied": true,
	"constraintType": "HARD"
	}],
	"dateCreated": "Jun 13, 2018 12:02:14 PM"
	}

Response code	• 200 – OK: Valid response. • 400 - Bad request: The object included in the body is not subscriptable or it is badly formed • 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. • 503: Service unavailable. In case the OSS is not reachable.

Deprovision of a network slice: this request is realised by the VAO to the OSS in case that the created network slice is no longer required (e.g. no further application graph is going to be deployed over it). Based on the request, the allocated resources are released and the network slice instance is deactivated.

URL	**/api/v1/oss/sliceintent/{ID}**
Type	DELETE
Headers	Accept: application/json Content-type: application/json
Parameters	• Slice Delete request, which includes in the path the ID of the slice that need to be deleted
Response code	• 200 – OK: Valid response. • 400 - Bad request: The object included in the body is not subscriptable or it is badly formed • 405 - Method Not Allowed: The method is not allowed for the requested URL. In case DELETEis not specified in the call. • 503: Service unavailable. In case the OSS is not reachable.

5 The OSS Core Module

The OSS Core module is the central building block of the OSS against which all the other OSS modules should register and directly communicate with. The module also offers OSS external interfacing towards the VIMs and the WIM(s), since it includes the VIM and WIM convergence layers, as well as an interface for operations directly driven by the Telecom operator (e.g., for triggering the setup of base network services). It is worth noting that, for ease of readability, the WIM convergence layer will be described, along with the WIM, in Section 8.

From the software engineering perspective, it has been developed in C++ through a highly modular, asynchronous, and multi-threading architecture to facilitate future extensions, like for instance the support of new VIM and WIM interfaces. Moreover, the module has been realized as a cloud-native stateless service: all the status and configuration information regarding the slice intents, the materialized slices, and the base network service are stored in specific "*collections*" in the MongoDB persistency layer.

Moving on to the functional perspective, as shown in Figure 5, this module can be defined as a high-performance session manager and message dispatcher based on the CRUD (Create, Read, Update, and Delete) paradigm. When a new request is received, the OSS Core Module parses it and selects the proper session template to be applied. Each session template consists of a workflow of (sequential) actions to be requested to the other registered OSS modules, or to external architectural building blocks (i.e., the VIMs and the WIM(s)). Two highly representative examples of implemented workflows are the phases 1 and 2 of the 5G Slice Creation process, which have been introduced in the D4.1 report [2].

Further workflows implemented in the MATILDA prototype, like, for example the ones for creating and setting up base 4G/5G networking services, are not explicitly reported in this document, since they are very similar to the ones for the 5G slice creation. In this case, the main difference mainly consists on the fact that the requests do not arrive from the 5G Slicing Northbound Module, but from a simple user interface dedicated to the Telecom provider.

Thanks to the undertaken architectural approach, the OSS Core module is mainly in charge of managing the logic and the sequence of actions (and of their result) of the other building blocks. Policies, optimization and reinforcement algorithms are not integrated into this module, but they are rather delegated to the OSS RSO module. This choice allowed to better organize the internal architecture of the OSS, as well as to provide new workflows without the need of providing policies/optimization algorithms, and vice versa.

The various Convergence Layers integrated into the OSS Core Module can be seen as parsing libraries/plugins that allow to translate requests or replies from the other modules (and their specific interface protocols/metamodels) to an OSS-specific information metamodel of the data stored in the persistency layer.

It is worth noting that, due to the same nature of this module, the OSS Core consumes a very small subset of this metamodel in order to identify error or exception messages, or to keep decisions during the defined workflow processing. The rest of information is meant to be consumed by the other MATILDA architectural components.

The specifications of interfaces and REST messages used by the OSS Core in the implemented workflows, whereas non-standard and specifically designed for the MATILDA framework, are

reported within the sections dedicated to the other modules or OSS-external building blocks communicating with it.

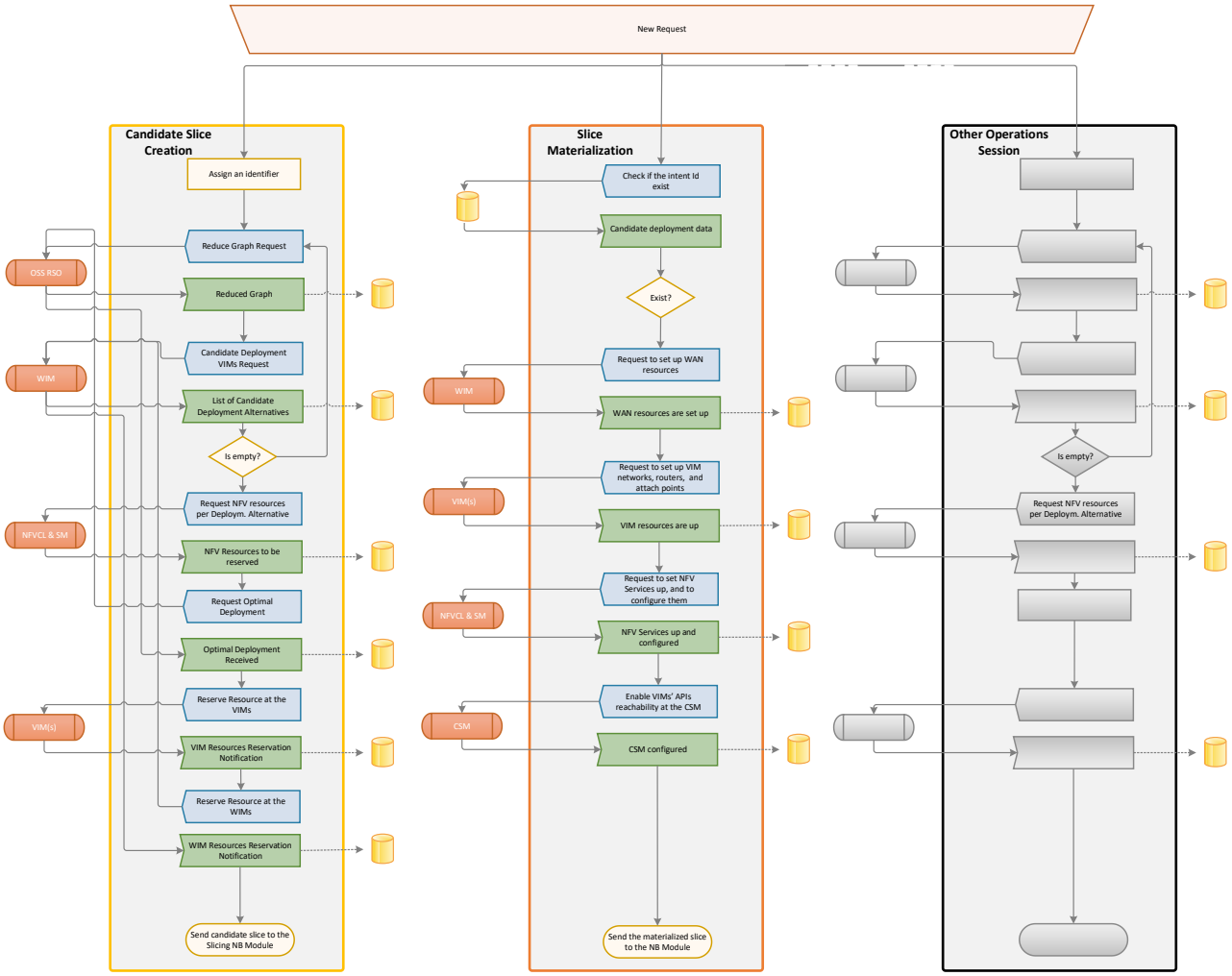


Figure 5: Simplified representation of the OSS Core module functional structure, and of the management of the various asynchronous sessions/workflows by dispatching requests and awaiting for reply messages from the other building blocks in the Telecom Layer Platform. The OSS Core module also stores any partial information retrieved at any intermediated step into the MongoDB persistency layer. It can be noted that the two first workflows correspond to the two phases of the vApp slice creation as defined in the D4.1 report.

6 The Resource Selector Optimizer

As introduced in Sections 3 and 5, the RSO module is in charge of providing optimization and reinforcement policies/algorithms to be executed in the OSS. Since these algorithms might have heavy computational requirements, can run in parallel, and should provide results with short and finite time-horizons, the RSO is specifically designed to scale horizontally. Multiple instances of the RSO can register themselves against the OSS Core Module by indicating which algorithms and policies they provide. When a certain algorithm/policy has to be applied in a workflow, the OSS Core Module triggers the proper RSO instance providing it.

As already discussed in Section 3, each RSO instance is based on an image of the open-source SageMath project. Adopting such an advanced mathematical framework (which includes well-known software packages like R) as is possible to avoid the porting of the algorithms from an environment where they are designed and preliminarily tested through simulations directly (like MatLab, Mathematica, Octave, etc.) to the production environment. Unifying the two environment the integration effort (and related bugs) can be almost zeroed, while providing a familiar and comfortable ecosystem to mathematicians and scientists working on algorithm design.

The base image containing SageMath has been extended with agents and libraries to communicate with the OSS Core and the persistency layer. In detail, as far as the persistency layer is concerned, the RSO can access to:

- the MongoDB database to write produced data, or to retrieve information and configuration data/requirements on the slice intents, on the services, and on the infrastructure elements;
- the Prometheus database to retrieve performance metrics stored by the OSS monitoring framework (see Section. 10).

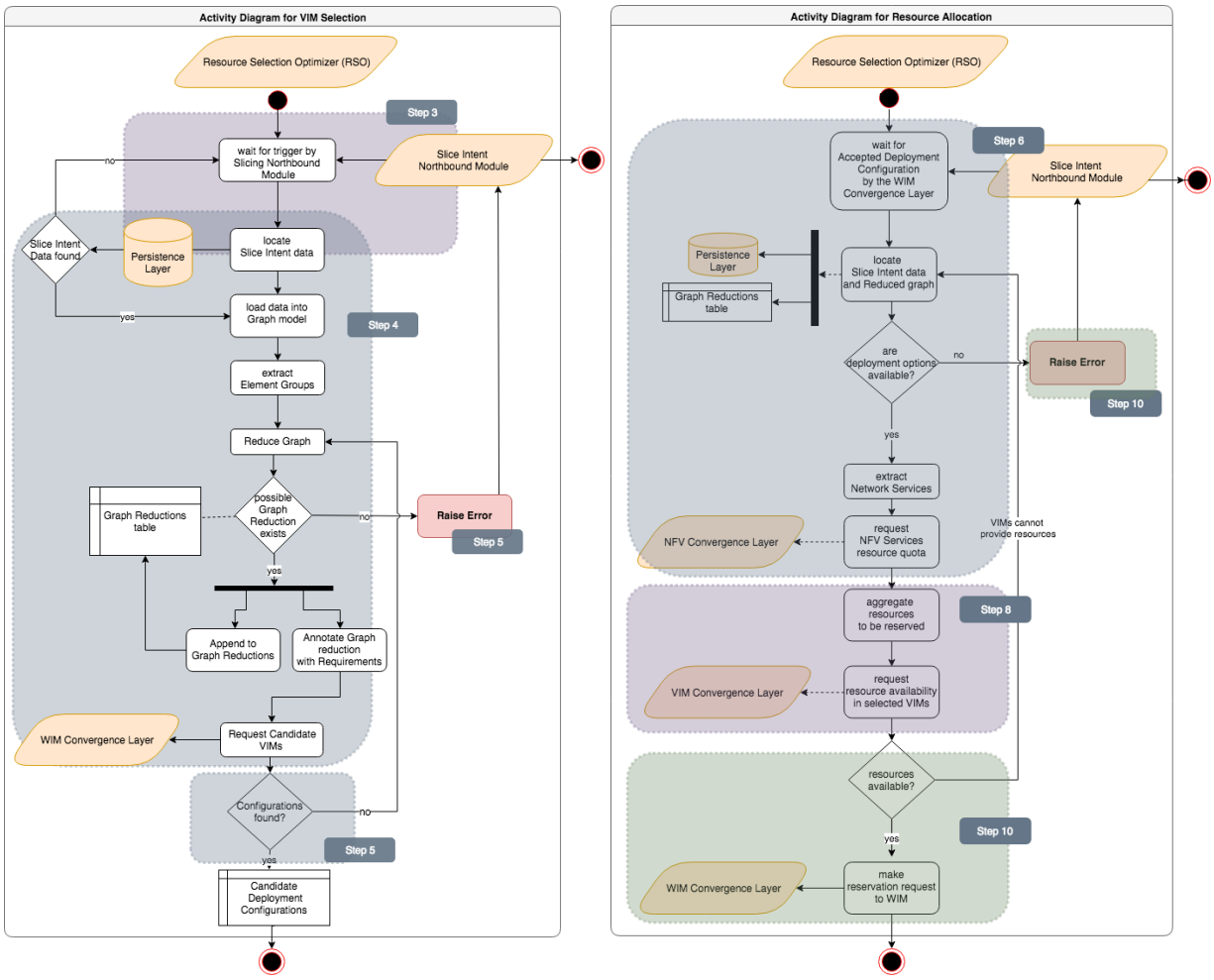
Passing to the prototype development, three submodules has been provided in the RSO to cope with the required operations for the phase 1 of the 5G slice creation procedure. These submodules include algorithms for the vertical application graph reduction (Step 4 of the 5G slice creation procedure – see Figure 6), the forecasting of resource utilization at VIMs, and the placement optimization of the (reduced) vApp graph (both applicable to the Step 10 of the aforementioned procedure), respectively.

Other steps originally assigned to the RSO (i.e., Steps 6, 10, and some parts of Steps 4 and 8 as well with reference to Figure 6) have been delegated to the OSS Core, since they just include simple interactions with the WIM, the VIMs and the NFV Convergence Layer to retrieve the list of candidate deployment VIMs, the amount of VIM resources required by the needed NFV services, and to reserve the resources at the infrastructure for the candidate materialized slice, respectively. All these data are made available to the RSO by the OSS Core through the MongoDB database.

Given the application graph and the link constraints specified in the slice intent originated by the VAO, at Step 4, the ***Graph Reduction*** submodule aggregates application components according to a set of link QoS constraint thresholds, generating a reduced graph composed of "macro nodes". The vApp components in the same macro-nodes will be treated as an inseparable set in the following placement and deployment operations/actions.

It is important to highlight that the threshold parameters determine which components are part of the same macro-node and, then, need to be placed in the same VIM. Hence, tuning these parameters can result in different outcomes. Therefore, these values need to be carefully pondered, for instance by considering the peculiarities of the telecom infrastructure (e.g., the average end-to-end delays among the VIMs).

The ***Utilization Forecasting*** submodule periodically collects from the Prometheus database performance metrics for each VIM registered in the OSS. Such metrics include the utilization of vCPU, RAM and disk available in a pre-determined observation period. The measured metrics' values are used by the ***Utilization Forecasting*** submodule to predict their behavioural trends



Activity Diagram 1 *Activity Diagram 2*

Figure 6: RSO Activity Diagrams 1 and 2 vs the steps of the 5G slice creation procedure. This figure was originally reported in the D4.1 to specify the RSO role in the phase 1 of the aforementioned procedure.

for a given time horizon. This submodule is not associated to a specific step of the procedure in Figure 6, but it is rather a background process that is periodically executed.

Then, at Step 8 in Figure 6, the OSS Core module triggers the ***Placement Optimization*** submodule, which exploit the aggregate resource requirements of the network and computing slice and the metrics forecasted by the previous submodule in order to select the most suitable combinations of VIMs where to place the macro nodes.

The algorithms implemented in the three submodule introduced above are described in more detail in the following.

6.1 The vApp Graph Reduction submodule

This submodule takes as input the graph specifications from the slice intent, as well as the link QoS constraint thresholds. In the current implementation, the former basically includes the `componentNodeInstanceID`, the `graphLinkNodeInstanceID` with the corresponding endpoints (i.e., `fromComponentNodeInstanceID` and `toComponentNodeInstanceID`)

and link QoS constraints – each one identified with a `graphLinkNodeID`, `constraintMetric` (e.g., delay, jitter, packet loss and throughput), `constraintValue`). The idea is to jointly consider the constraints defined in the slice intent, and group application components in such a way that the thresholds are met (i.e., *delay ≤ Delay_th*, *jitter ≤ Jitter_th*, *packet loss ≤ PacketLoss_th* and *throughput ≥ Throughput_th*). The output is then a reduced graph whose specifications in a similar format as the input.

For instance, consider a simple slice intent with 4 application components (`componentNodeInstanceID` ϵ {1, 2, 3, 4}) interconnected by 4 links (`graphLinkNodeInstanceID` ϵ {445, 456, 480, 500}), with only delay constraints (in ms). Figure 7 shows how the resulting reduced graph changes with the threshold.

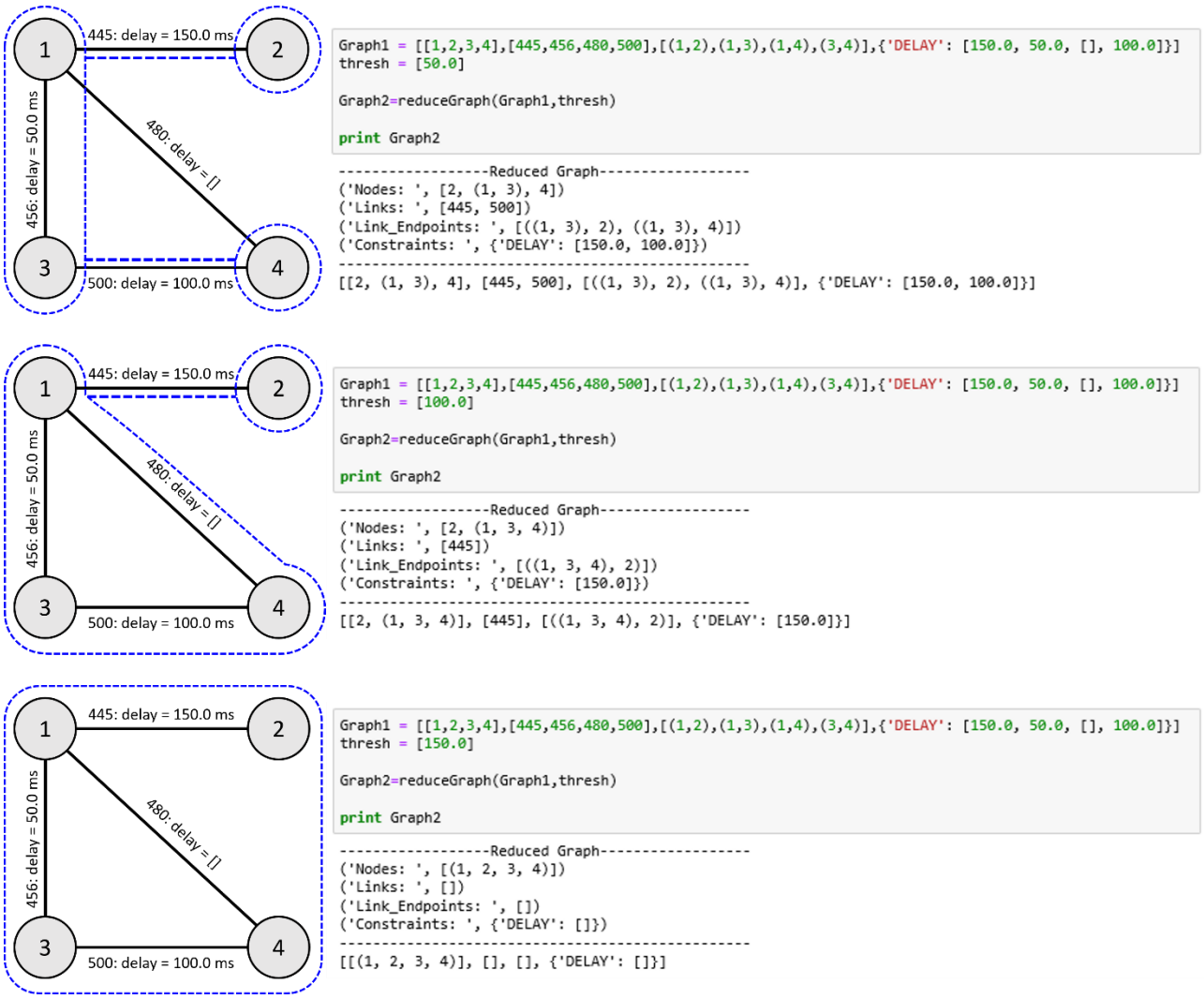


Figure 7: vApp graph reduction results according to different threshold values.

6.2 The RSO Utilization Forecasting submodule

Given the list of candidate VIMs from the WIM Convergence Layer, it is necessary to first define a set of metrics to be used in the selection of the most suitable VIMs for the macro nodes. With this in mind, we consider the monitoring metrics on vCPU, RAM and disk utilization of the

candidate VIMs available in Prometheus for the previous observation period (e.g., time series data of the last 3 weeks), specifically the amount of free resources, the overcommit ratio and the actual usage.

With R's `forecast` library, this submodule models the multi-seasonality (i.e., daily and weekly) of the time series data using the `msts` function. The resulting models are in turn used as input to the `forecast` function to predict the time series values for a certain horizon (e.g., for the next 3 days). For instance, the following figures show the resulting forecasts regarding the vCPUs of a candidate VIM, with the 80% and 95% confidence intervals.

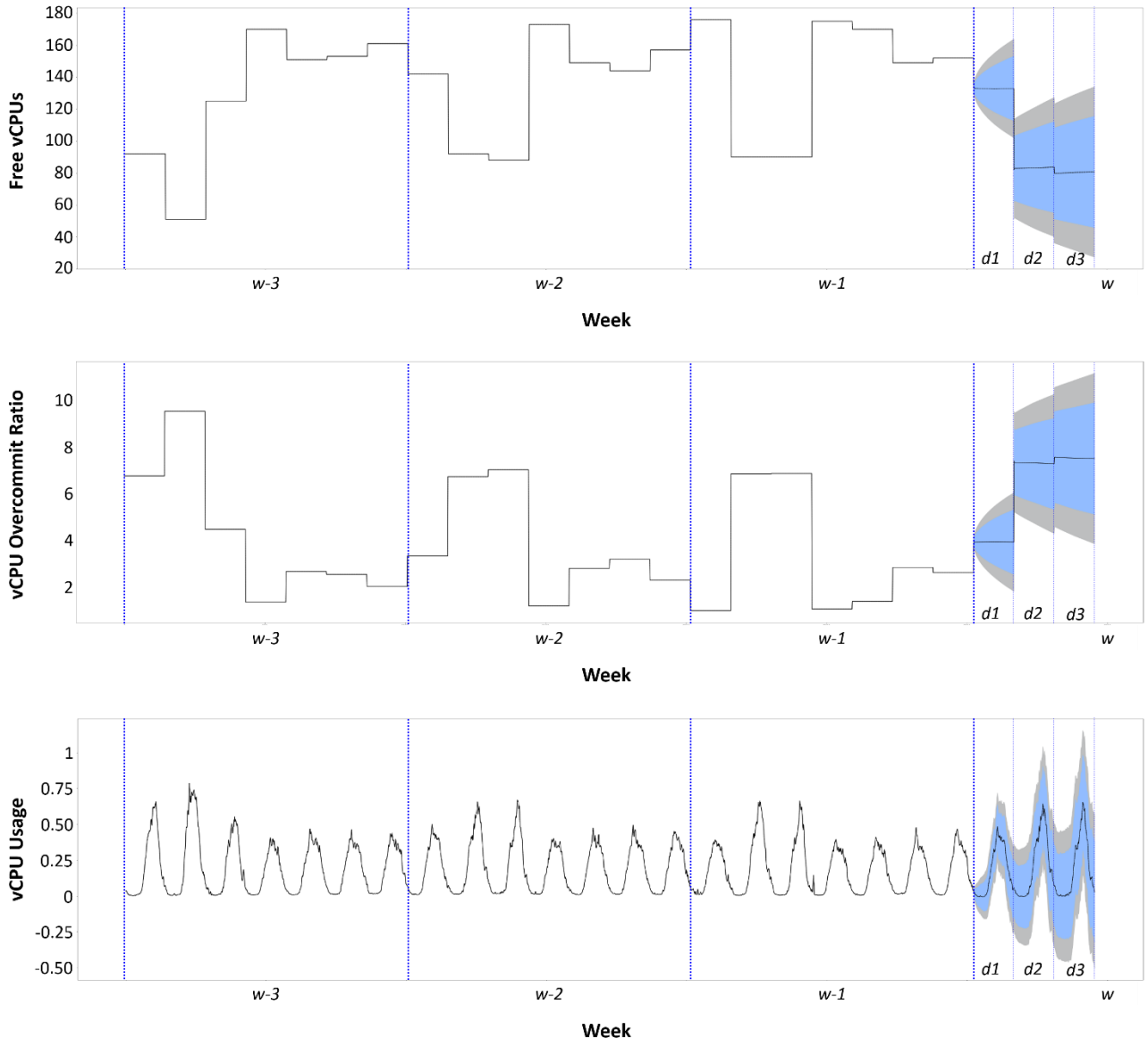


Figure 8: Example of the multi-seasonal forecasting algorithm applied in the RSO according three different time series. The observation period has been set to 3 weeks, and the forecasting horizon to 3 days. The graphs report the average estimate with maximum value and quantile based forecasting bound.

As shown in Figure 8, the idea is to consider either the Maximum values or the Quantiles of the forecasts in the placement optimization, rather than the current values of the monitoring metrics in order to provide the necessary headroom in the allocation of resources, while supporting different dynamics in the time series. Moreover, with the thread-based parallelism in Python, the modelling function can be run in the background, with a running window of training data, in order to keep the models up-to-date. Then, the forecasting function can be called at runtime, and will use the most recently updated models.

6.3 The RSO Placement Optimization submodule

As previously anticipated, this submodule takes as input the aggregate resource requirements (of the macro nodes and the NSs), the Maximum values (or the Quantiles) of the resource utilization forecasts for the candidate VIMs, as well as the deployment options (i.e., mapping between the macro nodes and candidate VIMs). In addition, agreement costs are also considered to support scenarios in which the candidate VIMs have different owners and/or costs.

With the goal of drawing the line between deployment options, we further define the coefficient α as the ratio between the usage and the overcommit ratio, which is computed for the three resource types in a VIM – specifically, $\alpha\_vcpu = vcpu\_usage/vcpu\_ocratio$, $\alpha\_ram = ram\_usage/ram\_ocratio$ and $\alpha\_disk = disk\_usage/disk\_ocratio$.

Basically, two decision rules are used to select the "best" deployment:

(1) ***Find the deployment option with the minimum agreement costs.***

This simply looks at the sum of the involved VIMs' agreement costs. If all the VIMs have the same costs, this aims at minimizing the number of VIMs involved, otherwise the selection will be driven by the individual VIM costs, such that a deployment option with two low-priced VIMs can be selected over one with a single high-priced VIM. Of course, this is under the assumption that all deployment options provided by the WIM Convergence Layer satisfy all node and link QoS constraints.

Note that multiple deployment options can still result with this simple rule. Hence, in such a case, the following rule is applied to further differentiate the options.

(2) ***Find the deployment option with the minimum alpha-weighted resultant.***

For this rule, the amount of free resources per resource type in each VIM are first weighted by their corresponding α coefficients, and by the agreement costs; note that the computations are done using the Maximum values (or the Quantiles) of the forecasted values, as mentioned above. For each deployment option, the results are then aggregated and normalized per resource type, and finally, the resultant is computed. The one corresponding to the minimum resultant value is considered. This is a sort of weighted 3D Best Fit Bin-Packing approach.

While there are high odds that multiple deployment options will correspond to the minimum resultant value, in such a case we suppose that the options have more or less the same expense and randomly select one.

Figure 9 shows a simple example, where 6 deployment options {('V2','V1','V1'), ('V3','V1','V1'), ('V2','V1','V2'), ('V3','V1','V3'), ('V2','V1','V3'), ('V3','V1','V2')} are available for the 3 macro nodes {2, (1, 3), 4}. The proposed approach selects one of these options, involving 2 VIMs: {2: 'V3', (1, 3): 'V1', 4: 'V3'}.

```
Slice_reqs = [[2, (1, 3), 4],{'vcpu':[2.0,8.0,1.0],'ram':[4096.0,16384.0,2048.0],'disk':[40.0,100.0,20.0]}]
VIMs_res = [['V1','V2','V3'],[1,1,1],{'vcpu_free':[15.0,5.0,3.0],'ram_free':[25600.0,8192.0,7168.0],
                                      'disk_free':[180.0,60.0,80.0]},{'vcpu_usage':[0.6,0.8,0.5],'ram_usage':[0.5,0.6,0.5],
                                      'disk_usage':[0.8,0.6,0.6]},{'vcpu_ocratio':[8.0,1.5,2.0],'ram_ocratio':[1.5,1.3,1.1],
                                      'disk_ocratio':[1.0,1.0,1.0]}]
Deplymt_optns = [('V2','V1','V1'),('V3','V1','V1'),('V2','V1','V2'),('V3','V1','V3'),('V2','V1','V3'),('V3','V1','V2')]

BestDeplymt=placeGraph(Slice_reqs,VIMs_res,Deplymt_optns)

print BestDeplymt
```

```
-------------Best Deployment Option-------------
('Nodes: ', (2, (1, 3), 4))
('VIMs: ', ('V3', 'V1', 'V3'))
------------------------------------------------
{2: 'V3', (1, 3): 'V1', 4: 'V3'}
```

Figure 9: Simple example of the placement optimization algorithm applied in the RSO.

7 The OSS NFV Convergence Layer and Network Service Manager

The module including the NFV Convergence Layer and the Network Service Manager (NFVCL+NSM) [2] provides a level of abstraction for the flexible and high-level management of the complete lifecycle orchestration of network services, VNFs and PNFs instantiated in the 5G infrastructure. As stated in Section 3, the NFV Convergence Layer (NFVCL) and the Network Service Manager (NSM) functions have been merged in a single module for two main reasons:

- The two functions are strictly related, since the NFVCL creates and manages the lifecycle of Network Services (NSs) and of the included V/PNFs, while the NSM is in charge of producing and updating the configurations of network processes contained in the V/PNFs.
- The NFVCL maintains information on all V/PNF instances (including their management IP address), and provides a set of easy-to-use primitives to monitor their state and to perform Day-2 actions, such as transferring and committing network processes' configurations in V/PNFs.

The NFVCL+NSM module oversees the NFV end-to-end orchestration within the vApp slice or base network services, creating the required platform in terms of instantiation and configuration of network functions, links or specific services to cope with performance, functional, and deployment requirements of the VAO or of the Telecom Platform Provider.

In detail, the responsibilities of this module are:

1. To keep the communication with the Resource Selection Optimizer (through the OSS Core Module, as described in Sections 5 and 6), during the phase 1 of the slice intent processing, producing the list of resources to be reserved on each VIM for the slice deployment.
2. At the phase 2 of the slice creation procedure, to receive from the OSS Core module (more specifically, from the Slicing Lifecycle Manager function) the materialized slice to deploy, and apply the initial configuration to the NSs that compose the slice.
3. To consume further messages related to 5G slice modifications/update or decommission from the OSS Core module by performing proper operations, commissioning or decommissioning resources, VNFs or NSs at the NFVO.
4. Triggered by the OSS Core module, to manage the entire lifecycle operations of base networking services and related VNF/PNF.

7.1 The Network Service Blueprints

To cope with the above responsibilities and functional objectives, the entire NFVCL+NSM module has been designed around the key concept of the **Network Service Blueprints**.

A Network Service Blueprint can be meant as a generator of NFV Network Service(s), including also all the functionalities to produce proper configurations for the (multi-site) instantiation/deployment, and for the different network services into the VNF instances. Each NS Blueprint is meant to provide a single, high-level network template with a pre-determined set of optional/mandatory capabilities. For instance, as reported in Section 9, an NS Blueprint can provide an entire 4G network (i.e., including both the EPC and RAN functionalities – see the example in Figure 10), or a Layer 3 overlay network composed of OSPF-enabled IP routers acting in different VIMs.

Mandatory capabilities include the list of functionalities that are always provided by any instances produced by that NS Blueprint, like, for instance in the 4G case, standard functions such as mobility support, UE authentication, etc.

Optional capabilities are capabilities that can be enabled at the creation of the blueprint. For example, still using the 4G Blueprint as reference, an optional capability can be the EPC support for the CIoT, or edge computing support (i.e., through the S1 Bypass VNF presented in Section 9). Thus, as shown in Figure 10, depending on their intrinsic nature, optional capabilities might be provided only by properly tuning the configurations of network processes at mandatory VNFs (e.g., like the the software application(s) providing EPC functionalities into the EPC VNF of Figure 10), or by adding further mission-specific VNFs to the NS graph (i.e., the S1 bypass VNF).

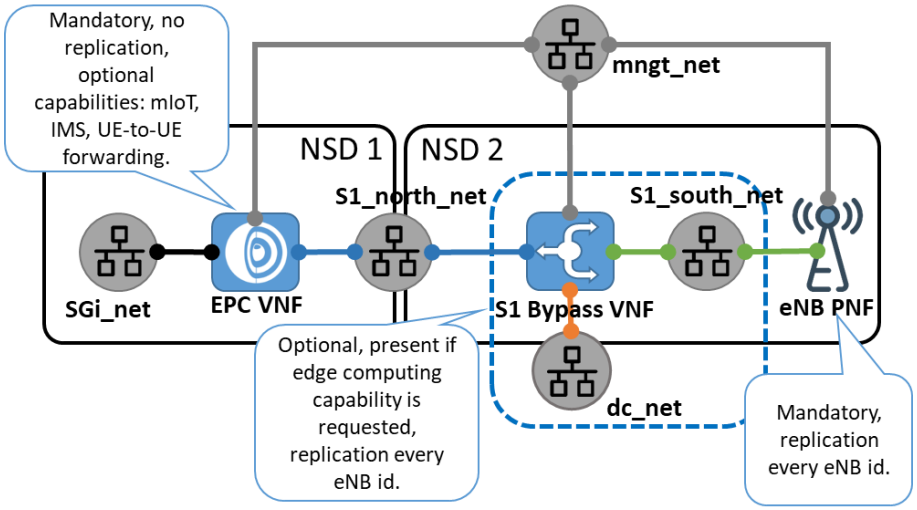


Figure 10: Simple example of the Blueprint template for setting up a complete 4G network with edge computing capabilities. The blueprint contains two NSs. The annotations for optional and mandatory capabilities, as well as the replication, are reported per each included VNF. The blueprint corresponds to the one described in more details in Section 9.

A further crucial abstraction/extension provided by the NS Blueprint concept regards the NS/VNF deployment and graph creation procedures, which allow to overcome some limitations of the current ETSI NFV stack. In detail, according to ETSI NFV specifications (rigidly followed by OSM), a NS Descriptor (NSD) is composed of a pre-determined number of VNF instances connected to internal or external virtual links. Each VNF instance is allocated according to what specified in its VNF Descriptor (VNFD), like for instance the number of virtual machines, their (virtual) network interfaces, the virtual machine instantiation parameters (e.g., the disk image, the number of vCPUs, the amount of RAM, etc.), and the VNF Manager (VNFM – i.e., the Juju charm script) to be applied. Moreover, all the network interfaces of a VNF should be mandatorily connected to a virtual link.

Therefore, the NSD implementing an end-to-end 4G network should make explicit a fixed list of eNB PNF, MME VNFs, S-GW VNFs, etc. and how these v/PNFs need to be connected. There is no possibility of adding further types of VNFs, or tuning the number of the eNBs/MMEs/etc. on-the-fly by using a same NSD. The only degree of freedom left by OSM consists in deciding the deployment VIM of each VNF at the NS instantiation time.

To boost NFV flexibility, the NS Blueprint is specifically designed to maintain a sort of template structure of one or more NSs concurring at the realization of the high-level virtual network. As shown in the example of Figure 10, such templates simply describe the logical structure of the NS graph in terms of types of VNFs, their logical inter-connection, and the virtual networks to be used. Within this template structure, single VNFs or a sub-sets of the template graph can be annotated in order to be replicated as required in the final NSD. For example, in the case of 4G network, the S1 bypass VNF (see Section 9) can be annotated to be replicated for each eNB PNF. Further examples can be found in Section 9. The above annotations are also used in the case of VNF providing optional NS capabilities.

It is worth noting that Blueprints can include multiple NFV NSs for the aim of enabling simple network lifecycle operations at the NFVO, like adding/removing an eNB PNF without destroying the entire virtual network instance (see the OSM limitations described above). Properly splitting the overall Blueprint in multiple NSs (e.g., by adding a NS for each RAN, and one only dedicated to the EPC like in the Figure 10 example) permits a finer control on the lifecycle of VNFs, and to limit the scope of eventual forced reset of a NS (e.g., to update the NSD, or a VNF). For example, with reference again to Figure 10, to add or remove an eNB from the virtual 4G network, it is possible to act only on the NFV NS related to that eNB, without involving the EPC VNF.

Therefore, given the specific instantiation request from the OSS Core Module, a Blueprint is selected and elaborated according to the parameters and requirements within the request. The Blueprint elaboration produces a set of NSDs to be onboarded to OSM in an on-the-fly fashion, and instantiated in the proper VIMs.

The MATILDA NS Blueprint concept is not only limited to the flexible creation and management of NSD of NS Instances (NSIs). It goes well beyond the common vision of a NS as a graph of VNFs and virtual networks, by explicitly considering the network nature of VNFs in the template structure, and how the network processes within the VNF Virtual Machines should be set up and configured. The need for such kind of extension becomes clear considering the common case where the same Blueprint is used multiple times to create a number of similar virtual networks within the same infrastructure: such virtual networks should have different

network-related identifiers (e.g., multiple 4G networks should have different PLMN identifiers), or, in any case different settings of the network functions/processes.

In OSM, this internal configuration of the VNFs can be achieved through the Day-2 operations, which are meant to be executed after the successful deployment of VNFs at the various VIM in the infrastructure. OSM supports Day-2 configurations through Juju-based VNF Managers. These VNF Managers correspond to scripts, called "*charms*", that are included into the VNFD, and onboarded on the NFVO. The charms are defined by the VNF provider, and can expose one or more actions. Each action is supposed to set the values of one or more configuration parameters of the network processes within the VNF.

In this context, the MATILDA Consortium decided to provide the Blueprints with complete capabilities for synthesizing the Day-2 configurations for all the VNFs in a Blueprint, and push them through the OSM VNFM. The Day-2 configuration contents are built on-the-fly according to the final NSIs produced (e.g., how many other VNF of a certain type are present, which are their IP addresses, etc.), the parameters in the request received by the OSS Core to set up the virtual network, and, if needed, tacking trace of all the other instances produced by the same Blueprint (e.g., to produce unique identifiers among virtual networks).

Day-2 configurations include:

- Configuration files/parameters for the network daemons contained in the VNF (e.g., the configuration file of the Amarisoft EPC or of the VyOS router– see Section 9);
- Network Configuration for the VNF Operating System, such as the creation of loopback or tunnel interfaces;
- The list of shell commands to be executed in the VNF.

7.2 Anatomy of the NFVCL+NSM Module

As depicted in Figure 11, the internal architecture of the NFVCL+NSM module comprises two components and a number of software plugins for the various available NS Blueprints. In addition, NS Blueprints include a further level of software plugins for the Day-2 configuration of VNFs, called **VNF Configurators**. The two aforementioned components are:

- the **NFV Service Mapper** (NFVSM), which is acting as interface towards mainly the OSS Core module. It is in charge of handling the requests from the OSS Core module, and of selecting and triggering the proper NS Blueprint to satisfy the performance and functional requirements, made explicit in the request. Depending on the request nature, the NFVSM can either provide information on the resources that a Blueprint deployment can require (as needed by the OSS RSO – see Section 6), or triggering its deployment and providing feedback to the OSS Core, when the NSs are correctly instantiated and all the VNF have been successfully configured. The NFVSM has also access to the OSS Persistency Layer to retrieve any needed additional configurations or data.
- the **NFV Service Setup** (NFVSS), is in charge of communicating with the NFVO Day-0/1/2 operations requested by the selected NS Blueprint. In detail, at Day-0/1, starting from the information generated by the Blueprint, it is responsible to properly generate NFV NSD packages, to onboard them on the NFVO, to trigger their instantiation, and to monitor their deployment. Upon the successful fulfilment of Day-0/1 operations, the

NFVSS will trigger the selected Blueprint plugin (and more precisely the contained VNF Configurators) for VNF Day-2 configurations. The NFVSS periodically queries the NFVO to retrieve the list of onboarded NSDs, VNFDs, and the status of NSIs and VNF Instances (VNFIs).

With reference to Figure 11, it is worth noting how the NS Blueprint plugins and the NFVSS component are involved in two sequential phases of operations to be performed by the NFVO.

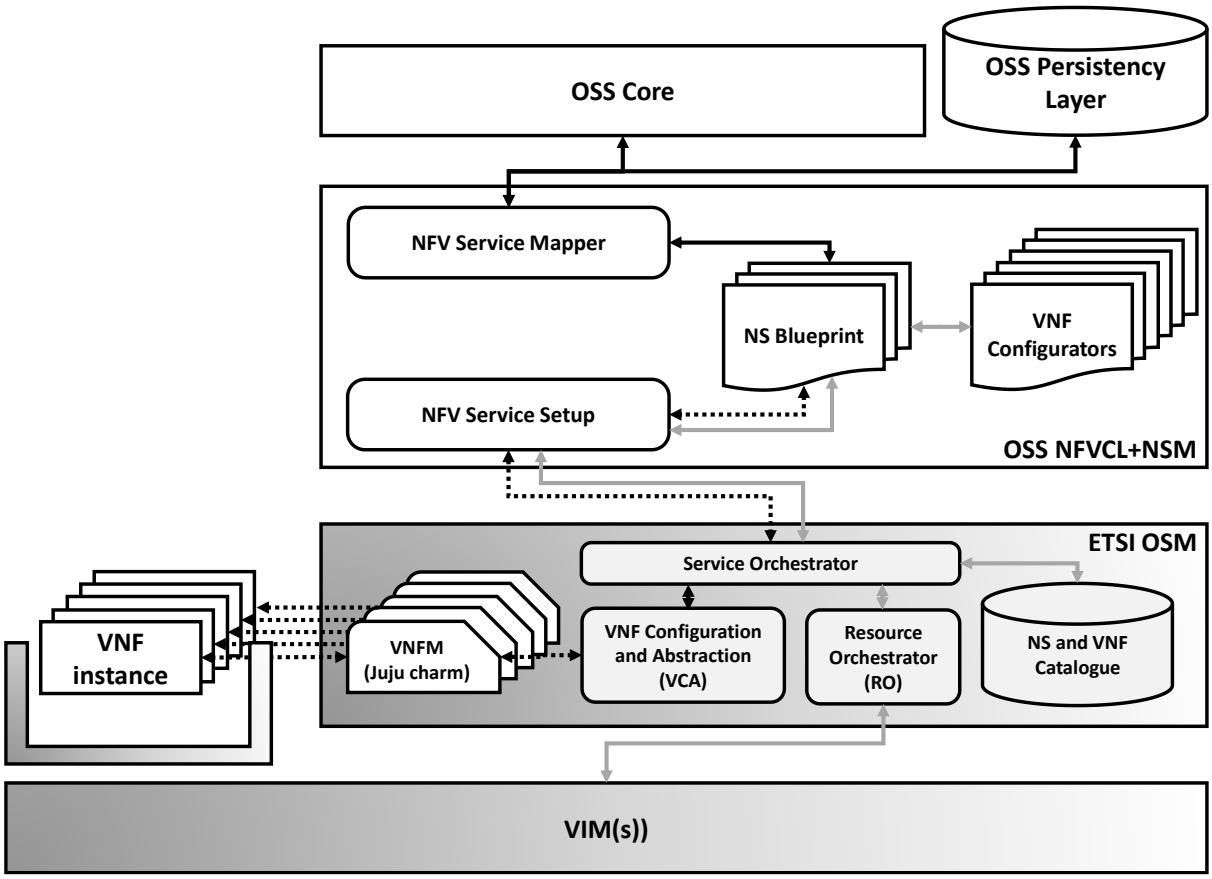


Figure 11: The internal architecture of the NFV Convergence Layer and NS Manager module and its interactions with external entities.

The first phase (depicted with sketched black arrows) corresponds to Day-0 and 1 operations, and, as previously mentioned, mainly regards the synthetization of the NSDs from the Blueprint template structure (see Figure 10), their onboarding, and their instantiation. This first phase concludes when the NFVSS retrieves from the NFVO that all the NSIs have been successfully deployed and are ready to receive the Day-2 configurations. It can be noted that all the operations included in this phase are performed through the interaction between the NFVO and the VIM(s).

The second phase (depicted with solid grey arrows in Figure 11) corresponds to Day-2 operations. When this phase starts, the Blueprint plugin triggers the VNF Configurator of each VNFI, and produces a list of actions to be sent to the NFVO. These actions contain the name of the VNFM primitive to be executed, the identifier of the VNFI, and one or more parameters and values to be passed. Each VNF can be configured through one or more actions/primitives.

In this phase, the NFVSS is responsible of parsing the Day-2 primitive requests, sending them to the NFVO, and monitoring their status. This second phase is completed when all the primitives have been successfully performed.

Differently from the previous phase, during Day-2 operations, the NFVO interacts with the VNFIs via their VNFMs, which are Juju charms running as container in the same OSM Virtual Machine. These containers are created at the instantiation of the VNFs from the charm images contained in the VNFD. Therefore, it can be noted that there is a strict relationship between the VNF Configurators and the charms provided in the VNFs, since the first ones have to produce all the needed information to trigger the VNFM primitives, which are provided by the latter ones.

To make the integration among the VNF Configurators and the VNFDs easier, the MATILDA Consortium decided to equip as many VNFMs as possible with three specific primitives, namely "*set-config*", "*set-start*", "*set-stop*". In detail, the *set-config* primitive accepts only one parameter, named "*config-content*." The value of these parameter is expected to be a YAML-formatted string whose contents depend on the specific nature/type of VNF (see Section 9). These YAML-formatted string are parsed and elaborated by the VNFM (/Juju charm), to produce specific commands to be sent to the Virtual Machine(s) composing the VNFI.

As in the previous OSS building blocks, the overall software architecture of this module has been specifically engineered to be a stateless cloud-native service, and to flavour extensibility towards other NFVOs, network services, VNFs and PNFs. It is completely developed in Python with an object-oriented style. The plugins for the Blueprints and the VNF Configurators are based on the inheritance of a base class, already providing base common functionalities.

As discussed in Section 9 in more detail, the current prototype supports ETSI OSM as NFVO (OSM releases 4, 5 and 6 have been successfully tested), and Day-0 to Day-2 operations for five types of NSs and sixteen V/PNFs.

The remainder of this section describes the NFVSM and NFVSS components, and their internal procedures in further details. The complete reference of the APIs used by these components can be found in Annex 1.

7.3 The NFV Service Mapper

As initially described in D4.1 [2], the NFV Service Mapper is the component in charge of selecting the most appropriate Blueprint inside the MATILDA Blueprint catalogue and calculate the amount of resources to be used in each node for the Blueprint selected, once a bootstrap message is requested by MATILDA OSS.

The NFVSM sequence diagram is shown in Figure 12.

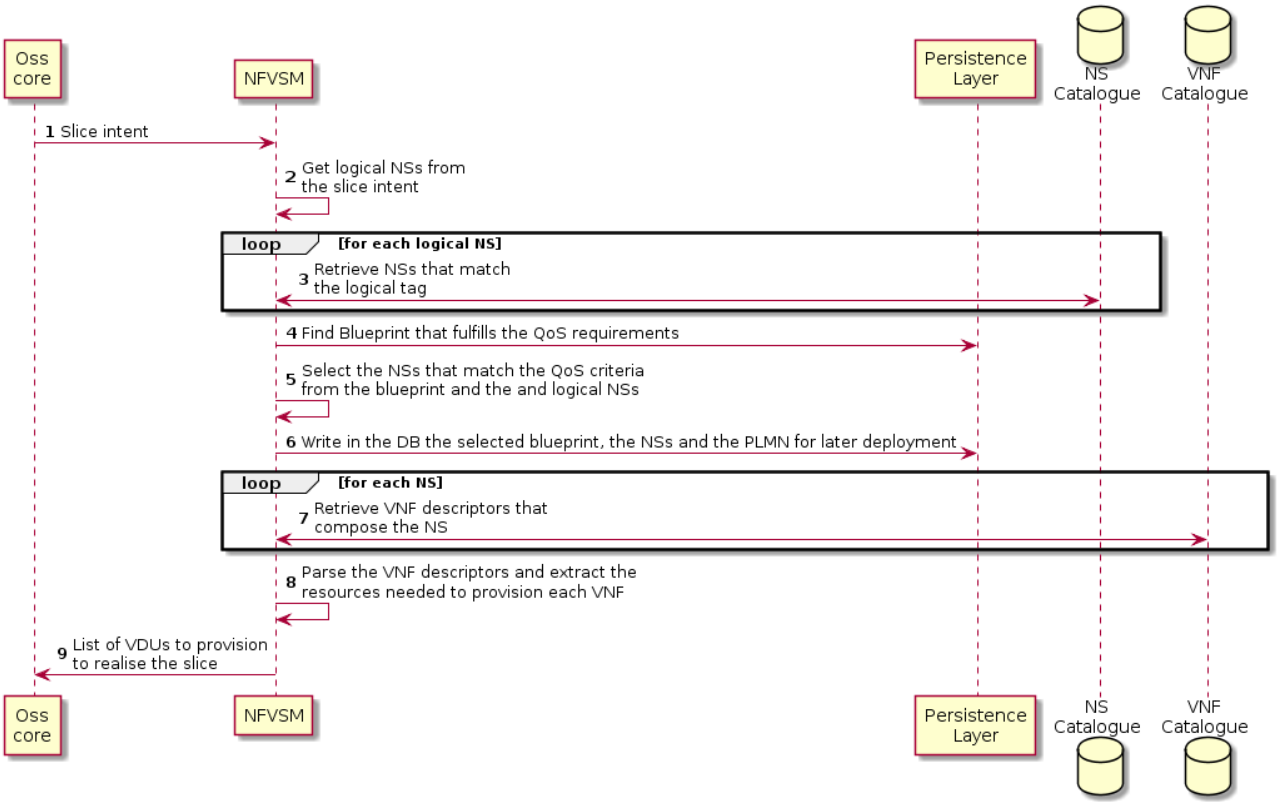


Figure 12: NFV Service Mapper sequence diagram.

7.4 NFV Service Setup

The mission of the NFV Service Setup (NFVSS) is to cover the instantiation, operation and deprovision of the MATILDA Network Slices [2].

The NFVSS will communicate with OSM using the Os-Ma-Nfvo interface **Error! Reference source not found.**, whose RESTful protocols specification is described in ETSI GS NFV-SOL 005 **Error! Reference source not found.** and implemented in the OSM northbound interface component.

The NFV Convergence Layer handles the instantiation requests from the OSS Core module and passes the request to the NFVSS to manage the service instantiation and initial configuration with the purpose of deploying all resources negotiated in the slice intent process, allowing all the necessary configuration without manual actions and providing the attach points for network flows to be terminated.

The first sequence diagram in Figure 13 illustrates the bootstrap message process used to deploy a virtual 4G network from the Blueprint template structure shown in Figure 10. The message contents correspond to the ones in "*bootstrap_4G*" message reported in Annex 1. The final deployment is corresponding to the ones reported in Figure 4 and Figure 1, and is composed of:

- One instance of the Amarisoft EPC VNF, that is always deployed in one of the core datacenters. This VNF is allocated in a separate OSM NS with respect to the other ones.

- One or more Bypass VNFs, each one attached to one eNodeB PNF component, and is always deployed in the edge datacenter where the eNodeB is attached. One of the network attached to this VNF will act as "attach point".
- eNodeB PNF that provides radio access to a geographical zone that has a 3GPP Tracking Area Code (TAC) related to.

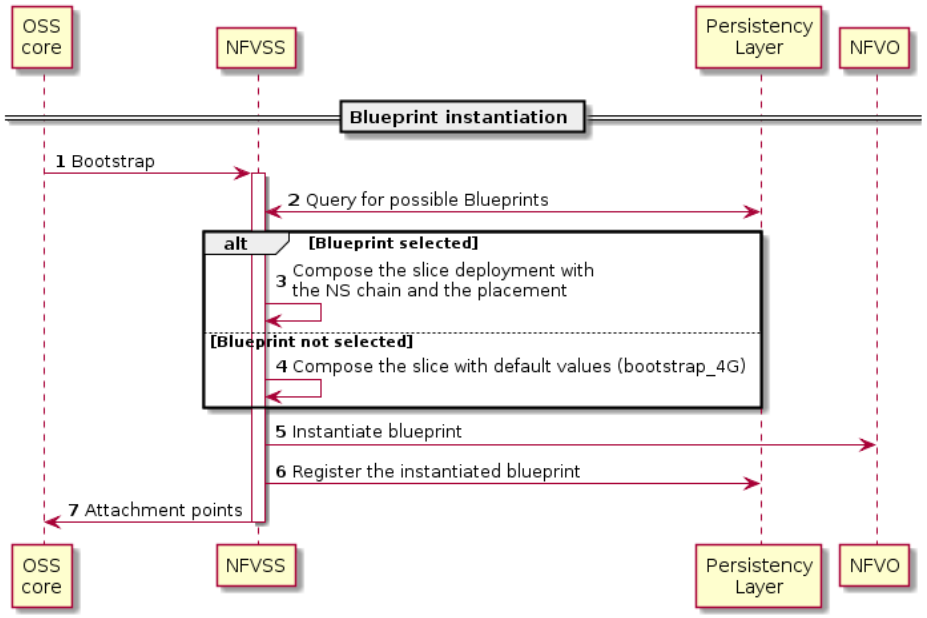


Figure 13: Blueprint sequence diagram.

For the sake of simplicity, but without any loss of generality, the MATILDA Telco Layer prototype makes the assumption that each tracking area code is served by one single eNodeB.

As described in Section 7.1, the NFVSS will create an NSD for the EPC VNF, and one for each TAC including one instance of the S1 Bypass VNF and the eNB PNF. Also, the NFVSS will deploy each couple Bypass-eNodeB in the proper edge datacentre (as indicated in the bootstrap_4G message) to make them able to provide service in each area.

Once these components are instantiated and Day-1 operations complete, all the information about the network services deployed, status of the Juju machines, IP addresses of the instantiated network functions, placement of each component, etc. are registered in the Persistency layer. This is a potential source of information about the status of the slice that is provided to the OSS dashboard.

Therefore, once the bootstrap message is correctlsy processed, vApp slice creation messages can be received. The purpose of these messages is the creation of slices over the common virtual 4G network identified with a specific PLMN value. The diagram of sequence of this message is described in Figure 14, and an example of the message is Annex 1.

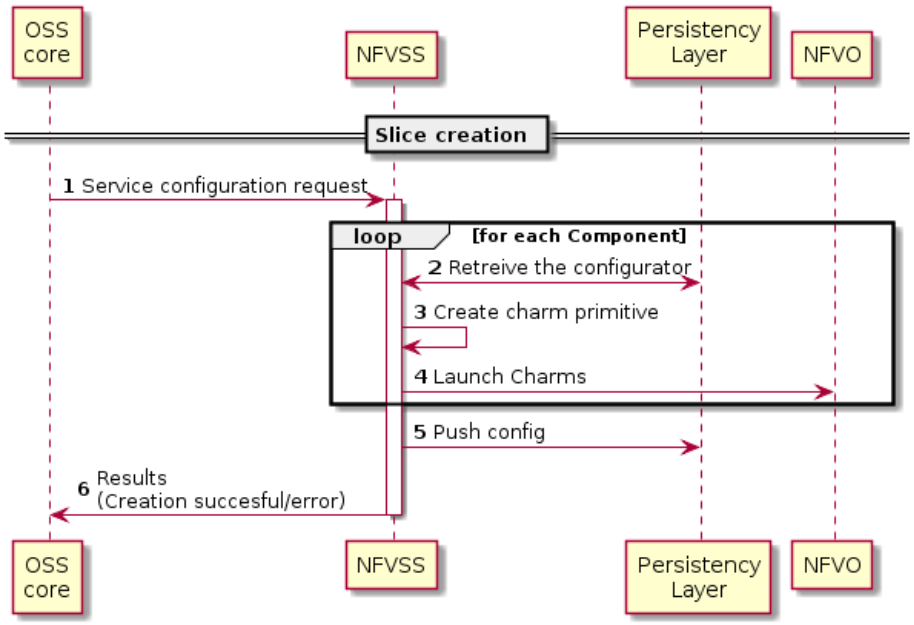


Figure 14: Slice creation sequence diagram.

As described in Section 7.1, the NFVSS oversees the retrieve of the configuration defined by VNF Configurators in the Blueprint associated to the 4G network, and send the list of Day-2 primitives to the VNFMs (i.e., Juju charm) of every VNF involved.

In case of deploying a slice over the 4G blueprint shown in Figure 10 and discussed in Section 9 in detail, the charm primitives perform the following operations:

- In the EPC VNF, the slice is simulated by adding a new Access Point Name (APN) to the component, with the proper parameters (e.g., QCI) in the default radio bearer.
- For the S1 Bypass VNFs corresponding to the TACs involved, a set new S1 bypass rules are added in the Bypass configuration in order to de/encapsulate from/to the GTP tunnel and steer the packets from UE to the front-end components of the vApp and vice versa.

For the logical network functions, that could be explicitly requested by the VAO (e.g., a firewall) or for additional virtual networks that need to be dedicated to the slice (e.g., the L3 overlay network connecting the VIMs where vApp components are placed – see Figure 4), the NFVSS handles these requests using the sequence diagram in Figure 15. Once the service is running correctly and Day-2 configurations are applied, all the related information and data are pushed to the OSS Persistence Layer.

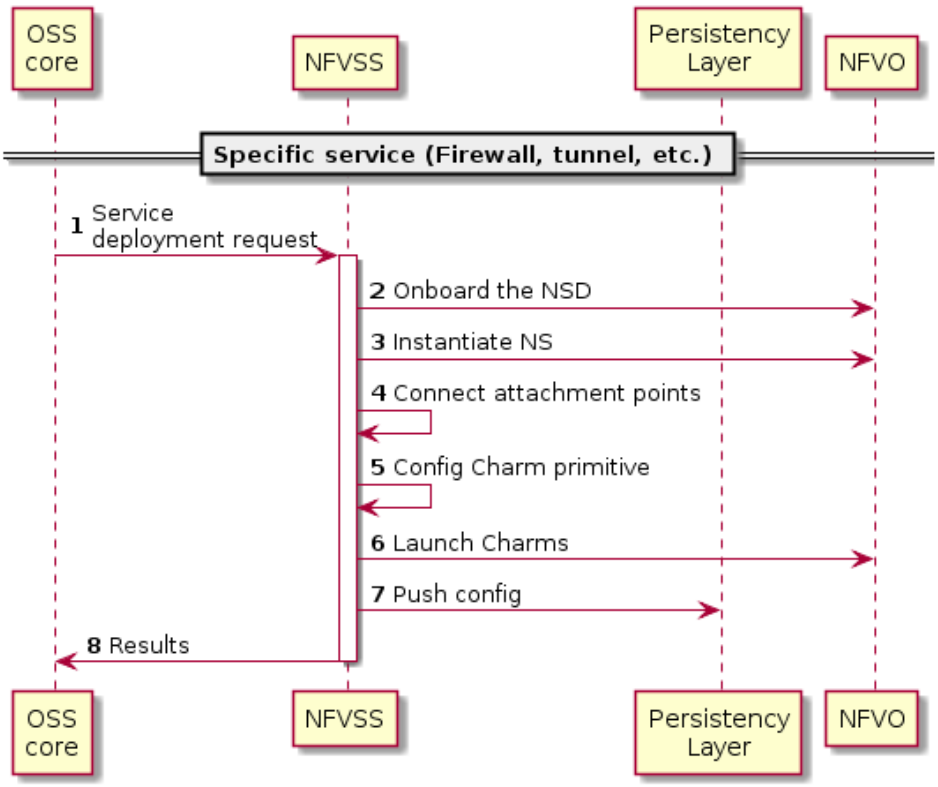


Figure 15: Logical function creation sequence diagram.

8 The Wide-Area Infrastructure Manager (WIM)

The WIM plays a central role within the Telecom provider layer in the MATILDA architecture, since it is the building block maintaining a complete knowledge not only of the status of the wide-area transport network, but also on how and where infrastructure resources (e.g., e/gNodeBs, VIMs, etc.) are attached.

Owing to the highly distributed nature/requirements of 5G network slices and edge computing deployments, this building block is rapidly becoming an essential auxiliary engine to support next-generation Operations Support Systems (OSSs) in novel challenging tasks.

In this respect, 5G-enabled WIMs should go well beyond classical traffic engineering and device management tasks, by supporting OSSs in the feasibility check of vertical applications' (functional and performance) deployment requirements, the selection and the dimensioning of network and network-attached resources.

Starting from the above considerations, the MATILDA Consortium sketched in the D4.1 report the first tentative procedures for the selection of the best paths in compliance with QoS requirements. In this deliverable, the objective is to go deeper in the specification of these components, describing the interfaces and APIs that define the communication between the WIM and other components in the MATILDA OSS and in OSM, the implementative choices, the testing procedures and so on.

The WIM interworks with the extended OSS blocks to help with the VIM selection, according to the desired target latency. After the OSS first tentative selection, it performs the final path

computation accordingly, adding the bandwidth constraints arising from the chosen selection to ensure its compliance to the constraints is maintained. The description will also cover the WIM Convergence Layer REST interface that allows the OSS to ask for the list of potential VIMs, to configure the parameters of the provided choices and to ask for the resource reservation and actuation of the one(s) selected as the best.

All these tasks mainly leverage on the functional blocks provided by the Wide-Area SDN Controller (WSC), a refined top of the art controller based on the OpenDaylight (ODL) framework, on the Path Computation Engine (PCE), and on the WIM Convergence Layer.

8.1 The Wide-Area SDN Controller (WSC)

The Wide-Area SDN Controller (WSC) is the entity in charge of supervising the individual SDN devices in the wide-area to set up the proper paths according to the WIM-RSO outputs as well as the feedbacks from the monitoring frameworks.

Based on OpenDaylight controller infrastructure, the WSC provides a number of control mechanisms that can be used depending on the specific network requirements, for example on the access technology, the latency constraints, the occurrence of migrations, and so on. For the implementation of the MATILDA prototype demonstrator, a simple, IP-based mechanism has been deployed. Moreover, further, more complex flow placement and management mechanisms are available, such as the Multi-Cluster Overlay (MCO) network paradigm [18].

The Multi-Cluster Overlay (MCO) is an SDN-based mechanism for 5G distributed infrastructures allowing the realization of isolated tenant networks. Its excellence resides in the implementation of non-overlapping OpenFlow rules among separate overlays without relying on resource-hungry tunnelling protocols, as well as in the support for connectivity within and among datacenters.

Regarding migrations, the distinct design of the MCO networks makes this solution particularly suited to move software instances in a scalable and dynamic fashion. In fact, the logical resources associated to a tenant overlay are bound to a center and further aggregated in so-called clusters of virtual objects exhibiting similar SLA requirements. This hierarchy allows supporting both migration of single virtual objects between two servers of the same in-network datacenter, and bulk migration of all the virtual objects bound to a cluster center between two in-network datacenters, while supporting reconfiguration of the underlying infrastructure and of overlying services, respectively.

8.2 The Path Computation Engine (PCE)

The Path Computation Engine (PCE) is responsible for computing, storing and retrieving slice solutions. Traditionally, such responsibility boils down to the creation of paths in the traffic engineering context by calculating routes according to a variety of requirements, spanning from connectivity of the links between switches as well as bandwidth and latency constraints.

Instead, in the upcoming edge computing environment, the decision process will be more elaborate, as it will entail not only finding the optimal path between two endpoints, but also the placement of at least one of the endpoints.

Because of such demanding procedures, the PCE was the most challenging block from the implementative point of view for the strict performance requirements, in terms of convergence

time vs scalability, which are indeed some of the most crucial aspects of this application. As a result, the PCE is a module running in a Spring Container ready to be deployed as Micro Service.

In the release implemented for the demonstrator, the PCE is in charge of maintaining topology information, including e-nodes, switches and VIMs, as well as keeping track of allocated resources to exclude them from other path computation requests.

A slice computation request asks the service to compute a set of solutions to the problem of allocating paths across the transport network in order to choose the most convenient way to deploy the given set of VMs to the available VIMs, connecting e-nodes end VMs according to the specified slice topology and constraints. Upon such requests, the PCE computes the best paths in the transport network implementing the links of the slice topology, and then combines them to produce a set of solutions. A solution is a set of paths, one for each link of the slice, connecting e-nodes and VIMs to implement the slice connectivity and also including allocation of the VMs to the VIMs. A solution has an associated cost metric; lower cost solutions are reported first, up to a configurable maximum number. Equal cost solutions can be merged if possible, producing alternative allocations of the same VM to different VIMs.

8.2.1 The Graphical User Interface

The WIM GUI has been conceived during the final implementation phase as a user-friendly tool to verify and monitor all the WIM operations, to gather specific details in a suitable visual way and to best appreciate the WIM powerful capabilities in a graphical format. Figure 16-Figure 18 show three representative views demonstrating the GUI capabilities.

In particular, Figure 16 is dedicated to Network Topology visualization: the topology is represented on a map containing set of nodes (OpenFlow switches, VIMs, eNodeB equipment) and links between them. Nodes and links details are provided to the GUI by the PCE module, and can by displayed just clicking on each entity.

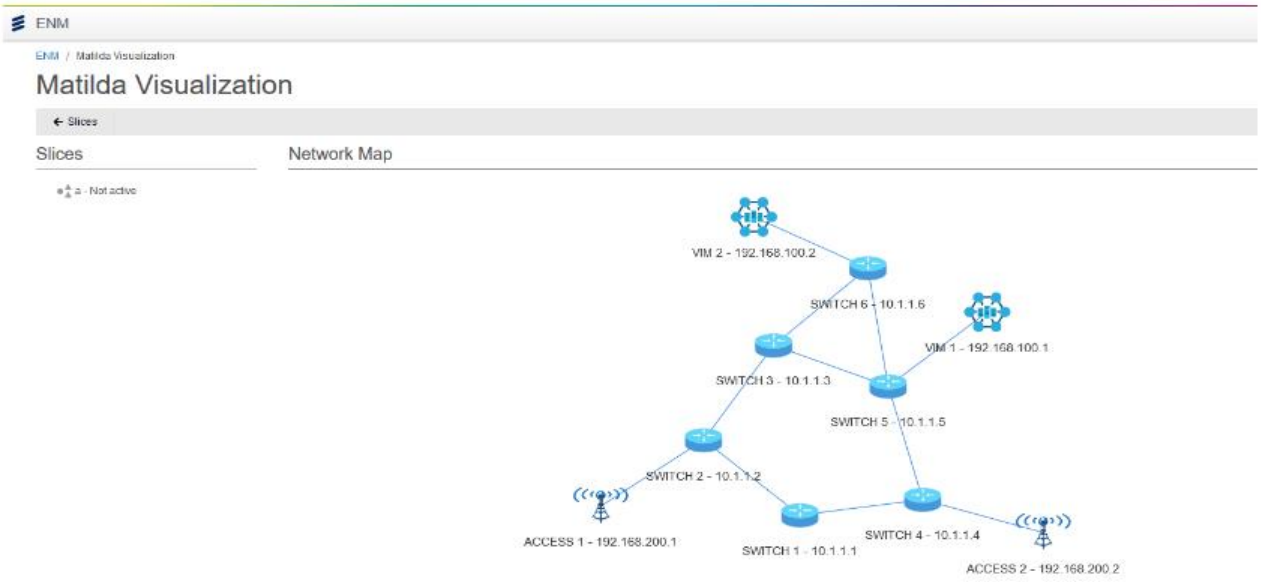


Figure 16: Topology Visualization.

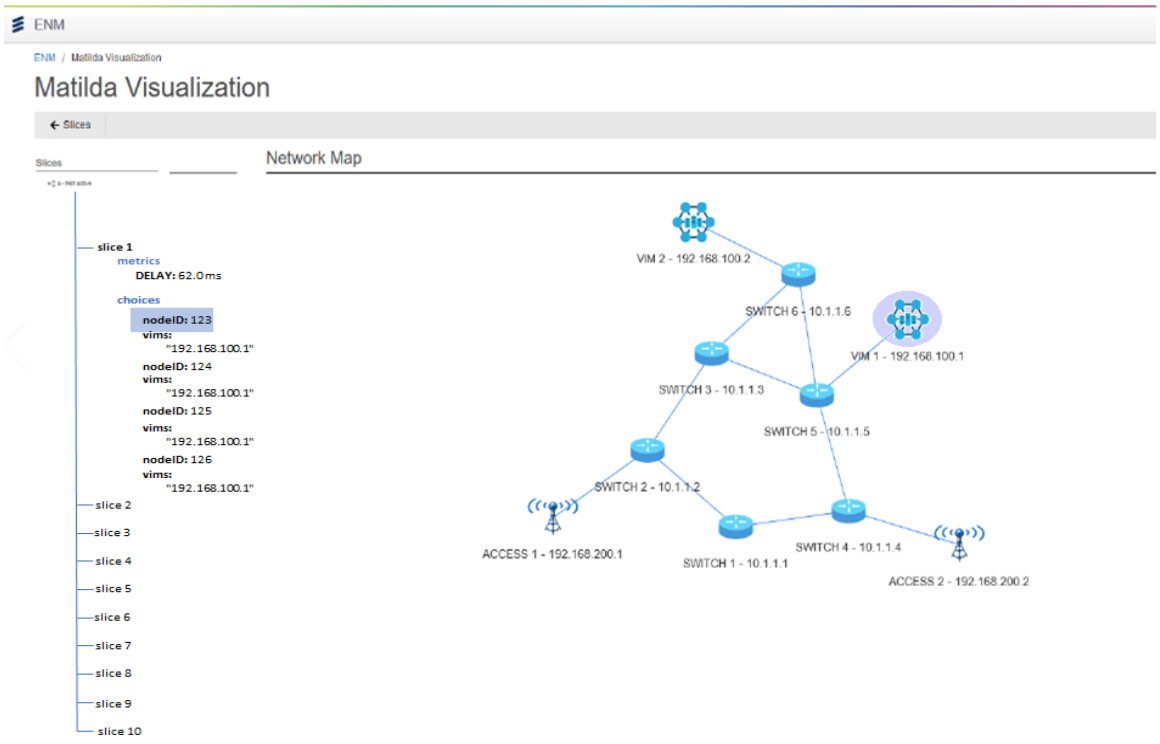


Figure 17: Computed Slice details visualization.

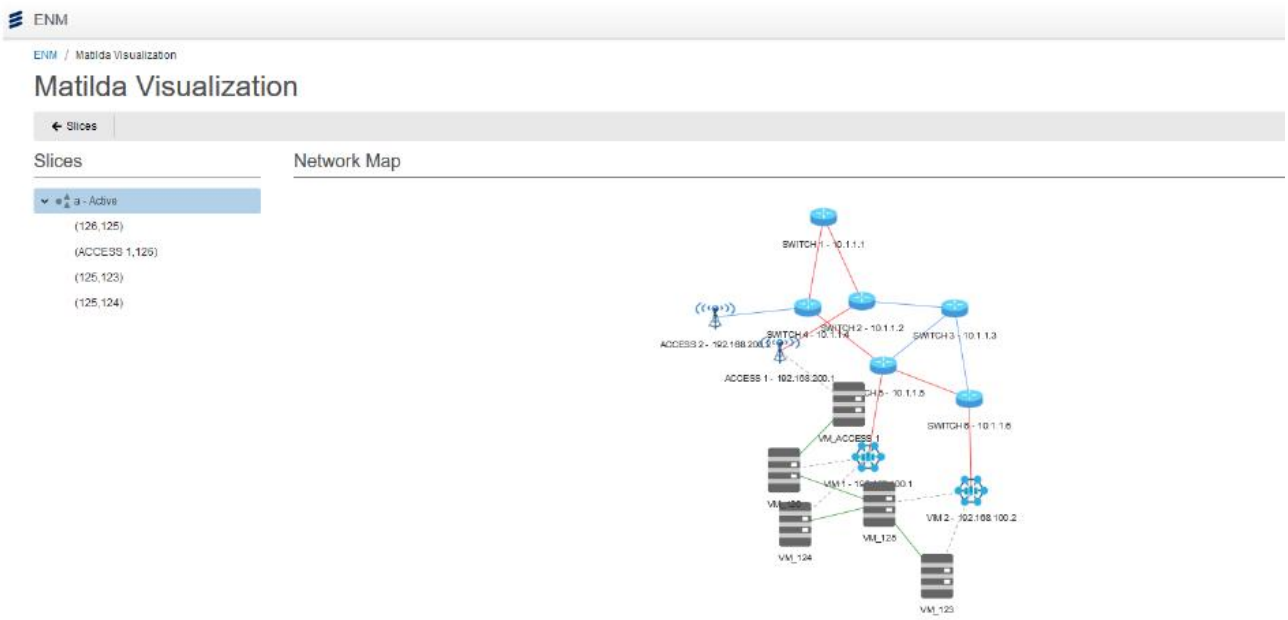


Figure 18: GET Slice visualization.

Figure 17 is dedicated to Computed Slice details visualization. After a Compute Slice request, on the tree view on the left, the set of the possible slices complying the provided constraints are displayed, showing the possible distribution of the Virtual Function on the VIMs. Selecting an element on the tree view, that will be highlighted on the map.

Last but not least, Figure 18 focuses on GET Slice visualization. When a slice is actuated on the network, it is displayed on the tree view, together with the routing details. Selecting the slice on the tree view, its path is highlighted in red on the map. Selecting a hop on the tree view only the corresponding link is shown in red.

The Virtual function are displayed on the map (black rectangles) too. The dotted line between a Virtual Function and a VIM shows where the Virtual Function has been located, while the green line among two Virtual Functions represents the connection between them.

8.3 API description[1]: REST Interface between OSS/BSS and WIM.

Retrieve the configuration parameters related to the maximum number of solutions (computed slices) and maximum allocation number of VMs allowed in the compute-slice response.

URL	**/rest/operations/matilda:config**
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*No parameter*
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Set the configuration parameters related to the maximum number of solutions (computed slices) and maximum allocation number of VMs allowed in the compute-slice response.

URL	**/rest/operations/matilda:config**
Type	PUT
Headers	Accept: application/json Content-type: application/json
Parameters	Parameters value
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Get Topology Nodes: Returns the list of network nodes, included Router, VIM and ACCESS Point.

URL	**/rest/matilda/node**
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*No parameters*
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed

[1] The API Request and Response examples are provided in Annex 2.

	▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Get Topology Links: Returns the list of links.

URL	/rest/matilda/link
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*No Parameters*
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Compute Slice Request: Returns a set of possible optimum slices given as input:

- The set of VMs to be put in place
- The set of VIMs the VMs must be deployed on
- The graph connecting the VMs
- The Delay Constraints on the graph links

The number of returned slices may be configured by the user.

- In case there are not slices fulfilling the required constraints, the return code is 200 OK, but the set of possible slices returned is empty.

POST	**/rest/operations/matilda:compute-slice**
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	The set of VMs to be put in place The set of VIMs the VMs must be deployed on The graph connecting the VMs The Constraints on the graph links, like Delay
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Slice Activate: asks for the activation of a slice among the set of the computed ones. The request contains an additional constraint, the Slice Bandwidth.

URL	**/rest/operations/matilda:activate-slice**
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	o Slice to be Activated o Required Bandwidth
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Get Slice: Returns the slice list, active or computed. The response contains all the slice details, included the route that must be configured on the transport network to put in place the slice.

URL	**/rest/matilda/slice**
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*No Parameters*
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

Slice Deactivate: asks for the deactivation of a slice. The input parameter is the slice identifier.

URL	**/rest/operations/matilda:deactivate-slice**
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	• Slice to be Deactivated
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the PCE is not available.

8.4 The WIM Convergence Layer

As reported in D4.1, the WIM Convergence Layer comprises all the necessary plugins to interface the network nodes (configuration, monitoring, etc.).

Among the most important plugins we can list the following:

- OpenFlow, for handling OpenFlow switches
- XRPC, more suitable than OpenFlow for optical equipment
- SNMP, the traditional Network and Management Protocol
- Netconf, the emerging Network Configuration Protocol with improved network functions

After the OSS/BSS first tentative selection, it will perform the final path computation, adding the bandwidth constraints arising from the chosen selection.

All these tasks mainly leverage on the functional blocks provided by a refined top of the art controller, based on the OpenDaylight framework and using the internal interface RestConf, due to this only some example of the complete set of RestConf API are reported here.

8.4.1 API description: Internal Interface between WIM and WIM convergence layer.

Returns the list of OpenFlow nodes with the configuration details.

URL	/restconf/operational/opendaylight-inventory:nodes
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*No parameter*, Basic Auth to be set in Header

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 500: Internal server Error. In case the service is not available.

9 Network Services and Network Functions

In this section, a description of the main network services currently supported by the OSS NFVCL+NSM module introduced in Section 7, of the main VNFs composing them.

As introduced in Section 3, the MATILDA Telecom Layer Platform is in charge of providing and managing the lifecycle of two categories of network services, namely "*base 4G/5G network services*" and "*vApp/slice-specific network services*."

The first type of services is deployed at the OSS bootstrap in order to provide base 3GPP 4G/5G network connectivity to UEs, and properly configured on-the-fly to host multiple vApp slices. Base 4G/5G NSs are usually shared by multiple vApp slices.

The second type corresponds to auxiliary network services, completely dedicated to a single vApp slice, and providing some requested functionalities to the vApp graph deployed in the distributed multi-site infrastructure. A relevant example of such second type is the L3 overlay NS shown in Figure 4, which connects the vApp components deployed in multiple VIMs.

Both NS types are completely managed by the NFVCL+NSM module in the MATILDA OSS (see Section 7). A NS Blueprint and related VNF Configurators are provided for each possible implementation of a NFV NS, in order to create the needed NS Descriptors, onboard them on OSM, and properly manage all the Day-0/1/2 operations.

It is worth recalling that a same Blueprint can be used to multiple times for creating a number of virtual network instances with the same high-level functionalities (e.g., a complete 4G network, a L3 inter-VIM overlay network, etc.), but with different number of V/PNFs, different deployment options, and different Day-2 configurations.

As discussed in Section 7.2, the NFVCL+NSM operations are strictly related to the VNFs used to build the Blueprint templates: the VNF Descriptors fix how a VNF can be connected in the NS graph, and which primitives are available at the VNF Manager for Day-2 configurations. Therefore, the design of the Blueprint and VNF Configurator plugins at the NFVCL+NSM module should somehow be performed in tight collaboration with the creation and the tuning of VNF packages.

The remainder of this section is organized as follows. Section 0 introduces the V/PNFs that have been prepared by the Consortium and used both for internal experimentation and for the demonstration pilots. Then, Sects. 9.2 and 9.3 will discuss the Network Services that are produced by the NFVCL+NSM module, and that wrealize the base 4G/5G base network services and the ones specific to a vApp slice, respectively.

9.1 List of Available V/PNFs

As shown in Table 2, the MATILDA Consortium released 16 different V/PNF packages, providing both 3GPP and non-3GPP functionalities. Each V/PNF is composed of:

- One or more disk images to be used to instantiate the Virtual Machines composing the VNF (called Virtual Deployment Units – VDUs). Images are pre-loaded at VIMs.
- Optional additional storage volumes for persistency purposes.
- one VNF Descriptor (an example of VNFD is reported in Figure 19). If Day-2 configurations are needed, the VNFD package should contains the Juju charm that will be used as VNFM, and all the supported primitives should be declared in the YAML-formatted VNFD file.

It can be noted that most of the VNFMs of the provided V/PNFD has been specifically designed to support the Day-2 primitives "*set-config*", "*set-start*", and "*set-stop*," as discussed in Section 7.

Table 2: List of V/PNFD provided by the MATILDA Consortium.

Network Function Name	Type	Functions Provided	License(*)	VNFM provider
Amarisoft EPC	VNF	Monolithic 4G EPC including MME, S-GW, P-GW, PCRF, HSS and EIR functionalities. It is based on the **Amari EPC** software package [19] and it is compliant to 3GPP Release 15. It supports UE handovers, VoLTE/IMS services, multiple APNs, dedicated bearers, CIoT.	Proprietary	CNIT
Amarisoft IMS	VNF	Internet Multimedia Subsystem (IMS) Server, meant to provide voice and SMS services to UEs. It is based on the **Amari IMS** software package [19]. It implements P-CSCF with built-in I-CSCF and S-CSCF.	Proprietary	CNIT
Amarisoft eNB	PNF	4G eNodeB based on the **Amari eNB** software package [20] and Software Defined Radio from Amarisoft or National Instruments. It supports multi-cell, NarrowBand-IoT, UE handovers, etc.	Proprietary	CNIT
Router	VNF	Virtual router based on the **VyOS** project [21]. Thanks to its complete and very advanced feature it is used as a Swiss army knife in the provided NSs. It supports routing protocols (OSPF, BGP, RIP), firewall, DNS, DHCP and NAT capabilities, advanced traffic policies and QoS, VPNs and tunnelling, load balancing and high availability protocols.	Open-Source	CNIT
S1 Bypass	VNF	DPDK-based software application realized by CNIT that implements the "bump-in-the-wire" solution to enable edge computing in 4G networks [22]. A full description of this VNF can be found in [2] and in [23]. Its main functionality is to steer packets from/to UEs (encapsulated into GTP tunnels) at the local edge datacenter, towards the vApp components acting as front-end points.	Open-Source	CNIT

Network Function Name	Type	Functions Provided	License(*)	VNFM provider
QMon Server	VNF	VNF the wraps the qMON Server software package (fully described in the D3.2 report) as VNF.	Proprietary	ININ
QMon Client	VNF	VNF the wraps the qMON Client software package (fully described in the D3.2 report) as VNF.	Proprietary	ININ
NextEPC MME	VNF	MME application based on the NextEPC open-source project [24]. It supports 3GPP Release 13. It provides the S1 interfaces to the eNodeBs and S11 interface to S-GW as well as S6a to the HSS. With respect to the Amarisoft EPC it does not support UE handovers, IMS, and CIoT. It does not support multi-MME deployments, but supports multiple S-GWs	Open-Source	AALTO/CNIT (**)
NextEPC HSS	VNF	HSS application based on the NextEPC open-source project [24]. It supports 3GPP Release 13. It implements the S6a interface towards MME using the DIAMETER protocol.	Open-Source	AALTO/CNIT (**)
NextEPC P-GW	VNF	P-GW application based on the NextEPC open-source project [24]. It implements the S5 interface toward the S-GW, SGi interface towards the Internet, and S7 interface to PCRF	Open-Source	AALTO/CNIT (**)
NextEPC S-GW	VNF	S-GW application based on the NextEPC open-source project [24]. It implements the S1u interface towards the eNBs, the S11 interface which is connected to MME, and S5 interface to the P-GW.	Open-Source	AALTO/CNIT (**)
NextEPC PCRF	VNF	PCRF application based on the NextEPC open-source project [24]. It controls the policies and rules for QoS of LTE users and bearers. It provides the Gx interface to P-GW	Open-Source	AALTO/CNIT (**)
Traffic Generator	VNF	Experimental, used only for internal tests. The traffic generator is based on the MoonGen project [25].	Open-Source	CNIT
Virtual eNB and UE Emulator	VNF	Experimental, used only for internal tests. The VNF provides a couple of software emulated eNB and UE. It is based on the LTE-SRS project [26]	Open-Source	CNIT
OpenWRT	VNF	Experimental, used only for internal tests. The VNF provide a simple router CPE with a fully functional web interface. It is based on the OpenWRT project [27].	Open-Source	CNIT
OpenVSwitch	VNF	Experimental, used only for internal tests. The VNF provide an OpenFlow switch based on the OpenVSwitch project [28].	Open-Source	CNIT

(*) The license is referred to the software packages providing the main functionalities of the VNF, and that are included into the VNF disk images.

(**) The VNFD package (including the VNFM) has been provided by AALTO, while the Blueprint and the VNF Configurator plugins by CNIT.

```
vnfd:vnfd-catalog:
    vnfd:
    -   connection-point:
        -   name: vnf-mgt
        -   name: vnf-in
        -   name: vnf-out
        description: A VNF including the VyOS Router with a Day-2 charm
        id: vyos-vnf_vnfd
        mgmt-interface:
            cp: vnf-mgt
        name: vyos-vnf_vnfd
        short-name: vyos
        vdu:
        -   cloud-init-file: cloud-config.txt
            count: 1
            description: VyOS Router
            id: vyos-VM
            image: VyOS
            interface:
            -   external-connection-point-ref: vnf-mgt
                name: eth0
                type: EXTERNAL
                virtual-interface:
                    type: PARAVIRT
            -   external-connection-point-ref: vnf-in
                name: eth1
                type: EXTERNAL
                virtual-interface:
                    type: PARAVIRT
            -   external-connection-point-ref: vnf-out
                name: eth2
                type: EXTERNAL
                virtual-interface:
                    type: PARAVIRT
            name: VyOS-VM
            vm-flavor:
                memory-mb: 1012
                storage-gb: 0
                vcpu-count: 2
        vendor: CNIT-S2N
        version: '1.0'
        vnf-configuration:
            config-primitive:
            -   name: set-config
                parameter:
                -   data-type: STRING
                    name: config-content
            -   name: set-start
                parameter:
                -   data-type: STRING
                    name: start-content
            -   name: set-stop
            initial-config-primitive:
            -   name: config
                parameter:
                -   name: ssh-hostname
                    value: <rw_mgmt_ip>
                -   name: ssh-username
                    value: vyos
                -   name: ssh-password
                    value: vyos
                seq: '1'
            juju:
                charm: vyos
```

Figure 19. Example of provided VNFD for the Router VNF.

9.2 Base 4G/5G Network Services

This section introduces the four base network services currently supported by the MATILDA Telecom Layer Platform. These base services include two alternative implementations of a

complete 4G network (i.e., RAN and EPC), and two services for active network monitoring based on the qMON VNFs.

As far as the two alternative implementations of a complete 4G network are concerned, they mostly differ at the EPC level. The first alternative is based on the monolithic Amari EPC VNF (supporting 3GPP Release 15 specifications), and provides full support to UE handovers, voice and video calls, CIoT, etc. The second one is based on the NextEPC VNFs, and provides a more modular architecture (multiple VNFs, one per each 4G EPC element), but at the cost of a much more limited support for 4G functionalities.

The remainder of this section is dedicated to discuss in detail the aforementioned services.

9.2.1 4G Network: Amarisoft and S1 Bypass Version

This set of NSs corresponds to the one taken as reference for both the deployment example in Figure 4, and for the Blueprint template structure, represented in Figure 10.

As previously sketched, it is composed of multiple NSs: one dedicated to the EPC and deployed in a "core" VIM, and one for each 3GPP Tracking Area Code (TAC) value listed in the *bootstrap_4g* message (see Annex 1). Each NSD has been specifically designed to be instantiated on a single VIM, in order to avoid some well-known bugs in the releases 5 and 6 of the OSM Orchestrator.

As shown in Figure 20, in addition to the virtual network previously introduced, a further network, named "*mngt_net*", has been added to allow the communications among the VNFMs and the VNFIs. The Virtual Machine hosting OSM (and then the VNFMs, too) is assumed to be connected to this network.

It is worth noting that, in order to make the understanding of NSs easier and more intuitively readable, we decided to report diagrams like the one on Figure 20, instead of the textual YAML-formatted NSDs and VNFDs, for the NSs here presented.

An example of the original NSD, as produced by the Blueprint plugin, corresponding to the one used for the Amari EPC NS of Figure 20 (i.e., the sketched blue box on the left) is reported in Figure 21. In the NSD, it can be noted that the parameter "*vim_network_name*" allows to map the logical networks (or more precisely the Virtual Link Descritors (VLD) according to the ETSI NFV terminology) connecting the VNFs in the NSD onto existing networks at VIMs. Moreover, multiple VLDs can be mapped onto the same VIM network. For example, in the current NSD, the S1_north_net and S1_south_net VLDs are usually mapped to the WAN, since the EPC VNF and the S1 bypass VNF can be deployed in different VIMs, and the eNB PNF is meant to reach the VIM through the same WAN. This mapping is exploited by the OSS, which before triggering the Blueprint deployment through NFVCL+NSM module, creates and properly shares the virtual networks to be used as attach points. Further VIM networks, like the WAN or the "public" network for Internet connectivity, are expected to be already available, since they are part of the VIM configuration.

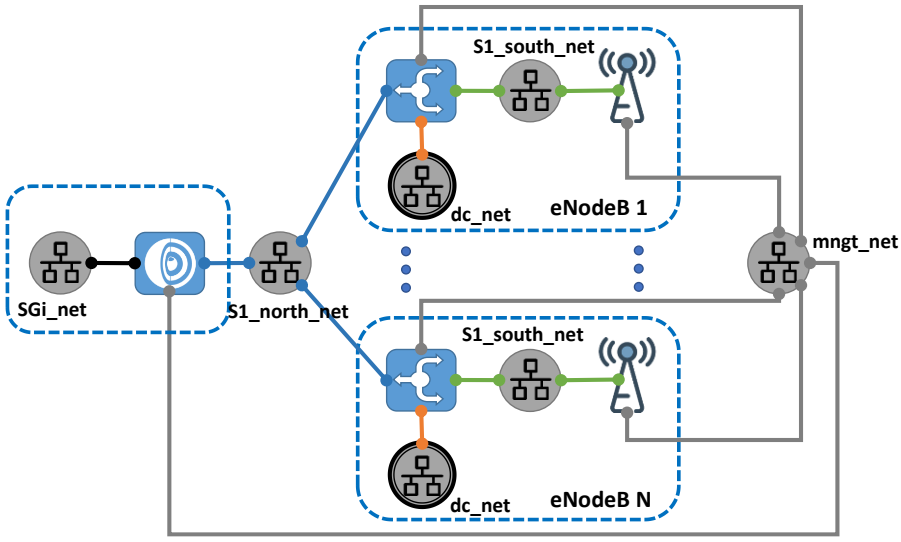


Figure 20: Network Service graphs as generated by the corresponding Blueprint for the 4G network with the Amari EPC and the S1 Bypass VNF.

```
nsd:nsd-catalog:
    nsd:
    -   constituent-vnfd:
        -   member-vnf-index: 1
            vnfd-id-ref: amarisoft-epc-vnf_vnfd
        description: 4G Core
        id: epc_29802
        name: amarisoft-epc-29802_nsd
        short-name: epc-29802_nsd
        vendor: CNIT-S2N
        version: '1.0'
        vld:
        -   id: mngt_net
            name: management
            short-name: management
            type: ELAN
            vim-network-name: mngt-vnf
            vnfd-connection-point-ref:
            -   member-vnf-index-ref: 1
                vnfd-connection-point-ref: vnf-mgt
                vnfd-id-ref: amarisoft-epc-vnf_vnfd
        -   id: sgi_net
            name: sgi
            short-name: sgi
            type: ELAN
            vim-network-name: public
            vnfd-connection-point-ref:
            -   member-vnf-index-ref: 1
                vnfd-connection-point-ref: vnf-sgi
                vnfd-id-ref: amarisoft-epc-vnf_vnfd
        -   id: s1_north_net
            name: s1_epc
            short-name: s1_epc
            type: ELAN
            vim-network-name: os1-intercon
            vnfd-connection-point-ref:
            -   member-vnf-index-ref: 1
                vnfd-connection-point-ref: vnf-s1
                vnfd-id-ref: amarisoft-epc-vnf_vnfd
```

Figure 21: YAML-formatted NSD used to create the OSM NSI containing the Amari EPC VNF.

An outlook on the final Day-2 configuration of the involved V/PNFs is provided in Figure 22, where, for sake of simplicity but without loss of generality, only one instance per VNF type is reported.

All the V/PNFs have a SSH Agent listening on the network interface attached to the *mngt_net*, which act as landing point for the VNFMs. In fact, the Juju charms composing the VNFMs are based on a SSH proxy that allows to send list of shell commands, and to retrieve the consequent output or eventual errors.

In addition, as anticipated in Section 3, it can be noted that on all the VNF network interfaces connected to VLDs that can be mapped to the WAN, end-to-end tunnel instances are set up. These tunnels allow to better discriminate the traffic at the SDN nodes composing the WAN, since the couple of outer source and destination IP addresses of the tunnelled packets will correspond to the IP addresses of the source and destination V/PNFs on the WAN.

In the specific case of the 4G Network implementation under consideration, such tunnels mainly carry S1u (i.e., GTP tunnels) and S1-AP flows (depicted in Figure 22 as coloured cylinders and a solid black lines, respectively).

The creation and the configuration of the inter-VNF tunnels is performed during the Day-2 operations, when the VNFs are already deployed and the IP addresses assigned to them. The tunnel configuration includes the tunnelling protocol to be used, protocol specific identifiers, the tunnel endpoints (i.e., the remote outer IP address on the WAN to connect to), and, if needed, the IP addresses assigned to the tunnel terminations within the VNF (i.e., the inner IP addresses).

In the case of the EPC VNF, the Day-2 configuration includes the creation of the tunnels (one for each connected S1 Bypass VNF), and the dumping of the configuration file of the Amari EPC application with the parameter values requested by the OSS. An excerpt of this file is reported in Figure 23.

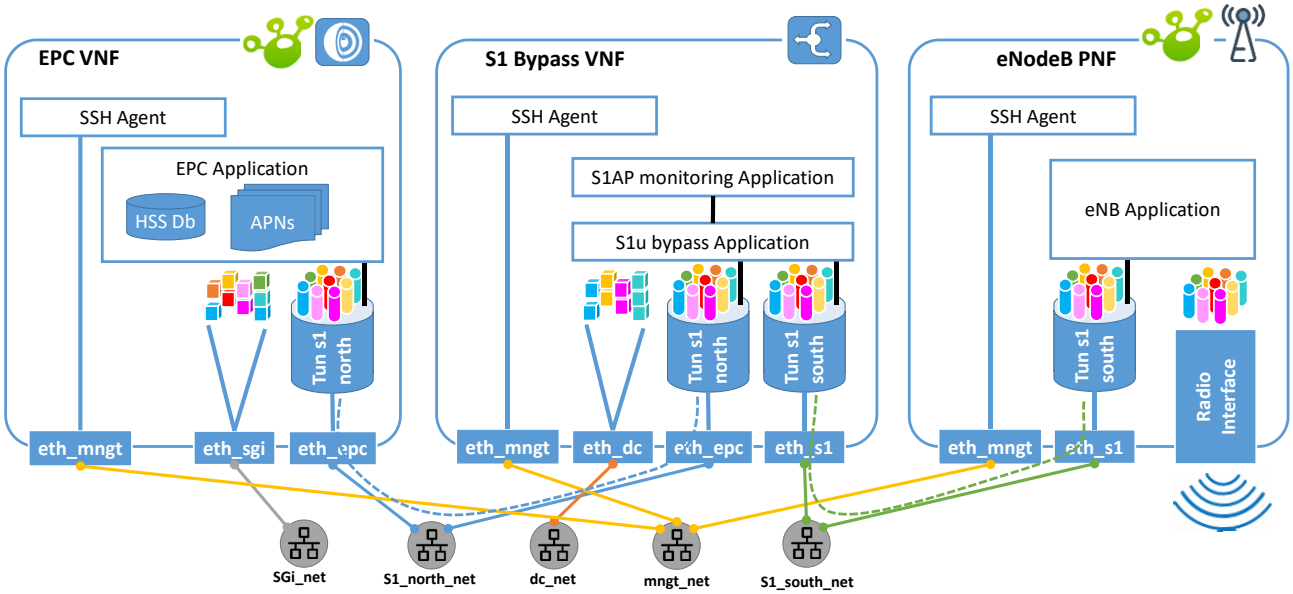


Figure 22: Internal VNF processes and their Day-2 configuration for the virtual 4G network with the Amari EPC and the S1 Bypass VNF.

```
/*  Amari EPC configuration file */
{
  […]
  gtp_addr: "10.100.1.31", /* bind address for GTP-U */
  plmn: "20893",
  mme_group_id: 32769,
  mme_code: 1,

  ims_vops: true, /* IMS supported */
  ims_list: [{ims_addr: "10.0.0.1", bind_addr: "10.0.0.2"}],
  cp_ciot_opt: true, /* Control Plane Cellular IoT EPS optimization support */
  ue_to_ue_forwarding: false, /* UE to UE forwarding support */

  […]
  network_name: "Amarisoft Network",
  network_short_name: "Amarisoft",
  emm_information_enable: true,

  /* Public Data Networks. The first one is the default. */
  pdn_list: [
    {
      pdn_type: "ipv4",
      access_point_name: "default",
      first_ip_addr: "192.168.2.2",
      last_ip_addr: "192.168.2.254",
      ip_addr_shift: 1,
      dns_addr: "192.168.1.1",
      erabs: [
        {
          qci: 9,
          priority_level: 15,
          pre_emption_capability: "shall_not_trigger_pre_emption",
          pre_emption_vulnerability: "not_pre_emptable",
        },
      ],
    },
    {
      pdn_type: "ipv4",
      access_point_name: "slice vApp1",
      first_ip_addr: "192.168.3.2",
      last_ip_addr: "192.168.3.254",
      ip_addr_shift: 1,
      dns_addr: "192.168.1.1",
      erabs: [
        {
          qci: 6,
          priority_level: 8,
          pre_emption_capability: "shall_not_trigger_pre_emption",
          pre_emption_vulnerability: "not_pre_emptable",
        },
      ],
    },
    […]
  ],
  […]
  ue_db: [
    {
        K: "f5b7678eb6add363d7d832965d056c01",
        amf: 36865,
        impi: "01@matilda-project.eu",
        impu: {0:"sip:input05", 1:"tel:33611123460"},
        imsi: "208930100001115",
        opc: "f2b5ad9508398330474dbaf0e6fe7253",
        sim_algo: "milenage",
        sqn: "00000000000000017687"
    },
    […]
  ]
}
```

Figure 23: Excerpt of the Amari EPC configuration file.

As it can be noted from the reported excerpt, the base data for configuring the EPC (e.g., the PLMN identifier, the CIoT, IMS and UE-to-UE forwarding optional capabilities) correspond to the ones sent by the OSS Core to the NFVCL+NSM module through the *boostrap_4g* message (see the Annex 1).

Data like the MME code and group identifier, and the IP address of the network interface where terminating the S1u and S1-AP traffic, are added on-the-fly by the VNF Configurator and Blueprint plugin during Day-2 operations, since they depend on the specific deployment of the service and on how the Blueprint is configured to produce the NSDs.

Further data in the EPC configuration file are retrieved by the NFVCL+NSM module in the OSS Persistency Layer. A "*collection*" in the MongoDB database lists the default settings for each possible PLMN identifier that the OSS can allocate. These settings include the list of the UEs with authentication and accounting parameters to be passed to the HSS, the name of the network to be visualized at the UE, and the values to configure the default Access Point Network (APN).

Further APNs are added as new vApp slices are added to the 4G Network. In such a case all the needed parameters (i.e., QCI and Priority Levels) are specified in the slice intent, and then passed to the NFVCL+NSM module through the "*service*" message with type "*4G*" reported in Annex 1. The VNFM simply adds the new APN to the existing list ("*pdn_list*" with reference to Figure 23) in the configuration file.

A similar Day-2 configuration is performed also for the eNBs. In addition to the creation of a single tunnel towards the S1 Bypass VNF, similarly to the EPC case, most of the Day-2 operations mainly regard the creation of the Amari eNodeB configuration file. Here, only two main type of parameters are needed: the PLMN identifier and the IP address of the MME (i.e., the one used by the EPC VNF to bind the S1 traffic). Usually, eNBs are configured only at the bootstrap of the 4G network, and therefore there are not affected by any slice lifecycle management operations.

At the bootstrap, the S1 Bypass VNF needs the data to set up the couple of tunnels towards the EPC and the eNB, respectively. If no further settings are passed, it behaves as a Layer 2 bridge between the two tunnels. When a vApp slice is allocated in the 4G network, the IP addresses of the front-end components of the vApp, and the list of UEs that can access the vApp are configured by the VNFM. All packets destined to those IPs and sent by enabled users will be decapsulated from the S1u tunnels and steered towards the attach point. The VNF will also allow to encapsulate packet in S1u tunnels from the front-end components to the enabled UEs.

9.2.2 4G Network: Next-EPC and S1 Bypass Version

As previously discussed, this network service mainly differs from the previous ones at the EPC, which is distributed instead on monolithic. As shown in Figure 24, S-GW instances become multiple, and are deployed in the NSDs at the "edge" datacenters close to the S1 Bypass VNF. The MME, HSS, PCRF and P-GW VNF are deployed in a single instance in a separated NSD at the "core" VIM.

Notwithstanding these changes in the NSD topology, Day-2 operations are very similar to the previous ones, because the type of functionalities provided in the two cases are almost overlapping. The data of Figure 23 is obviously split among the VNFM of the various VNFs composing the EPC, in order to properly write the configuration file of each of them.

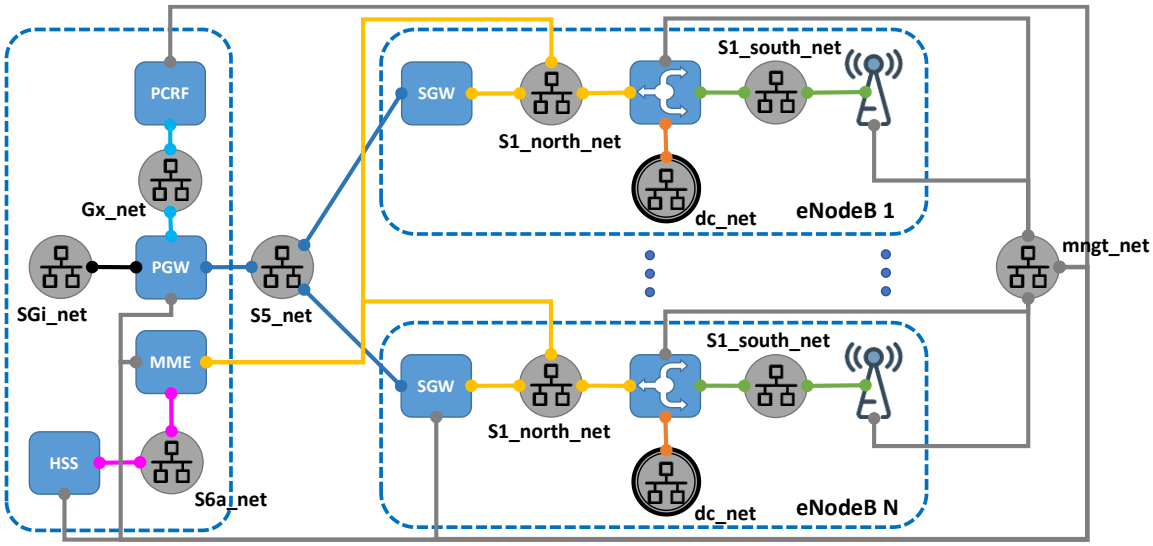


Figure 24: Network Service graphs as generated by the corresponding Blueprint for the 4G network based on the Next-EPC VNFs.

A further minor difference with respect to the previous case is at the VNFM Day-2 primitives. The EPC VNFMs provided to set up this service do not support the "*set-config*" primitive for passing the entire VNF configuration, but they rather implement one primitive for each parameter to be set. In this case, the VNF Configurator simply produces a list of actions, where each action triggers a different primitive.

9.2.3 Active Performance Monitoring: qMON Server and Client

Active network monitoring will be realized with the qMON NFV-based solution consisting of qMON Agent VNF and qMON Server VNF. The first one acts as a measurement client that is measuring network KPIs against the qMON Server VNF. Multiple Agent VNFs can use single Server VNF endpoint.

Measured Network KPIs include packet round trip time (RTT), packet delay variation, download/upload throughput, packet loss, etc. The results are pushed from each qMON Agent VNF to the centralized collector node (qMON Collector) that is part of the provider's infrastructure. From the qMON Collector node the results are available as a Prometheus target to the VAO or to the infrastructure provider itself to monitor end-to-end network performance.

As shown in Figure 25 and Figure 26, qMON Agent and Server VNFs will be each deployed by the NFVCL+NSM as separate NSs, each one stemming from a different Blueprint plugin. The choice of separating the Blueprints and the NSDs for the Client and Server parts of the qMON framework has been mainly driven by the necessity of maximizing the flexibility to deploy and to share these probes within the MATILDA framework. In detail, the Telecom Operator can decide to deploy only one Client/Server VNF, and to make it available to one or more vApp slices as "*Telecom-provided Logical Function*" [2]. Moreover, both Client and Server VNFs can be attached to any of the virtual networks used by the other NSs described in this Section: with reference to Figure 25 and Figure 26, the "*probing_net*" degined in the qMON NSDs can be mapped on the WAN, on the attach points network, or in any of the virtual network used by OSM in the various VIMs. Obviously, each of the various mapped networks can have a different network reachability scope (e.g., from the attach point networks it will not possible to reach 4G EPC interfaces).

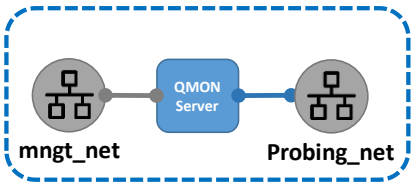


Figure 25: Network Service graph for the qMON Server.

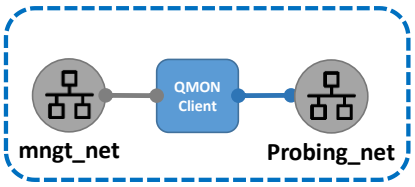


Figure 26: Network Service graph for the qMON Client.

More details on MATILDA monitoring architecture and mechanisms are provided in the Deliverable D3.2 [29].

As far as the VNF version of the qMON Agent and Server, the VNFMs support the following common configuration parameters:

- ***mode*** [*mandatory*]: client/server – define the mode of the qMON VNF;
- ***data_plane_nic*** [*optional*]: true/false – set to true, if separate network interface is used for data plane connectivity;
- ***data_plane_nic_dhcp*** [*optional*]: true/false – set to true, if DHCP client should be enabled on the data plane interface;
- ***data-plane-nic-ipv4-address*** [*optional*]: '172.24.0.12/24' – IPv4 address that is statically assigned to the data plane interface with the charm (in case that no DHCP server is present on the data plane network);
- ***data-plane-nic-ipv4-gw*** [*optional*]: '172.24.0.1'– IPv4 gateway address that is statically configured on the data plane interface with the charm (in case that no DHCP server is present on the data plane network);
- ***data-plane-nic-ipv4-dsn1*** [*optional*]: ''1.1.1.1'– IPv4 address for the primary DNS server that is statically configured on the data plane interface with the charm (in case that no DHCP server is present on the data plane network);
- ***data-plane-nic-ipv4-dsn1*** [*optional*]: '8.8.8.8'– IPv4 address for the secondary DNS server that is statically configured on the data plane interface with the charm (in case that no DHCP server is present on the data plane network).

For the qMON Agent VNF the following configuration parameters are mandatory:

- ***mn*** [*mandatory*]: 'mndev.iinstitute.eu' – the URL of the qMON management server to where the qMON Agent VNFs register and get configuration from;
- ***apikey*** [*mandatory*]: 'xxx'– the key to access API of the qMON management server;
- ***hash*** [*mandatory*]: 'hash' – the unique ID of each qMON Agent VNF (if empty, the charm will auto-generate unique ID).

Both VNF types are using the same charm and three configuration primitives:

- ***set-config*** – performs Day-2 configuration on the VNFs based on the configuration parameters defined in descriptor files;

- ***set-start*** – start the main task of the VNF (on the qMON Agent VNF that triggers start of the measurements);
- ***set-stop*** – stop the main task of the VNF (on the qMON Agent VNF that stop the measurements).

For the Agent VNF, the "set-config" requires two additional parameters:

- ***woid*** [*mandatory*] – defines "Work Order ID" that represents the actual configuration parameters of the measurement that the client needs to perform (e.g. the duration of the measurements);
- ***server-ip*** [*mandatory*] – defines the IP address of the qMON Server VNF providing measurent endpoint.

The Agent VNFs are additionally exposing methods as a configuration primitive for Day-2 configuration that will allow dynamic on-demand measurement configuration changes:

- ***set-work-order*** - allows changing the "Work Order ID" after initial configuration;
- ***set-server-ip*** - allows changing the address of the destination qMON Server VNF.

9.3 vApp/Slice Specific NS

Two types of NSs have been provided for the use within the vApp slices. The first one is for the realization of the L3 overlay network as from the example in Figure 4 (Section 3). The second one is a simple NSD to provide a Firewall functionality as "*Telecom-provided Logical Function*". Both the blueprints are based on the Router VNF, realized with VyOS virtual router instances.

9.3.1 L3 Overlay Network

Figure 27 shows an example of the NSDs produced by the L3 Overlay Blueprint. In detail, such Blueprint will be triggered when a "*service*" message of type "*tunnel*" is received by the NFVCL+NSM module.

The L3 Overlay Blueprint will produce one NSD (and consequently will trigger the instantiation of one NSI) per each VIM where vApp components will reside. Each NSD is a simple wrapper of a single Router VNF instance, which will be provided of 3 VLDs: one "internal" network (the *dc_[x]_net*) to be mapped to the local VIM network used by vApp components (see Figure 4); one external network for interconnectivity among VIMs (i.e., the WAN), and the *mngt_net* for connecting the VNFs with their VNFMs, residing in the OSM virtual machine.

As far as Day-2 configurations are concerned, as shown in Figure 27, Similarly to the 4G network case (see sub-Section 9.2), the Router VNFs will realize the inter-VIM connectivity through point-to-point tunnels. The virtual topology of such tunnels (i.e., their number, their end-points) can be explicitly requested in the "*service*" message. In case that no topology is specified, the L3 Overlay Blueprint builds a full-meshed topology (i.e., a tunnel is set up from any Router VNF to any other Router VNF).

The Layer 3 routing among the *dc_[x]_net* networks will be automatically provided through OSPF. Each router will enable an OSPF daemon, and it will export through it the IP addressing of its locally attached *dc_[x]_net* to the other router instances. The OSPF daemon is configured

to work (i.e., to establish router adjacencies, exchanging Link State Updates, etc.) only on the virtual network interfaces corresponding to the terminations of the aforementioned tunnels (i.e., the tun[x] interfaces with reference to Figure 28).

Additional settings (e.g., firewall and NAT rules) can be set on the Router VNFI if explicitly required by the VAO.

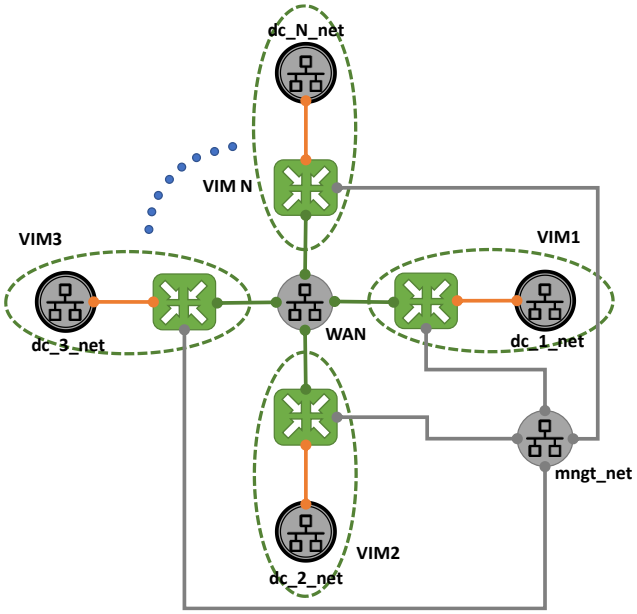


Figure 27: Network Service graphs as generated by the corresponding Blueprint for the virtual L3 overlay network.

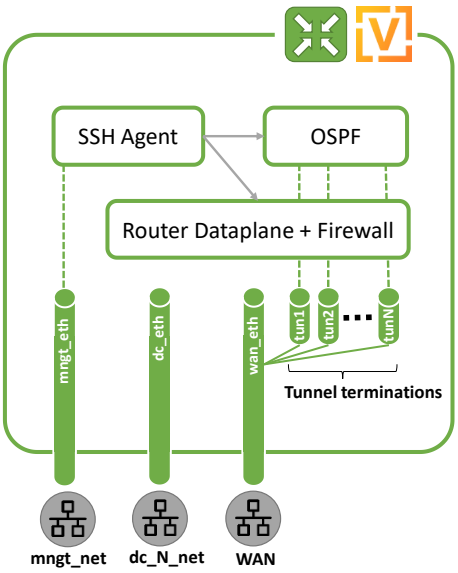


Figure 28: Internal VNF processes and their Day-2 configuration for the virtual L3 overlay network.

Differently from the cases reported in the previous sub-Sections, the Router VNF, based on the VyOS software, will be not configured by the VNFM through a configuration file, but through direct operations on its Command-Line Interface (CLI). This minor difference is mainly due to the VyOS peculiarities, which make CLI operations very straightforward. An example of the CLI

commands sent by the VNFM to the VNF instances for the Day-2 configuration is reported in Figure 29. When all the commands are successfully performed, the VNFM commit the new configuration at the VyOS CLI.

In more detail, from Figure 29 it can be seen that most of the data contained by the configuration is provided by the VNF Configurators and the Blueprint plugins in an on-the-fly fashion, since they are IP addresses acquired during the VNFI deployment (e.g., the outer IP addresses of the tunnels) or depending on the number of Router VNF enabled by the Blueprint.

```
set interfaces ethernet dc_eth address 'dhcp'
set interfaces ethernet dc_eth description 'INSIDE'
set interfaces ethernet wan_eth address '192.168.100.35/24'
set interfaces ethernet wan_eth description 'OUTSIDE'
set interfaces loopback lo address '1.10.1.1/32'
set interfaces tunnel tun0 address '1.1.1.1/30'
set interfaces tunnel tun0 ip ospf network 'point-to-point'
set interfaces tunnel tun0 local-ip '10.100.100.12'
set interfaces tunnel tun0 multicast 'disable'
set interfaces tunnel tun0 remote-ip '10.100.100.26'
[…]
set policy route-map CONNECT rule 10 action 'permit'
set policy route-map CONNECT rule 10 match interface 'eth1'
set protocols ospf area 0 network '1.10.1.0/24'
set protocols ospf parameters router-id '1.10.1.1'
set protocols ospf redistribute connected route-map 'CONNECT'
```

Figure 29: Internal VNF processes and their Day-2 configuration for the virtual L3 overlay network.

9.3.2 Firewall NS

The firewall NSD is fully equivalent to one of the NSDs in the previous section: it hosts only a Router VNF, based on the VyOS software, and has three VLDs: one for the management, one to be mapped as firewall internal network, and one as firewall external network. This type of NSD has been specifically designed to provide firewalling as "*Telecom-provided Logical Function*". Therefore, if the VAO will request the firewall as logical NFV function in the slice intent, this NSD and the related Blueprint will be applied. The internal and external networks can be attached to any of the virtual networks used in the VAO tenant domain (that in this case will be shared through the RBAC policies to the NFV domain) or to the ones used by the NFVO.

The Day-2 configuration is very similar to the previous case, but no tunnels will be set up, and the CLI commands will only enable the desired firewalling rules between the internal and the external interfaces.

10 The MATILDA OSS Monitoring Framework

The OSS Monitoring Framework has been one of the most recent developments to be introduced in the final specification of the platform. The goal of this framework is to improve the cognitiveness of the Telco Platform Layer components by using the collected metrics to drive allocation of resources in a dynamic way. As will be shown in the following sub-sections, metrics are exported from OpenStack (VIMs), OpenDaylight (WIM) and Amarisoft (eNodeBs as well as EPC).

Following the MATILDA practice of relying on existing open-source suites as much as possible, Prometheus [12] has been selected for the realization of the framework. Prometheus is an open-source time series database designed for monitoring that is part of the Cloud Native

Computing Foundation [30]. One of the main benefits of this software resides in its support for multi-dimensional data collection and queuing, which makes it particularly suited for working with microservices.

Moreover, the Prometheus exporters, that are the data collection agents, can be either created from scratch by using the available libraries or it is even possible to use pre-existing exporters. The list of the exporters used for the OSS monitoring framework, along with the exported metrics, is reported in Table 3.

10.1 OpenDaylight Monitoring

The OpenDaylight monitoring has been performed by using an existing open-source Prometheus exporter [31]. Such exporter permits to push to the Prometheus database in the OSS Persistency Layer a number of metrics for each OpenFlow device in the WAN network. Such metrics include: the number of active flows, the number of packets looked-up and matched, the packets and the bytes received and transmitted on each port and for each flow.

In the current version of the prototype, the above metrics are used only for debugging and visualization purposes, and are not used by any optimization/control mechanisms in the OSS, since the control and the status of the WAN network is fully managed in the WIM.

10.2 VIM Monitoring

As in the previous case, the monitoring of the performance metrics related to the VIMs has been performed through an existing open-source Prometheus exporter [32]. This exporter has been slightly modified by the MATILDA consortium in order to support the monitoring of multiple VIMs in the same Prometheus database.

The metrics collected cover a large spectrum of the KPIs provided by the main OpenStack components (i.e., Nova, Neutron, Keystone, Cinder and Glance). For example, the used Exporter mainly support the metrics reported in Table 3.

Table 3: Main KPIs collected by the OpenStack Prometheus Exporter.

Name	Sample Value	Description
OpenStack_neutron_floating_ips	4.0 (float)	Number of Public Floating IP addresses assigned
OpenStack_neutron_networks	25.0 (float)	Number of Neutron virtual networks created
OpenStack_neutron_subnets	4.0 (float)	Number of Neutron IP sub-networks created
OpenStack_neutron_security_groups	10.0 (float)	Number of Neutron security groups created
OpenStack_nova_flavors	4.0 (float)	Number of Virtual Machine Flavors
OpenStack_nova_total_vms	12.0 (float)	Number of Virtual Machine created
OpenStack_nova_running_vms	12.0 (float)	Number of Virtual Machine running
OpenStack_nova_local_storage_used_bytes	100.0 (float)	Amount of storage space used

Name	Sample Value	Description
OpenStack_nova_local_storage_available_bytes	30.0 (float)	Amount of storage space available
OpenStack_nova_memory_used_bytes	40000.0 (float)	Amount of memory space used
OpenStack_nova_memory_available_bytes	40000.0 (float)	Amount of memory space available
OpenStack_nova_vcpus_available	128.0 (float)	Number of virtual CPUs available
OpenStack_nova_vcpus_used	32.0 (float)	Number of virtual CPUs used
OpenStack_cinder_volumes	4.0 (float)	Number of disk volumes created
OpenStack_cinder_snapshots	4.0 (float)	Number of Virtual Machines snapshots
OpenStack_identity_domains	1.0 (float)	Number of Tenant domains created
OpenStack_identity_users	30.0 (float)	Number of user account
OpenStack_identity_projects	33.0 (float)	Number of Tenant Project created

10.3 Amarisoft Radio monitoring

Both Amarisoft EPC and eNB are providing communication via a remote API. The protocol used is WebSocket as defined in RFC 6455. The messages exchanged between the client and MME/ENB server are in strict JSON format. The APIs of both EPC and ENB are providing a plethora of messages (*config_get*, *config_set*, *log_get*, *stats*, *ue_get*, etc. …). The current implementation of the Amarisoft monitoring is using the messages: ***ue_get, enb, stats*** from the EPC API and the ***ue_get, stats*** from the eNB API.

The Amarisoft Radio monitoring is based on Prometheus, an open-source software application used for event monitoring and alerting. Prometheus collects the data in the form of time series. The time series are built through a pull model: the Prometheus server queries a list of data sources (exporters) at a specific polling frequency. Each of the data sources serves the current values of the metrics of that data source. The exporter using the Amarisoft monitoring system is based on **collectd**.

Collectd is a daemon that collects system and application performance metrics periodically and provides mechanisms to store the values in a variety of ways. The collectd daemon has implemented two plugins: the *write_prometheus* and the *python* plugin. The *write_prometheus* plugin is used to listen to queries from the Prometheus server, but also to translate the metrics to a form that can be read from the Prometheus server. The *python* plugin embeds a Python interpreter into the collectd and exposes the API to the python-scripts. This allows to write own plugins in the popular scripting language, which can be then loaded and executed by the daemon without the need to start new process and interpreter every few seconds.

We can name the Collectd alongside with the *python* plugin as the A**marisoft exporter**. In the Amarisoft exporter, there are two configuration files that define the IP addresses of the EPC and eNB that need to be monitored. There are also implemented two python Scripts, one for the EPC and one for the eNB that initiate a websocket connection with the EPCs and eNBs that are listed in the configuration files and retrieve the desired information for each of the component.

Currently, for the EPC, the metrics retrieved are:

- CPU_usage
- IP, PLMN, ENB_ID, active_UEs per connected ENB
- ENB count

The eNB metrics retrieved are:

- CPU_usage
- Total throughput (DL/UL)
- UE count
- Throughput, UL/DL mcs per connected UE
- Pusch/pucch SNR and CQI per connected UE

Every time that the Prometheus queries for data the collectd daemon, the python plugin asks for the desired data all the Amarisoft components API, translate the responses to the appropriate form and exposes the metrics.

For portability and ease of usage reasons, the Amarisoft Exporter, which is consisted from the collectd daemon, the Prometheus and the python plugin, is implemented as a Docker container image.

11 Conclusions

This Deliverable report, titled “Network and Computing Slice”, has described the final release of the Telecom Layer Platform designed for managing the entire lifecycle of 5G-ready Vertical Applications.

In the first release of the platform, outlined in the D4.1 report, the steps and required functionalities for replying to the VAO slice intent message with a materialized slice proposal, and the following slice instantiation, had been completely defined and partly implemented to provide a preliminary, but functional prototype as showcased during the review meeting in November 2018.

The final release of the Telecom Layer Platform described in this document has been developed in full compliance with the original design, and even more functionalities have been added, such as the support for a Telco-layer performance monitoring framework.

The Operations Support System (OSS), which can be considered as one of the most significant outcomes of WP4, has been completed by developing all the functionalities defined in the D4.1. All software services have been designed as state-of-the-art, parallelizable cloud native software to facilitate integration within service-meshes.

The multi-site NFV Orchestrator (NFVO) has been completed to manage the lifecycle of the network services involved in the platform. In more details, two categories of network services, namely “base 4G/5G network services” and “vApp/slice-specific network services” have been handled, providing both 3GPP and non-3GPP functionalities. For each physical and virtual network function (P/VNF), a package has been released for being onboarded in the NFVO, equipped with a manager in the form of a Juju charm for Day-2 configurations.

The Wide-area Infrastructure Manager (WIM), by interacting with the OSS (mainly, the RSO) and the NFVO, provides the management and monitoring of the wide-area communication resources over the distributed 5G infrastructure for creating slices/services to be exposed to vertical in an isolated fashion, as well as the interconnection to traditional and SDN southbound networking devices.

The completed Telecom Layer Platform now provides the full support of Edge Computing functionalities, that is exposing the resources available in the wide-area to the VAOs and providing the attach points between the NFVO and Vertical Applications domains at the edge VIMs. Integration has been achieved with all the components in the MATILDA framework for fully supporting the project demonstrators.

Annex 1: NFVCL API definition

A1.1 NFVCL Service Mapper

Retrieve the Potential Amount of Resources Consumed by a Slice Classified by VIM

URL	**/NFVSM**
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	▪ *Slice bootstrap*: Piece of the slice intent that includes the information to be provisioned.
	Example:
	{
	"bootstrap": {
	"plmn": "29802",
	"type": "4G",
	"config": {
	"ims": true,
	"ciot": true,
	"ue_forwarding": true
	},
	"options": ["XXX", "YYY"]
	"vims": [
	{
	"name": "openstack-1",
	"core": true,
	"wan": {
	"id": "uuid",
	"net_ip": "192.168.1.0/24"
	},
	"sgi": "uuid",
	"tacs": [
	{
	"id": "1",
	"enb_ip": "192.168.1.4",
	"net": "uuid"
	},
	{
	"id": "2",
	"enb_ip": "192.168.1.5",
	"net": "uuid"
	}
	]
	}
	]
	}
	}

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable. In case the NVFO is not reachable.

Response example	<pre>[
	  {
	  "vim_name": "openstack-1",
	  "vm_count": 4,
	  "vcpu-count": 8,
	  "storage-gb": 200,
	  "memory-mb": 5120
	  }
	]</pre>

As a secondary effect of this call, the Persistency Layer database will be updated with information useful for the deployment of the slice: A blueprint will be selected and considered in the resource count. The chosen blueprint is assigned to the slice, along with the resources consumed and persisted in the database:

- Collection name: blueprint_requirements

Given the requirements in the bootstrap message, is it possible to select a blueprint inside the blueprint catalog in the persistency layer that fulfills these requirements.

blueprint ID	options	type
1		4G
2	service1	4G

- Collection name: blueprint_components

This collection represents the composition of the blueprint in terms of VNF desciptors id.

blueprint ID	VNFD_ID (in OSM catalogue)	ORDER
1	44	1
1	431	2
1	5423	3
2	12	1

- Collection name: blueprint_slice_intent

ID of the blueprint selected, linked to the PLMN. This will be the blueprint proposed by the NFV service mapper, regarding the requirements in the slice intent. If validated in the next steps, the NFV service setup will deploy it blueprint.

PLMN	blueprint ID	nsd_id	configurator
21	2	32e324	{vnf-index: 1, configuration_file: "c.conf"

A1.2 NFVCL Service Setup

Interface Between NFVSS and MATILDA OSS (OSS-NFVSS)

- Instantiate the slice that interconnects the application components

URL	/nfvcl/
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	1. *Slice bootstrap*: Piece of the slice intent that includes the information to be provisioned.
	Example:
	<pre>{
	  "bootstrap_4g": {</pre>

<table>
<tr><td></td><td>
<pre>
        "plmn": "1",
        "config": {
            "ims": true,
            "ciot": true,
            "ue_forwarding": true
        },
        "vims": [
            {
                "name": "openstack1",
                "tenant": "matilda",
                "core": true,
                "wan": {
                    "id": "os1-intercon",
                    "net_ip": "192.168.1.0/24"
                },
                "mgt": "mngmnt-vnf",
                "sgi": "public",
                "tacs": [
                    {
                        "id": "1",
                        "enb_mgt_ip": "192.168.21.1",
                        "dc_net": "dc1",
                        "enb_net": "enb"
                    }
                ]
            },
            {
                "name": "openstack",
                "tenant": "matilda",
                "core": false,
                "wan": {
                    "id": "os2-intercon",
                    "net_ip": "192.168.1.0/24"
                },
                "mgt": "mngmnt-vnf",
                "sgi": "public",
                "tacs": [
                    {
                        "id": "2",
                        "enb_mgt_ip": "192.168.1.1",
                        "dc_net": "dc2",
                        "enb_net": "enb"
                    }
                ]
            }
        ]
    }
}
</pre>
</td></tr>
<tr><td>Response code</td><td>

200 – OK: Valid response.
400 - Bad request: The object included in the body is not subscriptable or it is badly formed
401 – Unauthorized: Provided credentials are not correct.
405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call.
503: Service unavailable. In case the NVFO is not reachable.

</td></tr>
<tr><td>Response example</td><td></td></tr>
</table>

This request is the same as the optional one for the Service Mapper part of the NFVCL. This is done to simplify the task for the upper layers, declaring default values and requirements for the slice.

- Delete the instances of all the components in the slice, and triggers the charms to delete the slices in the common components

URL	**/nfvcl/**
Type	DELETE
Headers	Accept: application/json Content-type: application/json
Parameters	2. *plmn_id*: ID of the slice instance. *Example*: `{ "slice_id": "580", }`
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable. In case the NVFO is not reachable.
Request example	`curl - XPOST - H "Content-type: application/json" - d'{ " plmn_id ": "580", }' 'http: // localhost:5000/NFVSS/delete'`

- Service configuration.

URL	**/nfvcl**
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	3. *Service config*: Connection details for the slice plus configuration parameters. *Example*: `{ "service": { "type": "4G", "slice": "580", "config": { "plmn": "29802", "apn": { "qci": 15, "prio": 5, "preem_c": false, "preem_v": false }, "tacs": [ "1", "2", "..." ] } } }`
Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable. In case the NVFO is not reachable.
Response example	

- Specific service instantiation.

URL	/nfvcl
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	4. *Service*: Specific service that needs to be instantiated, configured and attached to a existing slice.
	Example:
	
	{
	"service": {
	"type": "tunnel",
	"slice": "580",
	"config": {
	"vims": [
	{
	"name": "openstack-1",
	"wan": {
	"id": "uuid",
	"net_ip": "192.168.1.0/24"
	},
	"lan": "uuid"
	},
	{
	"name": "openstack-1",
	"wan": {
	"id": "uuid",
	"net_ip": "192.168.1.0/24"
	},
	"lan": "uuid"
	}
	],
	}
	}
	}

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable. In case the NVFO is not reachable.
Response example	

Interface Between NFVSS and OSM(NFVSS-NFVO)

- Create new token

URL	/admin/v1/tokens
Type	POST
Headers	Accept: application/json Content-type: application/json

Parameters	{
	"username": self.username,
	"password": self.password
	}

Response code	▪ 200 – OK: Valid response.

	▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 - Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- Check if VIM is registered in OSM

URL	**/admin/v1/vims/?vim_tenant_name={1}&name={2}**
Type	GET
Headers	Accept: application/json Content-type: application/json Authorization: Bearer {token}
Parameters	None
Response code	▪ 200 - OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 - Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- VIM Registration in OSM

URL	**/admin/v1/vims**
Type	POST
Headers	Accept: application/json Content-type: application/json Authorization: Bearer {token}
Parameters	5. *applicationInstanceID*: ID of the slice instance. *Example*: `{ "schema_version": "1.0", "schema_type": "No idea", "name": vimdata[0]["vimID"] + "#" + vimdata[0]["tenant"], "vim_url": vimdata[0]["endpoint"], "vim_type": "openstack", "vim_user": vimdata[0]["username"], "vim_password": vimdata[0]["password"], "vim_tenant_name": vimdata[0]["tenant"], "description": "VIM created for tenant XXX in dc YYY" }`
Response code	▪ 200 - OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 - Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- Create a new Network Service

URL	**/nslcm/v1/ns_instances/{nsd_id}/instantiate**
Type	POST
Headers	Accept: application/json Content-type: application/json Authorization: Bearer {token}
Parameters	`{ "nsdId": nsd_id, "nsName": name, "nsDescription": description,`

	<pre>"vimAccountId": vim_account_id,
	"config": config
	
	}</pre>

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- Onboard a new Network Service Descriptor

URL	**/nsd/v1/ns_descriptors_content**
Type	POST
Headers	Accept: application/gzip Content-type: application/gzip Content-Filename: pkg_name + '.tar.gz' Authorization: Bearer {token}

Parameters	<pre>{
	"nsdId": nsd_id,
	"nsName": name,
	"nsDescription": description,
	"vimAccountId": vim_account_id
	}</pre>

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- Execute a Day-2 configuration

URL	**/nslcm/v1/ns_instances/{nsd_id}/action**
Type	POST
Headers	Accept: application/yaml Content-type: application/yaml Authorization: Bearer {token}

Parameters	<pre>{
	member_vnf_index: "N",
	primitive: "primitive_name",
	primitive_params: {"primitive_key": "primitive_value"}
	}</pre>

Response code	▪ 200 – OK: Valid response. ▪ 400 - Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 - Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

- Read operation status

URL	**/nslcm/v1/ns_lcm_op_occs/{operation_id}**
Type	GET
Headers	Accept: application/json Content-type: application/json Authorization: Bearer {token}
Parameters	None

Response code	▪ 200 – OK: Valid response. ▪ 400 – Bad request: The object included in the body is not subscriptable or it is badly formed ▪ 401 – Unauthorized: Provided credentials are not correct. ▪ 405 – Method Not Allowed: The method is not allowed for the requested URL. In case POST is not specified in the call. ▪ 503: Service unavailable.

Annex 2: WAN Resource Mapper API Examples

A2.1 Interface Between OSS/BSS and WAN Resource Mapping

URL	**/rest/operations/matilda:config**
Request example	curl -X GET \ http://localhost:8081/rest/operations/matilda:config

Response example	{
	"maxSolutions": 3,
	"maxVms": 5
	}

URL	**/rest/operations/matilda:config**
Request example	curl -X PUT \ http://localhost:8081/rest/operations/matilda:config \ -H 'Content-Type: application/json' \ -d '{ "maxSolutions": 3, "maxVms": 5 }'
Response example	No body

URL	**/rest/matilda/node**
Request example	curl -X GET \ http://localhost:8081/rest/matilda/node

Response example	[
	{
	"nodeType": "PCE_NODETYPE_IP",
	"present": true,
	"userLabel": "SWITCH 4",
	"nodeId": {
	"value": 4
	},
	"routerId": {
	"value": 167837956
	},
	"segId": {
	"value": -1
	},
	"domainId": {
	"value": 0
	},
	"srgb": {
	"present": false,
	"n": 0,
	"listSize": 0,
	"ranges": [],
	"validObject": true,
	"handle": 140390065782752
	},

```
            "nodeSubType": "PCE_NODESUBTYPE_NONE",
            "nodeRelease": "",
            "connectivity": {
                "type": "PCE_SC_PSC",
                "present": true,
                "n": 0,
                "listSize": 0,
                "nstages": 1,
                "resPool": {
                    "input": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065782832
                    },
                    "present": false,
                    "internal": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065782848
                    },
                    "rbs": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065782816
                    },
                    "output": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065782864
                    },
                    "validObject": true,
                    "listSize": 0,
                    "handle": 140390065782816
                },
                "mtxList": [],
                "validObject": true,
                "handle": 140390065782808
            },
            "opState": "PCE_OPERSTATE_ENABLED",
            "admState": "PCE_ADMINSTATE_UNLOCKED",
            "validObject": true,
            "listSize": 0,
            "handle": 140390065782736
        },
        {
            "nodeType": "PCE_NODETYPE_IP",
            "present": true,
            "userLabel": "VIM 2",
            "nodeId": {
                "value": 8
            },
            "routerId": {
                "value": -1062706174
```

```
        },
        "segId": {
            "value": -1
        },
        "domainId": {
            "value": 0
        },
        "srgb": {
            "present": false,
            "n": 0,
            "listSize": 0,
            "ranges": [],
            "validObject": true,
            "handle": 140390065784896
        },
        "nodeSubType": "PCE_NODESUBTYPE_NONE",
        "nodeRelease": "",
        "connectivity": {
            "type": "PCE_SC_PSC",
            "present": true,
            "n": 0,
            "listSize": 0,
            "nstages": 1,
            "resPool": {
                "input": {
                    "present": false,
                    "n": 0,
                    "listSize": 0,
                    "blocks": [],
                    "validObject": true,
                    "handle": 140390065784976
                },
                "present": false,
                "internal": {
                    "present": false,
                    "n": 0,
                    "listSize": 0,
                    "blocks": [],
                    "validObject": true,
                    "handle": 140390065784992
                },
                "rbs": {
                    "present": false,
                    "n": 0,
                    "listSize": 0,
                    "blocks": [],
                    "validObject": true,
                    "handle": 140390065784960
                },
                "output": {
                    "present": false,
                    "n": 0,
                    "listSize": 0,
                    "blocks": [],
                    "validObject": true,
                    "handle": 140390065785008
                },
                "validObject": true,
                "listSize": 0,
                "handle": 140390065784960
            },
            "mtxList": [],
            "validObject": true,
            "handle": 140390065784952
        },
```

```
            "opState": "PCE_OPERSTATE_ENABLED",
            "admState": "PCE_ADMINSTATE_UNLOCKED",
            "validObject": true,
            "listSize": 0,
            "handle": 140390065784880
        },
        {
            "nodeType": "PCE_NODETYPE_IP",
            "present": true,
            "userLabel": "ACCESS 2",
            "nodeId": {
                "value": 10
            },
            "routerId": {
                "value": -1062680574
            },
            "segId": {
                "value": -1
            },
            "domainId": {
                "value": 0
            },
            "srgb": {
                "present": false,
                "n": 0,
                "listSize": 0,
                "ranges": [],
                "validObject": true,
                "handle": 140390065785104
            },
            "nodeSubType": "PCE_NODESUBTYPE_NONE",
            "nodeRelease": "",
            "connectivity": {
                "type": "PCE_SC_PSC",
                "present": true,
                "n": 0,
                "listSize": 0,
                "nstages": 1,
                "resPool": {
                    "input": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065785184
                    },
                    "present": false,
                    "internal": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065785200
                    },
                    "rbs": {
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065785168
                    },
                    "output": {
```

```
                        "present": false,
                        "n": 0,
                        "listSize": 0,
                        "blocks": [],
                        "validObject": true,
                        "handle": 140390065785216
                    },
                    "validObject": true,
                    "listSize": 0,
                    "handle": 140390065785168
                },
                "mtxList": [],
                "validObject": true,
                "handle": 140390065785160
            },
            "opState": "PCE_OPERSTATE_ENABLED",
            "admState": "PCE_ADMINSTATE_UNLOCKED",
            "validObject": true,
            "listSize": 0,
            "handle": 140390065785088
        }
}
```


URL	/rest/matilda/link
Request example	curl -X GET \ http://localhost:8081/rest/matilda/link
Response example	


```
[
    {
        "delay": 20,
        "present": true,
        "userLabel": "",
        "segId": {
            "value": -1
        },
        "opState": "PCE_OPERSTATE_ENABLED",
        "admState": "PCE_ADMINSTATE_UNLOCKED",
        "srlgs": {
            "present": false,
            "n": 0,
            "srlgs": [],
            "listSize": 0,
            "validObject": true,
            "handle": 140390602785768
        },
        "wdm": {
            "present": false,
            "linkId": {
                "value": 0
            },
            "spanLength": 32767,
            "fibreType": "",
            "spanLoss": 32767,
            "dgd": 32767,
            "dispersion": {
                "refWaveLen": 1550,
                "present": false,
                "dispersion": 32767,
                "validObject": true,
                "listSize": 0,
                "handle": 140390602785832
            },
            "ageing": 32767,
            "validObject": true,
            "listSize": 0,
```

```
            "handle": 140390602785784
        },
        "linkId": {
            "value": 1
        },
        "fromNode": {
            "value": 1
        },
        "fromPort": {
            "value": 16843009
        },
        "toNode": {
            "value": 2
        },
        "toPort": {
            "value": 33620225
        },
        "backUniLink": {
            "value": 2
        },
        "scType": "PCE_SC_PSC",
        "maxBdw": 1250000000,
        "maxResBdw": 1250000000,
        "unresBdw": [
            1250000000,
            1250000000,
            1250000000,
            1250000000,
            1250000000,
            1250000000,
            1250000000,
            1250000000
        ],
        "unresBdwSize": 8,
        "minLSPBdw": {
            "bdw": 0,
            "traffic": "ipCh",
            "present": true,
            "validObject": true,
            "listSize": 0,
            "handle": 140390602785600
        },
        "maxLSPBdw": [
            {
                "bdw": 0,
                "traffic": "unassigned",
                "present": false,
                "validObject": true,
                "listSize": 0,
                "handle": 140390602785616
            },
            {
                "bdw": 0,
                "traffic": "unassigned",
                "present": false,
                "validObject": true,
                "listSize": 0,
                "handle": 140390602785632
            },
            {
                "bdw": 0,
                "traffic": "unassigned",
                "present": false,
                "validObject": true,
                "listSize": 0,
```

```
                    "handle": 140390602785648
                },
                {
                    "bdw": 0,
                    "traffic": "unassigned",
                    "present": false,
                    "validObject": true,
                    "listSize": 0,
                    "handle": 140390602785664
                },
                {
            "maxLSPBdwSize": 8,
            "igpMetric": 1,
            "teMetric": 1,
            "affinities": 0,
            "prot": "PCE_PROTTYPE_NONE",
            "validObject": true,
            "listSize": 0,
            "handle": 140390602785456
        }
    }
    ]
```


POST	**/rest/operations/matilda:compute-slice**
Request example	


```
curl -X POST \
  http://localhost:8081/rest/operations/matilda:compute-slice
\
  -H 'Content-Type: application/json' \
  -d '{
       "requestId": 1,
       "clientName": "a",
  "graph": {

    "nodes": [
      {
        "nodeID": "123"
      },
      {
        "nodeID": "124"
      },
      {
        "nodeID": "125"
      },
      {
        "nodeID": "126"
      }
    ],
    "constraints": [
      {
        "constraintID": "591",
        "graphLinkNodeInstanceID": "ACCESS_176",
        "qi": "10",
        "radioServiceType": "1",
        "resourceType": "DELAY_CRITICAL_GBR",
        "allocationRetentionPriorityProfile": 1,
        "minimumGuaranteedBandwidth": 120.0,
        "maximumRequiredBandwidth": 200.0,
        "constraintUnit": "kbps",
        "category": "ACCESS",
        "type": "HARD"
      },
            {
        "constraintID": "620",
        "componentNodeInstanceID": "125",
```

```
        "constraintMetric": "REGION",
        "constraintUnit": "region",
        "constraintValue": "eu-east",
        "category": "COMPONENT_HOSTING",
        "type": "HARD"
      },
      {
        "constraintID": "626",
        "componentNodeInstanceID": "126",
        "constraintMetric": "REGION",
        "constraintUnit": "region",
        "constraintValue": "eu-east",
        "category": "COMPONENT_HOSTING",
        "type": "HARD"
      }
    ],
    "graphLinkNodes": [
      {
        "graphLinkNodeInstanceID": "111",
        "fromComponentNodeInstanceHexID": "uarfSBqJjL",
        "fromComponentNodeInstanceID": "126",
        "toComponentNodeInstanceID": "125",
        "toComponentNodeInstanceHexID": "imvQugGxFN",
        "type": "CORE"
      },
      {
        "graphLinkNodeInstanceID": "110",
        "fromComponentNodeInstanceHexID": "imvQugGxFN",
        "fromComponentNodeInstanceID": "125",
        "toComponentNodeInstanceID": "124",
        "toComponentNodeInstanceHexID": "g35LeBNiru",
        "type": "CORE"
      },
      {
        "graphLinkNodeInstanceID": "ACCESS_176",
        "toComponentNodeInstanceID": "126",
        "toComponentNodeInstanceHexID": "uarfSBqJjL",
        "type": "ACCESS"
      }
    ]
  },
  "vims": [
    {
      "host": "192.168.100.1"
    },
    {
      "host": "192.168.100.2"
    }
  ]
}
'
```


Response example


```
{
    "requestId": 1,
    "clientName": "a",
    "slices": [
        {
            "sliceId": 1,
            "metric": {
                "metric": "DELAY",
                "unit": "ms",
                "value": "62.0"
            },
            "nodes": [
                {
                    "nodeId": "126",
```

```
                    "vims": [
                        {
                            "host": "192.168.100.1"
                        }
                    ]
                },
                {
                    "nodeId": "125",
                    "vims": [
                        {
                            "host": "192.168.100.1"
                        }
                    ]
                },
                {
                    "nodeId": "123",
                    "vims": [
                        {
                            "host": "192.168.100.1"
                        }
                    ]
                },
                {
                    "nodeId": "124",
                    "vims": [
                        {
                            "host": "192.168.100.1"
                        }
                    ]
                }
            ]
        },
        {
            "sliceId": 2,
            "metric": {
                "metric": "DELAY",
                "unit": "ms",
                "value": "82.0"
            },
            "nodes": [
                {
                    "nodeId": "126",
                    "vims": [
                        {
                            "host": "192.168.100.2"
                        }
                    ]
                },
                {
                    "nodeId": "125",
                    "vims": [
                        {
                            "host": "192.168.100.2"
                        }
                    ]
                },
                {
                    "nodeId": "123",
                    "vims": [
                        {
                            "host": "192.168.100.2"
                        }
                    ]
                },
                {
```

```
                        "nodeId": "124",
                        "vims": [
                            {
                                "host": "192.168.100.2"
                            }
                        ]
                    }
                ]
            },
            {
                "sliceId": 3,
                "metric": {
                    "metric": "DELAY",
                    "unit": "ms",
                    "value": "84.0"
                },
                "nodes": [
                    {
                        "nodeId": "126",
                        "vims": [
                            {
                                "host": "192.168.100.1"
                            }
                        ]
                    },
                    {
                        "nodeId": "125",
                        "vims": [
                            {
                                "host": "192.168.100.2"
                            }
                        ]
                    },
                    {
                        "nodeId": "123",
                        "vims": [
                            {
                                "host": "192.168.100.2"
                            }
                        ]
                    },
                    {
                        "nodeId": "124",
                        "vims": [
                            {
                                "host": "192.168.100.2"
                            }
                        ]
                    }
                ]
            }
        ],
        "additionalInfo": []
}
```


URL	/rest/operations/matilda:activate-slice
Request example	curl -X POST \ http://localhost:8081/rest/operations/matilda:activate-slice \ -H 'Content-Type: application/json' \ -d '{ "clientName": "a", "slices": [{

<table>
<tr><td></td><td><pre>
                        "requestId": 1,
                        "sliceId": 1,
                        "reqBdw": {
                                "bandwidthUnit": "kbps",
                                "bandwidthValue": "100.0"
                        },
                        "nodes": [
                                {
                                        "nodeId": "126",
                                        "host": "192.168.100.1"
                                },
                                {
                                        "nodeId": "125",
                                        "host": "192.168.100.2"
                                },
                                {
                                        "nodeId": "124",
                                        "host": "192.168.100.1"
                                },
                                {
                                        "nodeId": "123",
                                        "host": "192.168.100.2"
                                }
                        ]
                }
        ]
}'
</pre></td></tr>
<tr><td>Response example</td><td><pre>
{
    "clientName": "a",
    "results": [
        {
            "requestId": 1,
            "sliceId": 1,
            "result": "SUCCESS"
        }
    ]
}
</pre></td></tr>
</table>

<table>
<tr><td>URL</td><td>/rest/matilda/slice</td></tr>
<tr><td>Request example</td><td><pre>
curl -X GET \
  http://localhost:8081/rest/matilda/slice
</pre></td></tr>
<tr><td>Response example</td><td><pre>
[
    {
        "client": "OSS",
        "id": 1,
        "maxSolutions": 3,
        "maxVms": 5,
        "solutions": [
            {
                "id": 37,
                "totalMetric": {
                    "metric": "DELAY",
                    "unit": "ms",
                    "value": "62.0"
                },
                "links": {
                    "(126,125)": {
                        "head": "126",
                        "tail": "125",
                        "inVim": {
                            "value": 7
                        },
                        "outVim": {
                            "value": 7
</pre></td></tr>
</table>

```
                        },
                        "constraints": [],
                        "metric": 0,
                        "ero": {
                            "present": false,
                            "n": 0,
                            "proutes": [],
                            "listSize": 0,
                            "validObject": true,
                            "handle": 140390468681744
                        },
                        "lspId": null
                    },
                    "(ACCESS 1,126)": {
                        "head": "ACCESS 1",
                        "tail": "126",
                        "inVim": {
                            "value": 9
                        },
                        "outVim": {
                            "value": 7
                        },
                        "constraints": [],
                        "metric": 62,
                        "ero": {
                            "present": true,
                            "n": 1,
                            "proutes": [
                                {
                                    "metrics": {
                                        "present": true,
                                        "n": 1,
                                        "pmetrics": [
                                            {
                                                "type":
"PCE_METRIC_DELAY",
                                                "mvalue": 62,
                                                "present":
true,
                                                "frequired":
"PCE_BOOL_FALSE",
                                                "freturn":
"PCE_BOOL_TRUE",
                                                "fbound":
"PCE_BOOL_FALSE",
                                                "validObject":
true,
                                                "listSize": 0,
                                                "handle":
140390468693544
                                            }
                                        ],
                                        "listSize": 1,
                                        "validObject": true,
                                        "handle":
140390468700136
                                    },
                                    "bdw": {
                                        "bdw": 0,
                                        "traffic": "ipCh",
                                        "present": true,
                                        "validObject": true,
                                        "listSize": 0,
                                        "handle":
140390468700104
```

```
                                        },
                                        "present": true,
                                        "ingressNodeId": 9,
                                        "egressNodeId": 7,
                                        "n": 5,
                                        "ingressPortId": 0,
                                        "egressPortId": 0,
                                        "phops": [
                                            {
                                                "bdw": {
                                                    "bdw": 0,
                                                    "traffic":
"unassigned",
                                                    "present":
false,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696608
                                                },
                                                "present": true,
                                                "nodeId": 2,
                                                "portId":
33620737,
                                                "labelId": {
                                                    "present":
false,
                                                    "labelIdx": -
1,
                                                    "label": 0,
                                                    "otn": false,
                                                    "t":
"PCE_LABEL_TYPE_NONE",
                                                    "wdm": false,
                                                    "lambdaIdx":
999,
                                                    "ip": false,
                                                    "sdh": false,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696636
                                                },
                                                "pkey": {
                                                    "present":
false,
                                                    "flsp":
"PCE_BOOL_FALSE",
                                                    "pceId": {
                                                        "value": 0
                                                    },
                                                    "pathKey": 0,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696644
                                                },
                                                "linkId": 20,
                                                "floose":
"PCE_BOOL_FALSE",
                                                "validObject":
true,
```

```
                                        "listSize": 0,
                                        "handle":
140390468696600
                                    },
                                    {
                                        "bdw": {
                                            "bdw": 0,
                                            "traffic":
"unassigned",
                                            "present":
false,
                                            "validObject":
true,
                                            "listSize": 0,
                                            "handle":
140390468696664
                                        },
                                        "present": true,
                                        "nodeId": 1,
                                        "portId":
16843009,
                                        "labelId": {
                                            "present":
false,
                                            "labelIdx": -
1,
                                            "label": 0,
                                            "otn": false,
                                            "t":
"PCE_LABEL_TYPE_NONE",
                                            "wdm": false,
                                            "lambdaIdx":
999,
                                            "ip": false,
                                            "sdh": false,
                                            "validObject":
true,
                                            "listSize": 0,
                                            "handle":
140390468696692
                                        },
                                        "pkey": {
                                            "present":
false,
                                            "flsp":
"PCE_BOOL_FALSE",
                                            "pceId": {
                                                "value": 0
                                            },
                                            "pathKey": 0,
                                            "validObject":
true,
                                            "listSize": 0,
                                            "handle":
140390468696700
                                        },
                                        "linkId": 2,
                                        "floose":
"PCE_BOOL_FALSE",
                                        "validObject":
true,
                                        "listSize": 0,
                                        "handle":
140390468696656
                                    },
```

```
                                                {
                                                    "bdw": {
                                                        "bdw": 0,
                                                        "traffic":
"unassigned",
                                                        "present":
false,
                                                        "validObject":
true,
                                                        "listSize": 0,
                                                        "handle":
140390468696720
                                                    },
                                                    "present": true,
                                                    "nodeId": 4,
                                                    "portId":
67174657,
                                                    "labelId": {
                                                        "present":
false,
                                                        "labelIdx": -
1,
                                                        "label": 0,
                                                        "otn": false,
                                                        "t":
"PCE_LABEL_TYPE_NONE",
                                                        "wdm": false,
                                                        "lambdaIdx":
999,
                                                        "ip": false,
                                                        "sdh": false,
                                                        "validObject":
true,
                                                        "listSize": 0,
                                                        "handle":
140390468696748
                                                    },
                                                    "pkey": {
                                                        "present":
false,
                                                        "flsp":
"PCE_BOOL_FALSE",
                                                        "pceId": {
                                                            "value": 0
                                                        },
                                                        "pathKey": 0,
                                                        "validObject":
true,
                                                        "listSize": 0,
                                                        "handle":
140390468696756
                                                    },
                                                    "linkId": 5,
                                                    "floose":
"PCE_BOOL_FALSE",
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696712
                                                },
                                                {
                                                    "bdw": {
                                                        "bdw": 0,
```

```
                                                    "traffic":
"unassigned",
                                                    "present":
false,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696776
                                                },
                                                "present": true,
                                                "nodeId": 5,
                                                "portId":
83951873,
                                                "labelId": {
                                                    "present":
false,
                                                    "labelIdx": -
1,
                                                    "label": 0,
                                                    "otn": false,
                                                    "t":
"PCE_LABEL_TYPE_NONE",
                                                    "wdm": false,
                                                    "lambdaIdx":
999,
                                                    "ip": false,
                                                    "sdh": false,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696804
                                                },
                                                "pkey": {
                                                    "present":
false,
                                                    "flsp":
"PCE_BOOL_FALSE",
                                                    "pceId": {
                                                        "value": 0
                                                    },
                                                    "pathKey": 0,
                                                    "validObject":
true,
                                                    "listSize": 0,
                                                    "handle":
140390468696812
                                                },
                                                "linkId": 7,
                                                "floose":
"PCE_BOOL_FALSE",
                                                "validObject":
true,
                                                "listSize": 0,
                                                "handle":
140390468696768
                                            },
                                            {
                                                "bdw": {
                                                    "bdw": 0,
                                                    "traffic":
"unassigned",
                                                    "present":
false,
```

```
                                                "validObject":
true,
                                                "listSize": 0,
                                                "handle":
140390468696832
                                            },
                                            "present": true,
                                            "nodeId": 7,
                                            "portId":
117506305,
                                            "labelId": {
                                                "present":
false,
                                                "labelIdx": -
1,
                                                "label": 0,
                                                "otn": false,
                                                "t":
"PCE_LABEL_TYPE_NONE",
                                                "wdm": false,
                                                "lambdaIdx":
999,
                                                "ip": false,
                                                "sdh": false,
                                                "validObject":
true,
                                                "listSize": 0,
                                                "handle":
140390468696860
                                            },
                                            "pkey": {
                                                "present":
false,
                                                "flsp":
"PCE_BOOL_FALSE",
                                                "pceId": {
                                                    "value": 0
                                                },
                                                "pathKey": 0,
                                                "validObject":
true,
                                                "listSize": 0,
                                                "handle":
140390468696868
                                            },
                                            "linkId": 15,
                                            "floose":
"PCE_BOOL_FALSE",
                                            "validObject":
true,
                                            "listSize": 0,
                                            "handle":
140390468696824
                                        }
                                    ],
                                    "iro": {
                                        "present": false,
                                        "n": 0,
                                        "frequired":
"PCE_BOOL_FALSE",
                                        "pincRes": [],
                                        "listSize": 0,
                                        "validObject": true,
                                        "handle":
140390468700152
```

```
                                        },
                                        "lspa": {
                                            "present": false,
                                            "frequired":
"PCE_BOOL_FALSE",
                                            "flocalProt":
"PCE_BOOL_FALSE",
                                            "includeAny": 0,
                                            "excludeAny": 0,
                                            "includeAll": 0,
                                            "setupPrio": 0,
                                            "holdingPrio": 0,
                                            "validObject": true,
                                            "listSize": 0,
                                            "handle":
140390468700080
                                        },
                                        "ingressLinkId": 0,
                                        "ingressLabel": {
                                            "present": false,
                                            "labelIdx": -1,
                                            "label": 0,
                                            "otn": false,
                                            "t":
"PCE_LABEL_TYPE_NONE",
                                            "wdm": false,
                                            "lambdaIdx": 999,
                                            "ip": false,
                                            "sdh": false,
                                            "validObject": true,
                                            "listSize": 0,
                                            "handle":
140390468700036
                                        },
                                        "egressLinkId": 0,
                                        "egressLabel": {
                                            "present": false,
                                            "labelIdx": -1,
                                            "label": 0,
                                            "otn": false,
                                            "t":
"PCE_LABEL_TYPE_NONE",
                                            "wdm": false,
                                            "lambdaIdx": 999,
                                            "ip": false,
                                            "sdh": false,
                                            "validObject": true,
                                            "listSize": 0,
                                            "handle":
140390468700056
                                        },
                                        "listSize": 5,
                                        "tunnels": {
                                            "present": false,
                                            "n": 0,
                                            "listSize": 0,
                                            "ptunnels": [],
                                            "validObject": true,
                                            "handle":
140390468700064
                                        },
                                        "resBdw": {
                                            "bdw": 0,
                                            "traffic":
"unassigned",
```

```
                                        "present": false,
                                        "validObject": true,
                                        "listSize": 0,
                                        "handle":
140390468700120
                                    },
                                    "srLabels": {
                                        "present": false,
                                        "n": 0,
                                        "phops": [],
                                        "listSize": 0,
                                        "validObject": true,
                                        "handle":
140390468700168
                                    },
                                    "srLabelsRev": {
                                        "present": false,
                                        "n": 0,
                                        "phops": [],
                                        "listSize": 0,
                                        "validObject": true,
                                        "handle":
140390468700184
                                    },
                                    "validObject": true,
                                    "handle": 140390468700024
                                }
                            ],
                            "listSize": 1,
                            "validObject": true,
                            "handle": 140390468697280
                        },
                        "lspId": null
                    },
                    "(125,123)": {
                        "head": "125",
                        "tail": "123",
                        "inVim": {
                            "value": 7
                        },
                        "outVim": {
                            "value": 7
                        },
                        "constraints": [],
                        "metric": 0,
                        "ero": {
                            "present": false,
                            "n": 0,
                            "proutes": [],
                            "listSize": 0,
                            "validObject": true,
                            "handle": 140390468635296
                        },
                        "lspId": null
                    },
                    "(125,124)": {
                        "head": "125",
                        "tail": "124",
                        "inVim": {
                            "value": 7
                        },
                        "outVim": {
                            "value": 7
                        },
                        "constraints": [],
```

```
                            "metric": 0,
                            "ero": {
                                "present": false,
                                "n": 0,
                                "proutes": [],
                                "listSize": 0,
                                "validObject": true,
                                "handle": 140390468693104
                            },
                            "lspId": null
                        }
                    },
                    "nodeAllocations": [
                        {
                            "nodeId": "126",
                            "vims": [
                                {
                                    "host": "192.168.100.1"
                                }
                            ]
                        },
                        {
                            "nodeId": "125",
                            "vims": [
                                {
                                    "host": "192.168.100.1"
                                }
                            ]
                        },
                        {
                            "nodeId": "123",
                            "vims": [
                                {
                                    "host": "192.168.100.1"
                                }
                            ]
                        },
                        {
                            "nodeId": "124",
                            "vims": [
                                {
                                    "host": "192.168.100.1"
                                }
                            ]
                        }
                    ],
                    "maxVms": 4
                }
}
```


URL	**/rest/operations/matilda:deactivate-slice**
Type	POST

Request example	curl -X POST \
	  http://localhost:8081/rest/operations/matilda:deactivate-slice \
	  -H 'Content-Type: application/json' \
	  -d '{
	  "clientName": "a",
	  "ids": [
	  1
	  ]
	}'

Response example	No body

A2.2 Wide-area Routing Setup

URL	/restconf/operational/opendaylight-inventory:nodes

Request example	curl -X GET \
	  http://100.93.92.204:8181/restconf/operational/opendaylight-inventory:nodes \
	  -H 'Accept: */*' \
	  -H 'Authorization: Basic YWRtaW46YWRtaW4='

Response example	{
	  "nodes": {
	  "node": [
	  {
	  "id": "openflow:1",
	  "node-connector": [
	  {
	  "id": "openflow:1:1",
	  "flow-node-inventory:port-number": "1",
	  "flow-node-inventory:name": "s1-eth1",
	  "flow-node-inventory:supported": "",
	  "flow-node-inventory:current-feature": "copper ten-gb-fd",
	  "flow-node-inventory:configuration": "",
	  "flow-node-inventory:peer-features": "",
	  "flow-node-inventory:advertised-features": "",
	  "flow-node-inventory:hardware-address": "F2:37:7F:41:F7:9A",
	  "flow-node-inventory:state": {
	  "link-down": false,
	  "blocked": false,
	  "live": false
	  },
	  "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
	  "receive-errors": 0,
	  "receive-frame-error": 0,
	  "receive-over-run-error": 0,
	  "receive-crc-error": 0,
	  "bytes": {
	  "transmitted": 225974,
	  "received": 1274
	  },
	  "receive-drops": 0,
	  "duration": {},
	  "transmit-errors": 0,
	  "collision-count": 0,
	  "packets": {
	  "transmitted": 1957,
	  "received": 21
	  },
	
	  "transmit-drops": 0
	  }
	  }
	  ],
	  "flow-node-inventory:table": [
	  {
	  "id": 36,

```
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
    ],
                "flow-node-inventory:table": [
                    {
                        "id": 36,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 34,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 0,
                        "flow-hash-id-map": [
                            {
                                "hash": "Match [_inPort=Uri
[_value=openflow:1:2], augmentation=[]]00",
                                "flow-id": "#UF$TABLE*0-49"
                            },
                            {
                                "hash": "Match [_inPort=Uri
[_value=openflow:1:3], augmentation=[]]23098476543630901249",
                                "flow-id": "#UF$TABLE*0-57"
                            },
                            {
                                "hash": "Match [_inPort=Uri
[_value=openflow:1:4], augmentation=[]]00",
                                "flow-id": "#UF$TABLE*0-48"
                            },
                            {
                                "hash": "Match
[_ethernetMatch=EthernetMatch [_ethernetType=EthernetType
[_type=EtherType [_value=35020], augmentation=[]],
augmentation=[]], augmentation=[]]1003098476543630901257",
                                "flow-id": "#UF$TABLE*0-44"
                            },
                            {
                                "hash": "Match
[augmentation=[]]03098476543630901257",
                                "flow-id": "#UF$TABLE*0-50"
                            },
                            {
                                "hash": "Match [_inPort=Uri
[_value=openflow:1:2], _vlanMatch=VlanMatch [_vlanId=VlanId
[_vlanId=VlanId [_value=100], _vlanIdPresent=true,
augmentation=[]], augmentation=[]], augmentation=[]]655350",
                                "flow-id": "#UF$TABLE*0-43"
                            },
                            {
```

```
                                    "hash": "Match [_inPort=Uri
[_value=openflow:1:1], augmentation=[]]00",
                                    "flow-id": "#UF$TABLE*0-47"
                                },
                                {
                                    "hash": "Match [_inPort=Uri
[_value=openflow:1:1], augmentation=[]]655350",
                                    "flow-id": "#UF$TABLE*0-46"
                                },
                                {
                                    "hash": "Match [_inPort=Uri
[_value=openflow:1:1], augmentation=[]]23098476543630901248",
                                    "flow-id": "#UF$TABLE*0-58"
                                },
                                {
                                    "hash": "Match [_inPort=Uri
[_value=openflow:1:3], augmentation=[]]00",
                                    "flow-id": "#UF$TABLE*0-45"
                                },
                                {
                                    "hash": "Match [_inPort=Uri
[_value=openflow:1:4], augmentation=[]]23098476543630901250",
                                    "flow-id": "#UF$TABLE*0-59"
                                }
                            ],
                            "flow": [
                                {
                                    "id": "#UF$TABLE*0-47",
                                    "cookie": 0,
                                    "match": {
                                        "in-port": "openflow:1:1"
                                    },
                                    "hard-timeout": 0,
                                    "priority": 0,
                                    "table_id": 0,
                                    "opendaylight-flow-
statistics:flow-statistics": {
                                        "byte-count": 504,
                                        "duration": {
                                            "second": 8605,
                                            "nanosecond":
191000000
                                        },
                                        "packet-count": 12
                                    },
                                    "idle-timeout": 0
                                },
                                {
                                    "id": "#UF$TABLE*0-58",
                                    "instructions": {
                                        "instruction": [
                                            {
                                                "order": 0,
                                                "apply-actions": {
                                                    "action": [
                                                        {

"order": 0,

"output-action": {

"max-length": 65535,

"output-node-connector": "3"
                                                                }
```

```
                                                                },
                                                                {

"order": 2,

"output-action": {

"max-length": 65535,

"output-node-connector": "CONTROLLER"
                                                                    }
                                                                },
                                                                {

"order": 1,

"output-action": {

"max-length": 65535,

"output-node-connector": "4"
                                                                    }
                                                                }
                                                            ]
                                                        }
                                                    }
                                                ]
                                            },
                                            "cookie": 3098476543630901248,
                                            "match": {
                                                "in-port": "openflow:1:1"
                                            },
                                            "hard-timeout": 0,
                                            "priority": 2,
                                            "table_id": 0,
                                            "opendaylight-flow-
statistics:flow-statistics": {
                                                "byte-count": 0,
                                                "duration": {
                                                    "second": 1633,
                                                    "nanosecond":
166000000
                                                },
                                                "packet-count": 0
                                            },
                                            "idle-timeout": 0
                                        },
                                        {
                                            "id": "#UF$TABLE*0-48",
                                            "cookie": 0,
                                            "match": {
                                                "in-port": "openflow:1:4"
                                            },
                                            "hard-timeout": 0,
                                            "priority": 0,
                                            "table_id": 0,
                                            "opendaylight-flow-
statistics:flow-statistics": {
                                                "byte-count": 229130,
                                                "duration": {
                                                    "second": 8605,
                                                    "nanosecond":
153000000
                                                },
                                                "packet-count": 1713
```

```
                                    },
                                    "idle-timeout": 0
                                },
                                {
                                    "id": "#UF$TABLE*0-59",
                                    "instructions": {
                                        "instruction": [
                                            {
                                                "order": 0,
                                                "apply-actions": {
                                                    "action": [
                                                        {
"order": 0,
"output-action": {
"max-length": 65535,
"output-node-connector": "1"
                                                        }
                                                    },
                                                    {
"order": 1,
"output-action": {
"max-length": 65535,
"output-node-connector": "3"
                                                        }
                                                    }
                                                ]
                                            }
                                        }
                                    ]
                                },
                                "cookie": 3098476543630901250,
                                "match": {
                                    "in-port": "openflow:1:4"
                                },
                                "hard-timeout": 0,
                                "priority": 2,
                                "table_id": 0,
                                "opendaylight-flow-
statistics:flow-statistics": {
                                    "byte-count": 51984,
                                    "duration": {
                                        "second": 1633,
                                        "nanosecond":
165000000
                                    },
                                    "packet-count": 152
                                },
                                "idle-timeout": 0
                            },
                            {
                                "id": "#UF$TABLE*0-49",
                                "cookie": 0,
                                "match": {
                                    "in-port": "openflow:1:2"
                                },
                                "hard-timeout": 0,
                                "priority": 0,
```

```
                                    "table_id": 0,
                                    "opendaylight-flow-
statistics:flow-statistics": {
                                        "byte-count": 252386,
                                        "duration": {
                                            "second": 8605,
                                            "nanosecond":
190000000
                                        },
                                        "packet-count": 1781
                                    },
                                    "idle-timeout": 0
                                },
                                {
                                    "id": "#UF$TABLE*0-43",
                                    "instructions": {
                                        "instruction": [
                                            {
                                                "order": 0,
                                                "apply-actions": {
                                                    "action": [
                                                        {

"order": 0,
                                                            "pop-
vlan-action": {}
                                                        },
                                                        {

"order": 1,

"output-action": {

"max-length": 0,

"output-node-connector": "1"
                                                            }
                                                        }
                                                    ]
                                                }
                                            }
                                        ]
                                    },
                                    "cookie": 0,
                                    "match": {
                                        "in-port": "openflow:1:2",
                                        "vlan-match": {
                                            "vlan-id": {
                                                "vlan-id-present":
true,
                                                "vlan-id": 100
                                            }
                                        }
                                    },
                                    "hard-timeout": 0,
                                    "priority": 65535,
                                    "table_id": 0,
                                    "opendaylight-flow-
statistics:flow-statistics": {
                                        "byte-count": 806,
                                        "duration": {
                                            "second": 8258,
                                            "nanosecond": 78000000
                                        },
                                        "packet-count": 9
```

```
                                    },
                                    "idle-timeout": 0
                                },
                                {
                                    "id": "#UF$TABLE*0-44",
                                    "instructions": {
                                        "instruction": [
                                            {
                                                "order": 0,
                                                "apply-actions": {
                                                    "action": [
                                                        {
"order": 0,

"output-action": {

"max-length": 65535,

"output-node-connector": "CONTROLLER"
                                                        }
                                                    }
                                                ]
                                            }
                                        }
                                    ]
                                },
                                "cookie": 30984765436309012 57,
                                "match": {
                                    "ethernet-match": {
                                        "ethernet-type": {
                                            "type": 35020
                                        }
                                    }
                                },
                                "hard-timeout": 0,
                                "priority": 100,
                                "table_id": 0,
                                "opendaylight-flow-
statistics:flow-statistics": {
                                    "byte-count": 83640,
                                    "duration": {
                                        "second": 1639,
                                        "nanosecond": 90000000
                                    },
                                    "packet-count": 984
                                },
                                "idle-timeout": 0
                            },
                            {
                                "id": "#UF$TABLE*0-45",
                                "cookie": 0,
                                "match": {
                                    "in-port": "openflow:1:3"
                                },
                                "hard-timeout": 0,
                                "priority": 0,
                                "table_id": 0,
                                "opendaylight-flow-
statistics:flow-statistics": {
                                    "byte-count": 224000,
                                    "duration": {
                                        "second": 8605,
                                        "nanosecond":
163000000
```

```
                                    },
                                    "packet-count": 1698
                                },
                                "idle-timeout": 0
                            },
                            {
                                "id": "#UF$TABLE*0-46",
                                "instructions": {
                                    "instruction": [
                                        {
                                            "order": 0,
                                            "apply-actions": {
                                                "action": [
                                                    {

"order": 0,

"output-action": {

"max-length": 0,

"output-node-connector": "2"
                                                    }
                                                }
                                            ]
                                        }
                                    }
                                ]
                            },
                            "cookie": 0,
                            "match": {
                                "in-port": "openflow:1:1"
                            },
                            "hard-timeout": 0,
                            "priority": 65535,
                            "table_id": 0,
                            "opendaylight-flow-
statistics:flow-statistics": {
                                "byte-count": 770,
                                "duration": {
                                    "second": 8258,
                                    "nanosecond": 81000000
                                },
                                "packet-count": 9
                            },
                            "idle-timeout": 0
                        },
                        {
                            "id": "#UF$TABLE*0-57",
                            "instructions": {
                                "instruction": [
                                    {
                                        "order": 0,
                                        "apply-actions": {
                                            "action": [
                                                {

"order": 0,

"output-action": {

"max-length": 65535,

"output-node-connector": "1"
                                                }
```

```
                                                                },
                                                                {

"order": 1,

"output-action": {

"max-length": 65535,

"output-node-connector": "4"
                                                                    }
                                                                }
                                                            ]
                                                        }
                                                    }
                                                ]
                                            },
                                            "cookie": 3098476543630901249,
                                            "match": {
                                                "in-port": "openflow:1:3"
                                            },
                                            "hard-timeout": 0,
                                            "priority": 2,
                                            "table_id": 0,
                                            "opendaylight-flow-
statistics:flow-statistics": {
                                                "byte-count": 27018,
                                                "duration": {
                                                    "second": 1633,
                                                    "nanosecond":
166000000
                                                },
                                                "packet-count": 79
                                            },
                                            "idle-timeout": 0
                                        },
                                        {
                                            "id": "#UF$TABLE*0-50",
                                            "cookie": 3098476543630901257,
                                            "match": {},
                                            "hard-timeout": 0,
                                            "priority": 0,
                                            "table_id": 0,
                                            "opendaylight-flow-
statistics:flow-statistics": {
                                                "byte-count": 0,
                                                "duration": {
                                                    "second": 1639,
                                                    "nanosecond": 87000000
                                                },
                                                "packet-count": 0
                                            },
                                            "idle-timeout": 0
                                        }
                                    ],
                                    "opendaylight-flow-table-
statistics:flow-table-statistics": {
                                        "packets-matched": 196440,
                                        "packets-looked-up": 196440,
                                        "active-flows": 11
                                    }
                                },
                                {
                                    "id": 247,
```

```
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 187,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 217,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },

                    {
                        "id": 80,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 35,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 156,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 66,
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    },
                    {
                        "id": 96,
```

```
                        "opendaylight-flow-table-
statistics:flow-table-statistics": {
                            "packets-matched": 0,
                            "packets-looked-up": 0,
                            "active-flows": 0
                        }
                    }
                ],
                "flow-node-inventory:switch-features": {
                    "max_buffers": 256,
                    "max_tables": 254,
                    "capabilities": [
                        "flow-node-inventory:flow-feature-
capability-flow-stats",
                        "flow-node-inventory:flow-feature-
capability-queue-stats",
                        "flow-node-inventory:flow-feature-
capability-arp-match-ip",
                        "flow-node-inventory:flow-feature-
capability-port-stats",
                        "flow-node-inventory:flow-feature-
capability-table-stats"
                    ]
                },
                "flow-node-inventory:ip-address":
"100.93.92.204",
                "flow-node-inventory:serial-number": "None",
                "flow-node-inventory:manufacturer": "Nicira,
Inc.",
                "flow-node-inventory:hardware": "Open
vSwitch",
                "flow-node-inventory:software": "2.0.2",
                "flow-node-inventory:description": "None"
            },
 ]
}
```

Annex 3: Status of the Integration between MATILDA and SliceNet

A3.1 OpenStack Framework

The OpenStack software is the open tool for creating different private clouds in Orange testbed infrastructure, controlling pools of compute, storage and networking resources. This approach is removing the limitation provided by the dedicated physical infrastructure limitations through virtualization techniques, allowing to deploy a fully virtualized software network, including cloud native and cloud ready capabilities. The proposed architecture and implementation offers the capability of separating the control plane by the user plane function, providing an SBA (Service Based Architecture) Infrastructure.

H2020 Orange Romania testbed is using an OpenStack Ocata version framework v15.0 installed in distributed way across a physical infrastructure with 5 HP servers with different compute capabilities and one HP Shared Storage server. As OS version, all infrastructure is built upon Ubuntu 16.04 LTS. This storage is used for images storage (glance) and as virtual storage pool (cinder).

The entire distribution of OpenStack components are installed as Figure 30 describes:

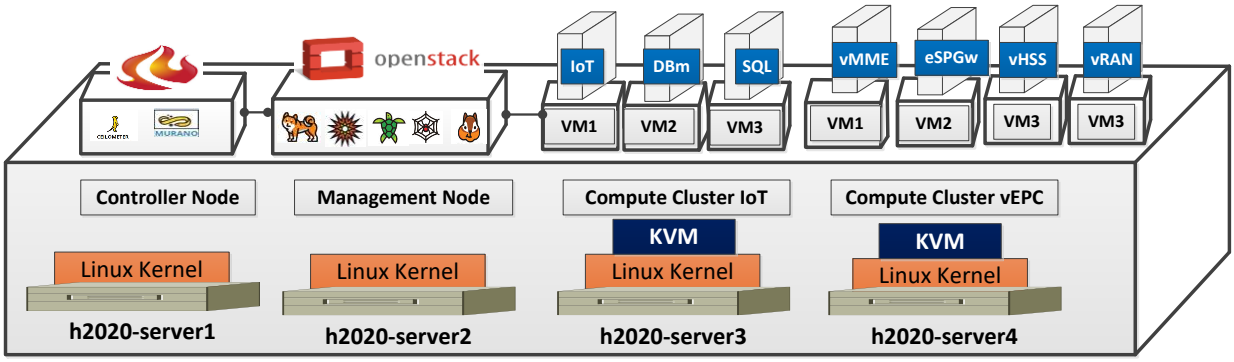


Figure 30: OpenStack hardware and software components.

- Compute node: h2020-server1
- Management node: h2020-server2
- Compute node: h2020-server3-5
- Shared storage node: h2020-storage1

The OpenStack software versions are as described below, providing the capabilities of the platform.

Controller node - h2020-server1		
Software Module	**Software package**	**Version**
Nova	nova-api	15.1.0
	nova-conductor	15.1.0
	nova-consoleauth	15.1.0
	nova-novncproxy	15.1.0
	nova-scheduler	15.1.0
Cinder	cinder-scheduler	10.0.6
Neutron	neutron-dhcp-agent	10.0.5
	neutron-l3-agent	10.0.5
	neutron-linuxbridge-agent	10.0.5
	neutron-linuxbridge-cleanup	10.0.5
	neutron-metadata-agent	10.0.5
	neutron-server	10.0.5
Glance	glance-api	14.0.1
	glance-registry	14.0.1
Horizon		11.0.0

Management node - h2020-server2		
Software Module	**Software package**	**Version**
Ceilometer	ceilometer-agent-notification	8.1.4
	ceilometer-agent-compute	8.1.4
	ceilometer-collector	8.1.4
Murano	murano-api	3.2.0
	murano-engine	3.2.0
Controller node - h2020-server1		
Software Module	**Software package**	**Version**
Nova	nova-compute	15.1.0
	nova-manage	15.1.0
Cinder	cinder-volume	10.0.6
Neutron	neutron-linuxbridge-agent	10.0.5
	neutron-linuxbridge-cleanup	10.0.5
Ceilometer	ceilometer-send-sample	8.1.4
	ceilometer-agent-compute	8.1.4
KVM	QEMU emulator	2.8.0

From network perspective, the testbed compute nodes components are configured with the OVS (OpenVSwitch) server and agents, ovs_version "2.6.3" [33].

The network infrastructure is based on the following design and approach, which allows the operator to deploy in a virtualized environment any components to demonstrate the Smart City use case, using VMs deployments or Docker containers. The composed architecture is described in Figure 31 and highlights the logical components of the testbed that permit not only to virtualize different network components but also provides capabilities and flexibility for end-to-end slicing implementations, including the separation at the logical network implementation level.

The OpenStack implemented in the Orange testbed is based on three IP networks, connecting the virtualized environment with the physical infrastructure, for several use case implementations.

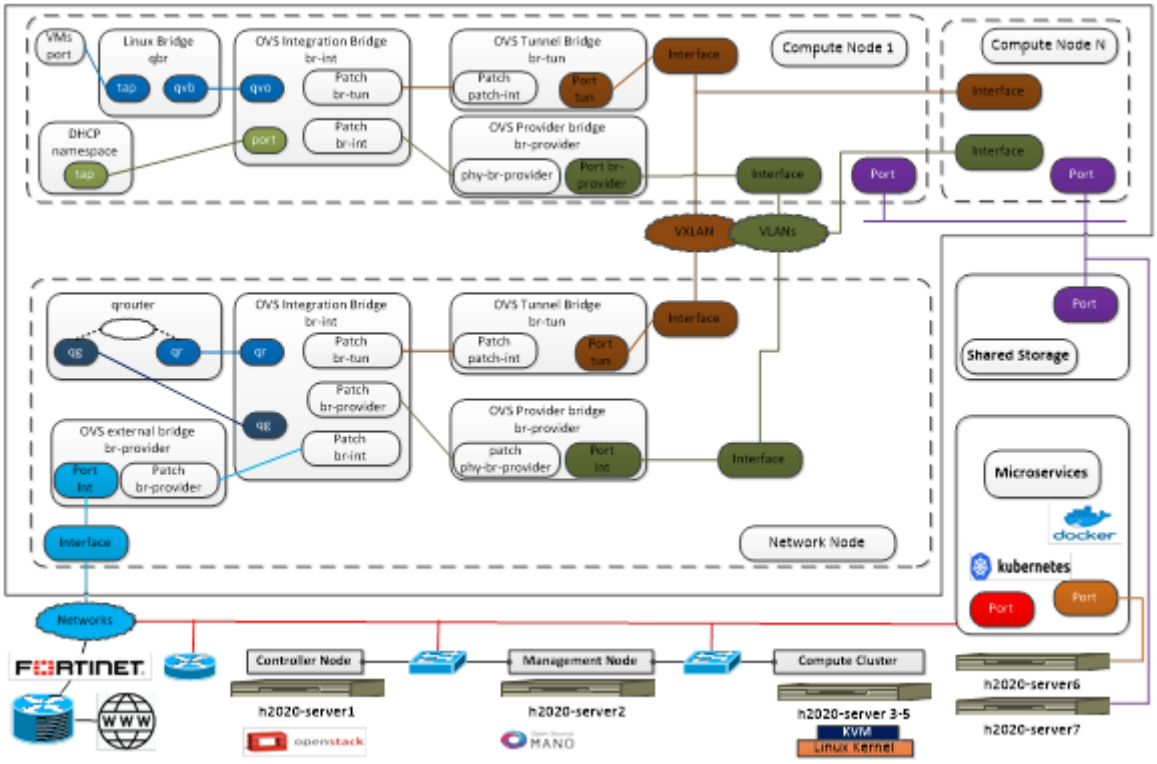


Figure 31: IaaS Network Design.

☐	provider-vlan-600	**subnet-vlan-600-usrp** 192.168.10.0/24
☐	provider-vlan-579	**provider3-management-subnet** 172.25.127.128/25
☐	provider-vlan-578	**provider1-vlan-578-internet-access** 192.168.204.0/26

Figure 32: Testbed OpenStack networks.

☐	selfservice1	**selfservice1-enterprise-network** 192.168.20.0/24
☐	selfservice2	**selfservice2-openairinterface-network** 192.168.0.0/24
☐	selfservice3	**selfservice3-network** 192.168.30.0/24

Figure 33: Logical network separation for slicing.

- 1st network(provider-vlan-600) is the connectivity layer with the physical RRU/BBU
- 2nd network(provider-vlan-579) is the connectivity layer between deployed VMs and has also the management function layer
- 3rd network(provider-vlan-578) is the connectivity layer with outside testbed domain, including internet access.

The deployed infrastructure used for use case demonstration is able to cope huge number of deployed VMs, networks and slicing capabilities, manually or automated deployed.

A3.2 Resource Orchestrator OSM v5

For automated resources and services deployment, it was identified the necessity of integrating a resource orchestrator inside the testbed, ETSI-MANO compliant. Orange testbed is using the Open Source MANO (OSM), the ETSI-hosted open source community delivering a production-quality MANO stack for NFV, capable of consuming openly published information models, available to everyone, suitable for all VNFs, operationally significant and VIM-independent [34].

In ORO testbed, OSMv5 is delivered under a microservices architecture, installed in a physical environment using Docker engine. For the required networks and resources deployments, OSM has been parameterized to work with distributed OpenStack Ocata VIM project previously described – Figure 32.

In the context of the project, OSMv5 is used as an orchestrator helping the deployment and maintenance of slices instantiated according to use cases.

The OSMv5 features and capabilities are, as described in [35], the network slicing for 5G, support for physical and hybrid network functions, multi-domain orchestration, monitoring and policy framework providing closed-loop operations, VNF configuration and deployment, features that applies on ORO test case scenario. The OSMv5 does not currently supports cloud-native network functions, not supporting K8S as a VIM. As described in [36] several options are under investigations for exploration, including the feature of running through OSM VNFs as containers.

The OSMv5 facilitates the NS deployment, as described in Figure 35, deploying of 2 basic VMs, inside own network through OpenStack VIM, based on templates:

- VMs: integration_test_vm-1-cirros_vnfd-VM-1 and integration_test_vm-2-cirros_vnfd-VM-1
- Network: cirros_2vnf_nsd_vld1

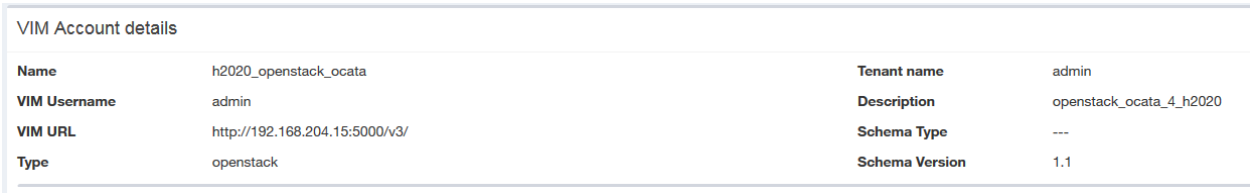


Figure 34: OSMv5 OpenStack VIM configuration.

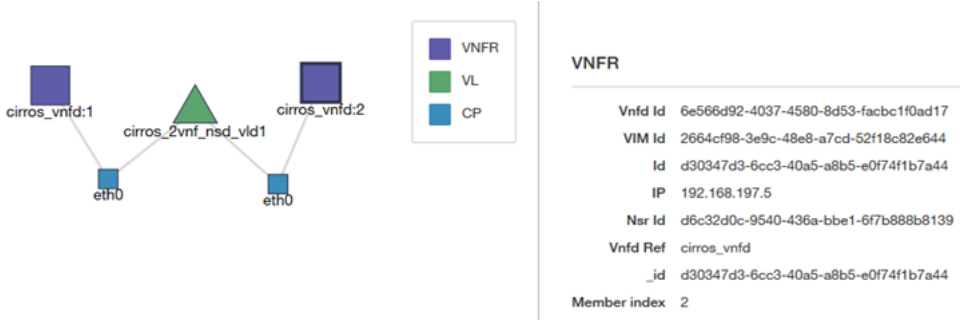


Figure 35: OSMv5 deployment graph example.

The OSMv5, as integrated on Orange testbed, provides the capability to deploy in an automated manner the services required for the Smart City use-case, resource allocation and the infrastructure entry point for automatically slice deployment and services orchestration. The OSMv5 is the resource orchestrator that can be integrated with any slice or service orchestrator, through the NBI, aligned with ETSI NVF, modelling ant automating the full lifecycle of NFVs, network services and slices.

A3.3 Open RAN and Core (OAI) with LTE-M Capabilities

Orange testbed implementation is based on OpenAirInterfaceTM Software Alliance (OSA) [37], a software defined 5G system including cloud RAN, provided by Mosaic5G [38] ecosystem, including several components, as:

- LL-MEC, the low-latency multi-access Edge computing Platform for SDN
- FlexRAN, the flexible programmable platform for SDN
- OpenAirInterface, including several components related to 4G and later 5G system components, as all the 4G components are deployed in a VM approach inside the testbed:
    - OAI-RAN, the software component for monolithic eNodeB with an SDR of the Ettus implementation USRP
    - OAI-Core Network(MME;SP-GW;HSS), the software components for monolithic or distributed Core Network, the platform being able to support both implementation and approaches, deployed through OSM
- The testbed supports the deployment of any RAN and Core Network, communication services and slice concepts isolation, automatically implementations

As the testbed is capable of both virtualization technologies based on OpenStack or Microservices Docker approach, the infrastructure is ready to support fully cloud native, microservices based implementation.

The OAI eNB is capable of basic LTE-M setup, as the IoT Smart City use case is based on different IoT RAN features. The software patch for OAI eNodeB LTE-M capabilities is already deployed on Orange testbed, using a dedicated hardware server.

Orange implemented on the testbed the OAI automated deployment process, using OSMv5, an image of Ubuntu with build-in all in one OAI core network. This version contains OAI applications for MME, P-GW and HSS all configured to work in one VM.

Brief description of the configured VM for the Core Network , having three network interfaces in order to be accessible for management, communicate with RAN infrastructure and have internet access, is described by Figure 36. The automated onbording process and CN VM instance functionality is provided as in the Figure 37 and Figure 38.

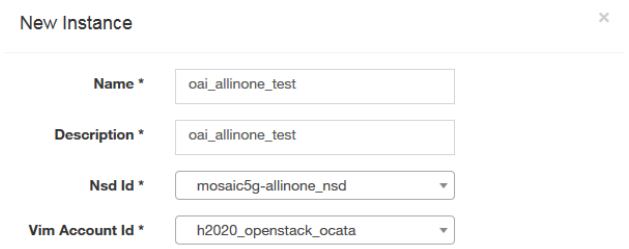


Figure 36: OSMv5 OAI CN parameters.

Figure 37: OAI CN OSM onboarding.

Figure 38: CN all-in-one OpenStack VM.

A3.4 Containerization Capabilities: VM or Bare Metal - [CBR]

ORO testbed infrastructure used for the use case implementation and demonstration, including virtualization capabilities are composed of the next hardware resources that are distributed in different virtualization technologies, as in Table 1.

Table 4: IaaS Virtualization capabilities.

NO	Server	CPU	RAM	Storage	Details
1	h2020-server3	48	128 GB	660 GB	OpenStack compute
2	h2020-server4	48	528 GB	1320 GB	OpenStack compute
3	h2020-server5	40	640 GB	1320 GB	OpenStack compute
4	h2020-server6	48	1024 GB	2640 GB	Docker containers
5	h2020-storage1	N/A	N/A	20 TB	Shared storage

With all above resources, ORO testbed is flexible regarding virtualization capacity and capability to use different technologies. According to this, so far it has been accomplished to deploy:

- VMs with all kinds of flavours and OS images
- Docker containers cluster using VMs instantiated through OpenStack
- Docker containers over bare metal.
- Kubernetes cluster over VMs instantiated through OpenStack.

For use case components deployment in ORO testbed infrastructure, related to the MATILDA requirements and correlated with the Marketplace capabilities and needs, a dedicated K8S infrastructure has been deployed. Using containers for an easy and scalable deployment of

services can offer a lot of benefits. Being also similar to a VM (has an OS, filesystem, CPU, memory, storage, etc), a container has important differences in terms of isolation from resources point of view not using a hypervisor as VM does, but a special engine that is running upon operating system. More notable benefits of containers are scalability, resource utilization, easy deployment and microservices architecture. This benefits make containers flexible and also permit to applications running inside to run on top of all types of infrastructure and operating system.

In this environment, Kubernetes is a special framework built for making infrastructures benefit from all container benefits. K8S provides service discovery, load balancing, storage orchestration, rollbacks, containers healing, secret configuration management and also easy and fast deployment.

In ORO H2020 lab, Kubernetes is instantiated in 2 nodes cluster as showed in **Error! Reference source not found.**.

A3.5 Integration Framework

The framework is composed of the entire application graph components integration, as described in the application graph proposed and provided by Figure 39. The main Smart City

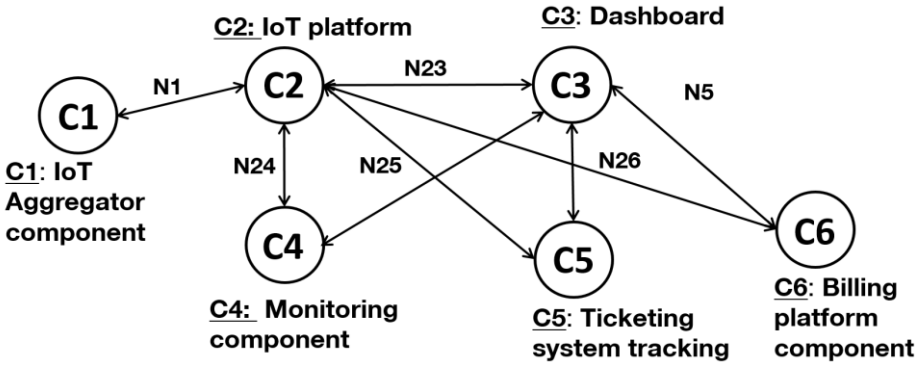


Figure 39: Application Graph for Smart City use case.

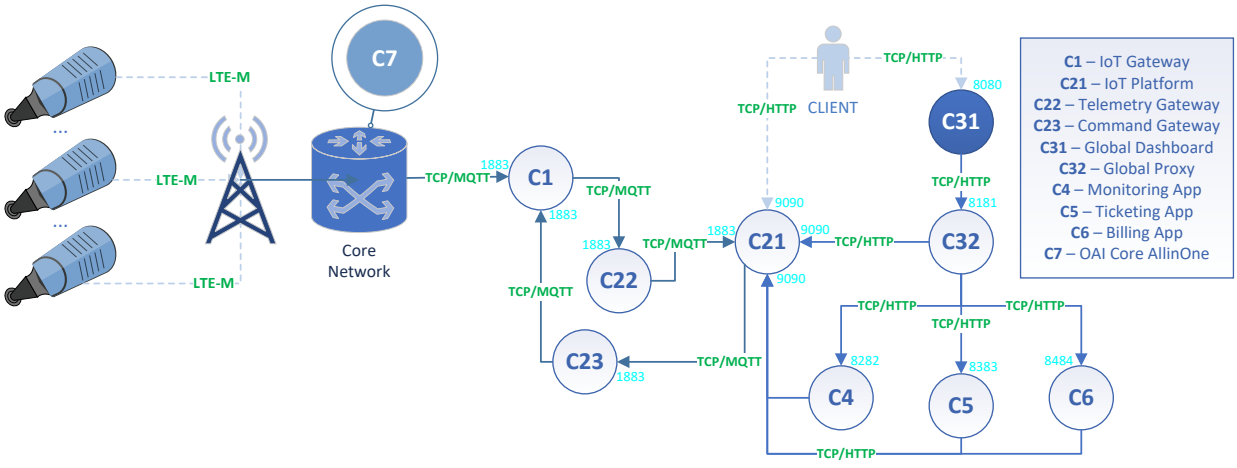


Figure 40: Smart City use case integration framework.

application components, the 5G ready application graph for the use case components are as described next:

- C1 (Component No. 1) - IoT Aggregator
- C2 (Component No. 2) - IoT Platform
- C3 (Component No. 3) - Dashboard Application
- C4 (Component No. 4) - Monitoring System
- C5 (Component No. 5) - Ticketing System
- C6 (Component No. 6) - Billing System.

The entire graph has been built around component C2 – IoT platform, because this application is capable to collect, store and process all kind of data: telemetry data from devices, billing or monitoring data and also different sets of commands and user interaction. All components communicate only with IoT platform to share data (e.g. telemetry, alarms, etc) and are also exposed to user through different endpoints (e.g. graphical user interfaces), as will be described in the following paragraphs.

C1 (Component No. 1): known as IoT Aggregator component, is the part of infrastructure connected to the end devices (smart lighting lamps). The IoT aggregator stands a proxy between the smart lightning lamps and C2 (IoT platform). Therefore, it aggregates all the messages from the smart lighting lamps and passes them forward towards C2. The communication between C1 and C2 uses MQTT protocol enabled by python scripting.

C2 (Component No. 2): is the main component of the use case and it is used to store and process the data (telemetry, alarms, monitoring, etc) and also to control devices (send different types of commands: on, off, dim). The IoT platform is an open-source version (community edition) of Thingsboard.IO platform that is connected to a PostreSQL Database or Apache Cassandra NoSQL Database. The platform has available several connectivity protocols and APIs, such as MQTT/MQTTs, CoAP, HTTP/HTTPS, SigFox , LoRa (only with specific network servers). In this use case all integrations have been made using MQTT and HTTP protocols.

C3 (Component No. 3): the component number 3, dashboard administration is a centralized front end application with multiple functions that can offer to user/admin an overall view and general control of the entire system.

C4 (Component No. 4): Monitoring system to check the status (including alarms) of the infrastructure and services.

C5 (Component No. 5): ticketing system for tracking and resolving possible issues. Can offer information regarding the alarms/incidents in the smart lighting system (infrastructure or services), root cause and tenants involved.

C6 (Component No. 6): is the billing application that is connected to C2 (IoT platform) and collects all information about tenants and users (e.g. number of devices connected, number of telemetry messages, type of devices, etc) of the Smart Lighting System. Based on a commercial offer regarding number of devices connected and other additional services, a bill for every tenant of the platform shall be generated.

Orange proposed architecture and integration is evolving, as described in Figure 40, to a system implementation that will provide a three main layer application architecture, as (1) the

Development Environment and Marketplace, (2) Vertical Application Orchestrator (VAO), (3) the Network virtualized infrastructure layer. The integration between layers, developed in MATILDA and in SliceNet is performed through a One-Stop API (OSA) solution, addressed in SliceNet also as a joint effort between the projects.

The dedicated interface One-Stop API (OSA) is to permit control data transfer (requirements and answers) between Vertical Application Orchestrator and Telco Infrastructure. The One-Stop API (OSA) aggregates the NSI/NSSI/NF offerings as a pool of selectable features that can be identified into a service creation request, as the design of OSA is still ongoing. In MATILDA project this interface is called the Telco Provider Northbound API.

References

[1] MATILDA Deliverable D1.1 – MATILDA Framework and Reference Architecture.

[2] MATILDA Deliverable D4.1 - Network and Computing Slice – First Release.

[3] R. Bolla, R. Bruschi, F. Davoli, P. Gouvas, A. Zafeiropoulos, "Mobile Edge Vertical Computing over 5G Network Sliced Infrastructures: an Insight into Integration Approaches", IEEE Communications Magazine, IEEE, 2019, Pisacaway, MA, USA.

[4] MATILDA Deliverable D1.4 - Network-aware Application Graph Metamodel.

[5] R. El Hattachi, J. Erfanian, "NGMN 5G White Paper," Feb. 2015, Online (Last Access on 30 Oct. 2018): https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf.

[6] The 3GPP Association, "Study on Management and Orchestration of Network Slicing for Next Generation Network," Technical Report (TR) 28.801, version 15.0.0, Sept. 2017.

[7] ETSI "Network Functions Virtualisation (NFV), Management and Orchestration," ETSI GS NFV-MAN 001 V1.1.1, URL: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf.

[8] Open Source MANO, https://osm.etsi.org/wikipub/index.php/Main_Page.

[9] The Ericsson Network Manager, https://www.ericsson.com/en/portfolio/digital-services/automated-network-operations/network-management/network-manager.

[10] OpenDaylight, http://www.opendaylight.org/software.

[11] MongoDB, www.mongodb.com/.

[12] Prometheus, https://prometheus.io/

[13] SageMath, http://www.sagemath.org/.

[14] Juju, https://jujucharms.com/.

[15] "Network Functions Virtualisation (NFV); Management and Orchestration; Network Service Templates Specification", https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/014/02.01.01_60/gs_NFV-IFA014v020101p.pdf

[16] "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Performance Measurements Specification", https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/027/02.04.01_60/gs_nfv-ifa027v020401p.pdf

[17] "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification", https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/001/02.05.01_60/gs_NFV-SOL001v020501p.pdf.

[18] R. Bruschi, F. Davoli, P. Lago, J. F. Pajo, "A Multi-Clustering Approach to Scale Distributed Tenant Networks for Mobile Edge Computing", IEEE Journal on Selected Areas in Communications, IEEE, 2019, USA.

[19] Amarisoft EPC Specification, https://www.amarisoft.com/technology/epc/

[20] Amarisoft eNodeB Specification, https://www.amarisoft.com/technology/enodeb/

[21] VyOS - an Open Source Linux-based Network OS, https://vyos.io/.

[22] "MEC Deployments in 4G and Evolution Towards 5G", ETSI White Paper, February 2018.

[23] R. Bruschi, G. Lamanna, C. Lombardo, S. Mangialardi, "Extending the TRIANGLE testbed towards Mobile Edge Computing", ISWCS 2018, Lisbon, Portugal, 2018.

[24] NextEPC, https://nextepc.org/.

[25] The MoonGen Project, https://github.com/emmericp/MoonGen/projects

[26] LTE SRS, https://github.com/srsLTE/srsLTE

[27] OpenWRT, http://openwrt.org

[28] OpenvSwitch, http://www.openvswitch.org.

[29] MATILDA Deliverable D3.2 – Intelligent Orchestration Mechanisms.

[30] The Cloud-Native Computing Foundation, https://www.cncf.io/

[31] ODL Prometheus Exporter, https://github.com/icclab/opendaylight-prometheus-exporter

[32] OS Prometheus Exporter, https://github.com/OpenStack-exporter/OpenStack-exporter

[33] OVS Release, https://github.com/openvswitch/openvswitch.github.io/blob/master/releases/NEWS-2.6.3

[34] OSM Rel. 5, https://osm.etsi.org/wikipub/index.php/OSM_Release_FIVE

[35] OSM White Paper, https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf

[36] 5G-PPP Software Network Working Group, Cloud-Native and Verticals services, 5G-PPP projects analysis, July 2019

[37] OpenAirInterface, https://www.openairinterface.org

[38] Mosaic5G, http://mosaic-5g.io