Deliverable 3.3

Design and Prototyping of 5G-Connected Virtualized Enterprise Infrastructure and Services

Editor:	Elena-Mădălina Oproiu, ORANGE ROMANIA
Deliverable nature:	Prototype (P)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	30-04-2018
Actual delivery date:	15-06-2018
Suggested readers:	Telecommunication Operators, Service Providers, Infrastructure providers; Communication service providers, Digital service providers; Network operators; Vertical industries
Version:	1.0
Total number of pages:	121
Keywords:	Enterprise, OpenStack, Smart City, Cloud, Virtualization, 5G

Abstract

This document presents all the activities related to the design and prototyping of a 5G-Connected Virtualized Enterprise Infrastructure and Services dedicated for Smart City Use Case. The design proposed ensures the extension of the slicing concepts at the enterprise level, as an end-to-end (E2E) slicing-friendly infrastructure. The demanded business model from the vertical perspective is built on an open source architectural proposal, including for the segment described in details the blocks involved: Data Plane (composed by Enterprise Infrastructure and Enterprise Private Cloud), Enterprise Applications and Enterprise Services, Control Plane, Management Plane and Cross Plane Orchestration under the umbrella of One Stop Application Programming Interface (OSA). The proposed architecture is intended to be opened and flexible, capable to be adapted to support different use cases over the same physical infrastructure.

Disclaimer

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number H2020-ICT-2014-2/671672.

Impressum

[Full project title] End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks

[Short project title] SLICENET

[Number and title of work-package] WP 3: 5G Integrated Multi-Domain Slicing-Friendly Infrastructure

[Number and title of task] T3.3: 5G-Connected Virtualized Enterprise Infrastructure and Services

[Document title] Design and Prototyping of 5G-Connected Virtualized Enterprise Infrastructure and Services

[Editor] Oproiu Elena-Mădălina, Orange Romania

[Work-package leader: Name, company] Oproiu Elena-Mădălina, Orange Romania

[Estimation of PM spent on the Deliverable]

Executive summary

5G-Connected Virtualised Enterprise Infrastructure and Services described in this deliverable is using open-source software for creating clouding capable infrastructures, OpenStack based, as a popular enterprise software tool for managing and controlling resources in the Datacentre. The 5G slice-friendly cross-domain infrastructure layer, physical and virtual is comprising the enterprise segment and addresses all the activities and transformation needed so that the enterprise domain to be adapted and ready for the novel E2E 5G slicing-friendly infrastructure.

The SliceNet enterprise architecture provided by the document is aiming to transform the classic enterprise concepts by requiring new technical and business approaches, including programming and automation into the Smart City vertical use case. It also provides an advantageous infrastructure operational model, flexibility, including fast deployment, vertical slice implementation, services and use case resources assurance and the prototyping design for the Enterprise 5G segment. The model proposed may be easily adapted to any open tool framework (European Telecommunication Standards Institute Management and Orchestration (ETSI MANO) approach) and defines specific metrics and Key Performance Indicators (KPIs) for the vertical enterprise's use case developed.

There are wide range of research related to Software Defined Networks (SDN) and Network Function Virtualization (NFV) implementations for MEC, RAN and Core segment for 5G infrastructure deployments, the current deliverable brings and innovative approach that extend the 5G concepts to the enterprise level, being highlighted by the following achievements:

- A 5G slice-friendly Enterprise infrastructure, ready to cover a real 5G Smart City use case, that can easily be adapted to any other vertical use-case;
- Infrastructure prototype ETSI MANO compliant, the concepts presented are available to be integrated in any enterprise scenario implementation;
- The prototype presented is fully functional, detailed presented for implementation, open to any further deployments in the enterprise segment, allowing further programmability for resource allocation, optimization and cognition;
- Ready to integrate development and innovation concepts related to the novel management and CP provided into the SliceNet system definition including cognition capabilities, QoE sensors and actuators, as vertical-informed implementations;
- The virtualized Enterprise infrastructure slice ready.

List of authors

Company	Author	Contribution
ECOM	Navid Nikaein	The transition of the existing enterprise infrastructure and services to a 5G-connected, virtualised deployment; SliceNet vision about 5G Connected Services, about Control Plane, Management Plane and Cross-Plane Orchestration for Enterprise; OSA for Enterprise and its integration in SliceNet E2E architecture.
ECOM	Xenofon Vasilakos	
ECOM	Tien Thinh Nguyen	
ORO	Elena-Madalina Oproiu	Abstract, Executive Summary, Introduction; Enterprise Data Plane: Enterprise Infrastructure with Smart City IoT segment, Private Cloud for Smart City; Enterprise Services: Smart City Applications and Services; Control Plane for Enterprise: Smart City Control Plane (Virtual Infrastructure Manager (VIM) and SDN controller); Management Plane for Enterprise: Smart City Management Plane (Monitoring, Life-Cycle Management (LCM), Configuration Management); Cross-Plane Orchestration for Enterprise: Smart City Cross-Plane Orchestration and Enterprise Integration in SliceNet E2E architecture; OSA for Smart City and its integration in SliceNet E2E architecture; Prototyped friendly network segment for Smart City; Conclusions; References; Annexes; Abbreviations;
ORO	Marius Iordache	
ORO	Mihai Idu	
ORO	Catalin Costea	
ORO	Catalin Brezeanu	
ORO	Alexandru Oprea	
ORO	Patachia Cristian	

Table of Contents

List of figures

List of tables

Abbreviations

3GPP	3rd Generation Partnership Project
3GPP TS	3rd Generation Partnership Project Technical Specifications
5G	Fifth Generation (mobile/cellular networks)
5G PPP	5G Infrastructure Public Private Partnership
5GEX	5G Exchange
API	Application Programming Interface
APP	Application
AWS	Amazon Web Services
BSS	Business Support System
CapEX	Capital Expenditure
CLI	Command Line Interface
CN	Core Network
CoAP	Constrained Application Protocol
COTS	Commercial Off-The-Shelf
CP	Control Plane
CPU	Central Processing Unit
CQI	Channel Quality Indicator
DB	Database
DC	Datacenter
DHCP	Dynamic Host Configuration Protocol
DSC	Digital Selective Calling
DSP	Digital Service Provider
E2E	End-to-End
ECOMP	Enhanced Control, Orchestration, Management and Policy
EM	Element Management
eNodeB	Evolved NodeB
ETSI	European Telecommunications Standards Institute
ETSI MANO	European Telecommunication Standards Institute Management and Orchestration
ETSI MEC	European Telecommunications Standards Institute Multi-access Edge Computing
ETSI NFV	European Telecommunications Standards Institute Network Function Virtualization
FCAPS	Fault, Management, Configuration, Accounting, Performance, Security
FPGA	Field Programmable Gate Arrays
FTP	File Transfer Protocol
GB	GigaByte
Gbps	Gigabits per second
GBR	Guaranteed Bit Rate
GHz	GigaHertz
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HDD	Hard Disk Drive

HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IaaS	Infrastructure as a Service
IMS	Internet Protocol Multimedia Subsystem
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
L3	Layer 3
LCM	Life-Cycle Management
LDE	Local Decision Engine
LoRa	Long range, low power wireless platform
LTE-M	Long Term Evolution for Machines
M2M	Machine to Machine
MANO	Management and Orchestration
Mbit	Megabit
MCPTT	Mission Critical Push to Talk
MEC	Mobile Edge Computing
ML2	Modular Layer 2
MPLSoUDP	Multiprotocol Label Switching over User Datagram Protocol
MQTT(S)	Message Queuing Telemetry Transport (Secure)
NA	Network Applications
NAT	Network Address Translation
NB IoT	Narrowband Internet of Things
NE	Network Element
NF	Network Function
NFP	Network Forwarding Path
NFV	Network Function Virtualization
NFV MANO	Network Function Virtualization Management and Orchestration
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator
NFVO+MEO	Network Functions Virtualization Orchestrator + Major Equipment Operator
NGCN	Next Generation Core Networks
NIC	Network Interface Card
NO	Network Operator
NR	New Radio
NS	Network Service
NSD	Network Service Descriptor
NSI	Network Slice Instance
NSMF	Network Slice Management Function
NSP	Network Service Provider
NSS	Network Support Services
NSSMF	Network Slice Subnet Management Function
OAI	Open Air Interface

O&M	Operation and Maintenance
ONAP	Open Network Automation Platform
OoM	Out of Memory
Open-O	Open Orchestrator Project
OPNFV	Open Platform for NFV
OR-VI	Cross Plane Orchestrator – Virtualized Infrastructure Manager
OR-VNFM	Cross Plane Orchestrator - VNF Manager
OSA	One Stop Application Programming Interface
OSM	Open Source MANO
OS-MA	OSS/BSS - NFV Management and Orchestration
OSS	Operational Support System
P&P	Plug and Play
P2P	Peer to Peer
PaaS	Platform As A Service
P-GW	Packet Data Network Gateway
PNF	Physical Network Function
Po	Portchannel
QCI	Channel Quality Indicator
QoE	Quality of Experience
QoS	Quality of service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RPC	Remote Procedure Call
RRM	Radio Resource Management
SBA	Service-Based Architecture
SDN	Software Defined Networks
SLA	Service Level Agreement
SLICENET	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks
SaaS	Software as a Service
SSaS	Scalable Slicing Implementation
SSD	Solid State Disk
TCP	Transmission Control Protocol
ToR	Top of the Rack
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
UP	User Plane
UUID	Universally Unique Identifier
V2X	Vehicle to Everything
vComputing	virtual Computing
vEPC	virtual Evolved Packet Core
VIM	Virtual Infrastructure Manager
VI-VNFM	Virtualized Infrastructure Manager – VNF Manager
VLAN	Virtual Local Area Network

VLD	Virtual Link Descriptor
VLT	Virtual Link Trunking
VM	Virtual Machine
VNF	Virtualized Network Function
VNFC	Virtual network Function Component
VNFD	Virtualized Network Function Descriptor
VNFFGD	VNF Forwarding Graph Descriptors
VNFM	Virtualized Network Function Manager
vPC	Virtual Port Channel
vStorage	Virtual Storage
VXLAN	Virtual Extensible Local Area Network
WAN	Wide area network
WP	Work Package
WWW	World Wide Web

Definitions

Enterprise Private Cloud Infrastructure is a type of cloud computing infrastructure that has the same characteristics and advantages as a public cloud including even more benefits but at a specific cost. The main difference is that this type of cloud infrastructure is a private one, it is developed, installed and administrated for dedicated needs of one and specific private organization. This offers connectivity and all additional services only for one company/institution being able to share it in special cases.

Slicing-friendly infrastructure is the proposed SliceNet approach for an efficient, low-cost, fast deployment and provision for a specific vertical's slice through software-networking 5G proof-of-concepts, investigating the mechanism and tools for a friendly software programmable infrastructure, allowing QoS and QoE awareness and enhanced control.

1 Introduction

1.1 Objective of this document

The scope of the this deliverable is to adapt into the SliceNet 5G architecture the scenario of the connected virtualized enterprise implementation and the specific services, by transforming the classic enterprise concepts into the future developments, requiring new approaches and technical implementations, including programming and automation into the Telco field. The document is intended also to design and prototype the concept of the slicing friendly into the global 5G scenario.

As already described into the previously deliverables, mainly into the Overall Architecture document D2.2 [1], there are defined pillars, as the SliceNet foundation, including network and vertical network slicing concepts, OSA, network management, configuring and automation and cross-plane orchestrators capabilities, aiming for orchestration the E2E slicing concept, related to the verticals requirements.

The role of the enterprise platform prototyping is strictly related to the Internet of Things (IoT) applications and the required transformations in every industrial sector and in particular in the Smart City vertical, as a platform that will be automatically deployed and scaled to accommodate millions of connected devices. It will be designed as a core IoT software platform, which is in fact an IoT Platform as a Service (PaaS).

The Enterprise IoT platform within the 5G networks context is intending to enable the designing of a cost-efficient prototyping platform for sensors connections, easy to be adaptable to the needs and capable to be extended and to any IoT scenario. The expected business evolution of the platform is related to the public sector applications, as a City Hall, due to the fact that today many municipalities are using an aging infrastructure. For sake of clarity, the IoT platform will be implemented using an open source application (app), as the platform to be deployed and prototyped also may be exposed to different scenarios of developments, as a simplified ecosystem for building and managing the app into a more automatic and programmable perspective, by using specific Application Programming Interfaces (APIs), as generic described in SliceNet as OSA. Even the output of the prototyping will not be presented as a mature product, commercial ready, we will be able to access and use a PaaS capable to be consumed by different 5G verticals and services.

The deliverable is a way of innovating the enterprise applications and use in the 5G area, and also to extend the management, configuration and related data to the E2E slicing friendly scenario.

1.2 State-of-the-art

The chapter provides a vision of the Enterprise state-of-the-art for IoT, as IoT platforms must be innovative, simplified, and easy to be used and integrated with different systems and apps that are addressing different communities.

There are series of commercial and open-source project that addresses the IoT platform, already integrated with the data sensors and IoT Gateways, as described in D3.1 [2], using performant data process engines and storage capacities and having the capability of exposing the results of the data processing to analytics dashboards, including the capabilities of taking decisions.

Commercial cloud IoT PaaS, as:

- Amazon Web Services (AWS) IoT Platform;
- Microsoft Azzure IoT Hub;
- IBM Watson IoT Platform;
- Google Cloud Platform;
- Cisco IoT cloud connect.

Open source IoT platforms:

- ThingsBoard IoT Platform;
- Kaa IoT platform;
- Thingspeak platform;
- Eclipse foundation.

The IoT platform, open source project, is enabled by a rapid development, providing an IoT cloud on the customer premises, for various IoT apps, that must be scalable, resilient, efficient, customizable and friendly to be integrated. In fact the IoT open platform is the software middleware used for connecting, acquisition and data processing from different sensors. The prototyping of the IoT platform is the foundation of the next years IoT communications, as it is expected that tens of billions of devices will be connected during the next years, for this argument raising the need of platform deployments into a customer-center manner.

The concept of the Enterprise IoT platform, software model, is based on a series of standards and engineering implementations, from technical perspective to business model adaptation, in our particular case the IoT platform for the Smart City Apps.

In SliceNet 5G E2E scenario the state-of-the-art of the development and prototyping the IoT Enterprise platform are related to several key points aspects:

- Digitalization, from consumer and also from communication service provider perspective;
- IoT platform, centralized approach, as an open middleware;
- Adopting the IT technology at the customer level (e.g. City Hall), including the enriched capabilities of virtualization, automation and programmability;
- Openness to different apps;
- E2E services deployments from catalogue;
- Intra and inter-domain capabilities, including the data plane Quality of Service (QoS) assurance;
- Relation to the new functions and logical block defined in the 5G SliceNet Architecture (OSA, Plug and Play (P&P), cognition, monitoring and software sensors implementation;
- 5G IoT Enterprise platform with respect of the Management, Control and Data Plane, as defined in previously architecture deliverable;
- Data processing and analysis, visualization and exposure;
- Platform control, including Fault, Management, Configuration, Accounting, Performance, Security (FCAPS) capabilities;
- Integration into the multi-domain 5G scenario.

The task will conclude that from the underlines of the IoT today's platform characteristics and implementations, current approaches as state-of-the-art we will expose our vision and future implementations regarding IoT enterprise architecture, as a major component into the next 5G generation networks.

1.3 Approach and Methodology

This section is related to the SliceNet Work Packages (WPs) interaction and describes the SLICENET project workflows, highlighting mainly the effort of WP2 [3], the SliceNet System Definition, and especially outputs from T2.1 [4], related to vertical sectors requirements.

The task SliceNet 5G-Connected Virtualized Enterprise Infrastructure and Services extends the E2E slicing concepts to the enterprise level.

The applied integration and verification process for prototyping can be observed from Figure 1.

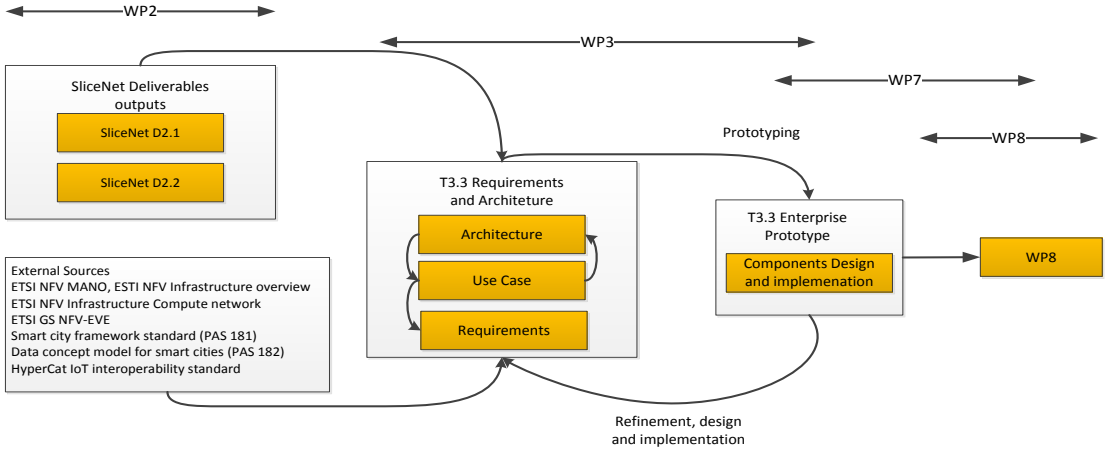

Figure 1 Applied integration and verification process for prototyping

1.4 Document Structure

This document is composed by nine chapters, as the first chapter, Introduction, presents the objectives of this document, state-of-the-art, approach and methodology, SliceNet requirements and vision and the transition of the existing enterprise infrastructure and services to a 5G-connected, virtualised deployment. Chapter 2 presents the Enterprise Data Plane which is composed by: Enterprise Infrastructure (Smart City IoT segment) and Enterprise Private Cloud (Private cloud for Smart City). In chapter 3, it is described the SliceNet vision about 5G Connected Services, Smart City Applications and Services. Chapter 4 describes the SliceNet vision about Control Plane for Enterprise (VIM and SDN Controller). SliceNet vision about Management Plane for Enterprise (Monitoring, LCM and Configuration Management) is presented in chapter 5. Chapter 6 presents the Cross-Plane Orchestration for Enterprise in SliceNet vision, Smart City Cross-Plane orchestration and Enterprise integration in SliceNet E2E architecture. OSA for Enterprise, OSA for Smart City and its integration in SliceNet E2E architecture are presented in chapter 7.

Chapter 8 describes the main part of this deliverable: the prototyped friendly enterprise network segment for Smart City use case, by detailing the Smart City segment, resources modelling and design, and prototyping the: Physical Layer, the Enterprise cloud (SDN and VIM), LCM, services and applications. Finally, chapter 9 brings the conclusions.

1.5 SliceNet Requirements and Vision

This subchapter describes the SliceNet vision, key drivers and general requirements about the virtualized enterprise infrastructure.

From the enterprise networks perspective, the challenge regarding 5G is to provide E2E network and cloud infrastructure slices over the physical infrastructure in order to fulfil specific requirements for the Smart City use case.

The main role of the enterprise networks will be to identify the key vertical sectors' requirements, anticipating relevant trends early and mapping them into the 5G SliceNet design. A successful Enterprise Infrastructure must be shared, secure and scalable and it must enable a smarter, safer and more sustainable development for each the use cases, by realizing multiple, highly, flexible, E2E dedicated network and cloud infrastructure slices over the same physical infrastructure, in order to fulfil specific requirements.

The Enterprise Segment for Smart City will connect sensors, machines, city administrations and citizens to cloud-based IoT apps as we can observe from Figure 2.

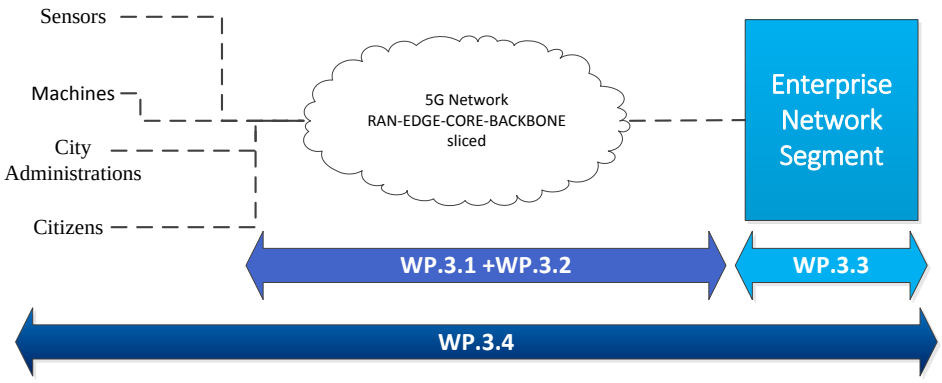

Figure 2 E2E connectivity for Enterprise Network Segment

The integration of enterprise Network Segment for Smart City use case in SliceNet context can be observed from Figure 3.

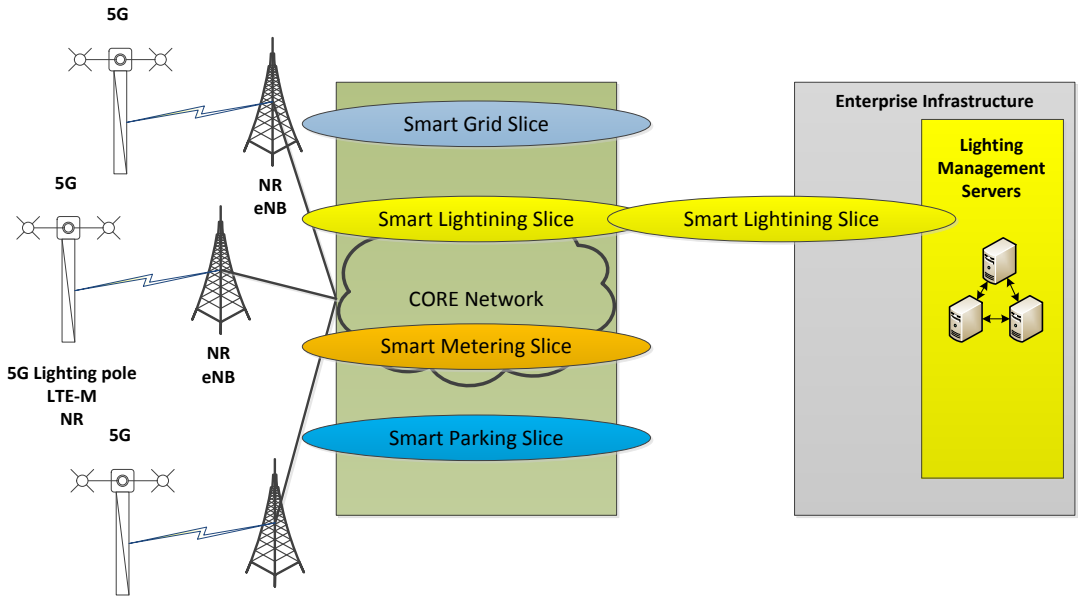

Figure 3 Enterprise Network Segment for Smart City use case in SliceNet context [4]

Taking into account the aspects presented in the deliverable D2.1 [4] about the requirements related to the Smart City use case, the aspects related to the architecture on SliceNet vision presented in D2.2 [1], SliceNet vison about Control plane (CP) presented in D2.3 [5] and about Management Plane in D2.4 [6] and correlate these SliceNet WPs results with the initial project proposal [3], it is proposed the overall diagram presented in Figure 4, also fixing the main objective of this task: create a 5G Connected Virtualized Enterprise Infrastructure that could serve the Smart City use case from the enterprise perspective and after that this platform could be adapt to support any other 5G use case or IoT scenario.

In order to fulfil the requirements listed above, the Enterprise infrastructure must be composed by several planes: Data Plane (composed by Enterprise Infrastructure and Enterprise Private Cloud), Enterprise Applications and Enterprise Services, CP, Management Plane and Cross Plane Orchestration under the umbrella of OSA. The proposed structure is presented in the diagram from Figure 4.

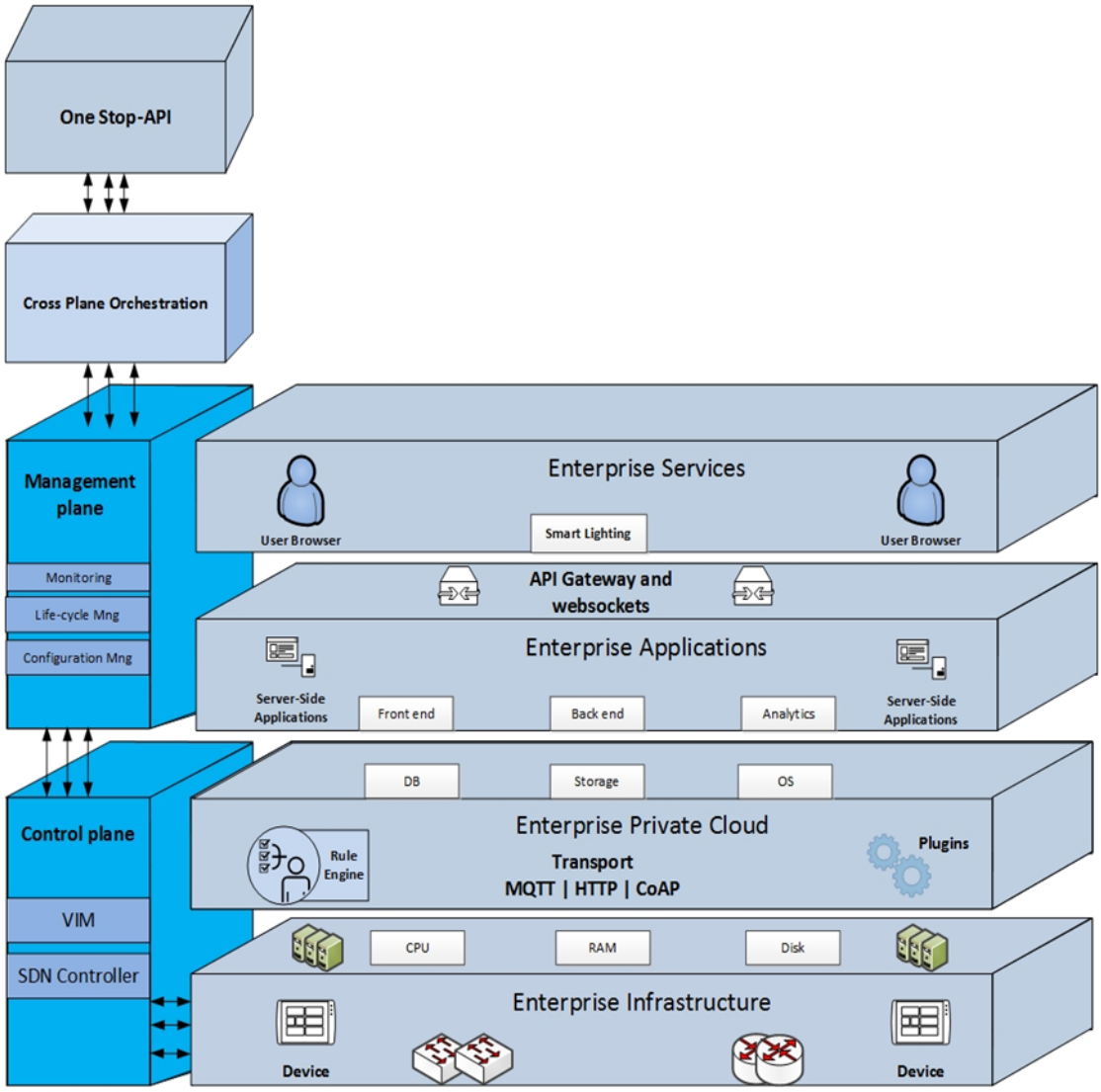

Figure 4 Enterprise Segment overall diagram

The first plane of the enterprise architecture is the Data plane (infrastructure) which contains an E2E heterogeneous network and distributed cloud platform. The Enterprise Services plane defines and implements the services of the use case. The Enterprise

Applications plane consists in databases, subscriber portal, back-end apps and scripts to fulfil the upper layer services requirements and constraints. The synchronization and interworking between these components will provide services to customers.

CP for Enterprise implements the abstractions provided by software networks technologies (essentially as SDN and NFV) to support an abstracted model for the 5G networks functions. The overall purpose of the Management Plane is to enable the monitoring, LCM and configuration mechanisms required to assemble the supported virtual resources running network functions.

With respect to use cases requirements, the enterprise infrastructure should provide reliability and availability up to 99.95%. The Enterprise network must accommodate a high number of devices per unit area that are 5G capable, although they might not all be generating traffic simultaneously for the specified app. For each of the three use cases, the Enterprise network should provide a maximum positioning error tolerated by the app. The service deployment time is also a requirement and represents the duration required for setting up the network slices in order to provide services to end customer. The enterprise infrastructure must assure the protection of data, encompassing several level of security such as authentication, data confidentiality, data integrity and access control in a multi-tenant environment. Enterprise infrastructure should focus in minimizing the power consumption and provide scaling capabilities according to network load.

Bellow table summarizes the use cases requirements and their applicability to each enterprise infrastructure layer.

Table 1 Requirements applicability to infrastructure layer

Requirements for Smart City use case per each Enterprise Layer	Reliability & Availability	Density	Position accuracy	Service deployment time	Security & Privacy	Low power consumption
Enterprise infrastructure	Yes	Yes	No	Yes	Yes	Yes
Enterprise private cloud	Yes	Yes	No	Yes	Yes	Yes
Enterprise apps	Yes	Yes	Yes	Yes	Yes	Yes
Enterprise services	Yes	No	Yes	No	Yes	No
CP	Yes	No	No	Yes	Yes	Yes
Management plane	Yes	No	No	Yes	Yes	Yes
Cross plane orchestration	Yes	No	No	Yes	Yes	Yes
OSA	Yes	No	No	Yes	Yes	No

1.6 The transition of the existing enterprise infrastructure and services to a 5G-connected, virtualised deployment

Upon the rise of a 5G slicing era, there is no need for dedicated network infrastructures to support enterprise environments, as Service Level Agreements (SLA) can be met via slicing and with proper QoS support. This requires control and management interfaces that allow

enterprise administrators to take control (i.e., ownership) over their network without interfering with the rest of the virtualized networks, which is referred as the *isolation property*. Furthermore, network services (NSs) can be provisioned on demand and then deployed in a virtualized infrastructure. This offers indeed a transition between vertical dedicated networks, which in the current reality correspond to a different use case each, towards a shared, customizable horizontal network. Due to this desired transition towards a shared – yet customizable — horizontal network infrastructure, the concept of 5G slicing is one of the most important communication innovations of the current times, yielding a series of anticipated benefits, as network slices can maximize the sharing of network resources across domains as well as within domains. Evidently, this reduces the Capital Expenditure (CapEX) substantially for network operators.

Slicing also allows a high flexibility for creating dedicated logical networks with customer-specific functions, which can meet the diverse requirements of vertical businesses, along with all implied QoS requirements. QoS requirements can vary from hard QoS guarantee requirements, which demand for a clear resource reservation, to soft QoS guarantee requirements that can be addressed with a combination of resource reservation and multiplexing. Regarding the 5G Radio Access Network (RAN) segment for instance, Table 2, shows the standardized Channel Quality Indicator (CQI) values [7] that associate QoS characteristics in terms of resource type priority (Guaranteed Bit Rate (GBR) or not), scheduling priority level, packet delay budget and error loss rate to a standardized CQI index values, edge-to-edge traffic between UEs and the Packet Data Network Gateway (P-GW). Based on this table, use case examples implying hard QoS requirements can include autonomous driving safety, which is based on ultra-low-latency for message exchanges, and reliable remote mobile life assistance, which is based on mobile broadband Ultra-High-Definition video streaming and ultra-low-latency. Last, we note that addressing QoS requirements on a per slice basis (i.e., a slice-aware resource slicing with QoS support in mind) requires extensions to the current QoS mechanisms.

Table 2 Channel Quality Indicator (CQI) defined in the 36.213 rel 14 standard [7]

QCI	Resource Type - GBR or not	Priority	Packet Delay Budget	Packet Error Loss Rate	Example Services
1	GBR	2	100ms	10^{-2}	Conversational Voice
2	GBR	4	150ms	10^{-3}	Conversational Video (Live Streaming)
3	GBR	3	50ms	10^{-3}	Real Time Gaming, Vehicle to Everything (V2X) messages
4	GBR	5	300ms	10^{-6}	Non-Conversational Video (Buffered Streaming)
65	GBR	0.7	75ms	10^{-2}	Mission Critical user plane Push To Talk voice (e.g., MCPTT)
66	GBR	2	100ms	10^{-2}	Non-Mission-Critical UP Push To Talk voice
75	GBR	2.5	50ms	10^{-2}	V2X messages
5	non-GBR	1	100ms	10^{-6}	Internet Protocol Multimedia Subsystem (IMS) Signalling
6	non-GBR	6	300ms	10^{-6}	Video (Buffered Streaming) Transmission Control Protocol (TCP) -Based (for example, World Wide Web (www), email, chat, ftp,

					Peer to Peer (p2p) and the like)
7	non-GBR	7	100ms	10^{-3}	Voice, Video (Live Streaming), Interactive Gaming
8	non-GBR	8	300ms	10^{-6}	Video (Buffered Streaming) TCP-Based (for example, www, email, chat, ftp, p2p and the like)
9	non-GBR	9	300ms	10^{-6}	Video (Buffered Streaming) TCP-Based (for example, www, email, chat, ftp, p2p and the like). Typically used as default bearer
69	non-GBR	0.5	60ms	10^{-6}	Mission Critical delay sensitive signalling (e.g., MC-PTT signalling)
70	non-GBR	5.5	200ms	10^{-6}	Mission Critical Data (e.g. example services are the same as QCI 6/8/9)
79	non-GBR	6.5	50ms	10^{-2}	V2X messages

In order to achieve the transition of the existing enterprise infrastructure and services to a 5G virtualized deployment, intelligent slicing offers configurable warranties in terms of QoS and/or Quality of Experience (QoE), paving the way for new markets and a wide range of diverse and innovative use cases, including those with hard QoS requirements. The underlying objectives are, first, to transit to a virtualized environment that is oriented towards verticals and desired QoE, along with a focus on cognitive network management, and second, to transit to a virtualized environment with control for E2E slicing operation and slice-based services across multiple operator domains, using the concepts of SDN and NFV. In addition, there is a notion of a “one-stop shop”, which facilitates the smooth and efficient migration of enterprises from the current reality to that of 5G slices for enhancing current use cases as well as for creating new ones. The latter can be done in terms of innovative use case onboarding, prompt slicing provisioning, flexible and efficient control and management.

Another important aspect towards moving to a 5G-connected virtualized deployment is traffic isolation, which can be achieved via network traffic control with a programmable Data plane. Traffic Isolation refers to performance isolation and/or subnetwork isolation in terms of middleboxes for reaching an app server (such as switched and routers). Also, it can refer to isolated tenant networks for the purpose of per tenant/user traffic differentiation. Physical resource slicing within an isolated tenant network provides a guarantee of hard QoS requirements in terms of network metrics (bandwidth, latency/delay, jitter, etc.). Further, it can guarantee user-specific QoS requirements for different users in the tenant network, via virtualized network control functions that offer the required control capabilities as well as mobility management for same-tenant UEs.

In addition, virtualized deployments move away from the traditional (and simpler) hardware-based data path model, towards a (more complex) both hardware-based and software-based data path model. Specifically, Virtual machines (VMs) can be interconnected with both software-data paths and with physical ports of Commercial Off-The-Shelf (COTS) computers. The combination of hardware-based and software-based data paths leads to control and monitoring points along E2E paths from source to destination. Regarding physical data paths, each machine has to provide at least two physical network interfaces, possibly connected to different hardware forwarding devices, respectively, interconnecting the network segments involved in the connectivity of the computer. As explained in further detail in section 5.1 of D2.3 [5] about QoS support for the non-RAN segment, high-end hardware-based network cards can provide basic network traffic telemetry and embedded

control functions that allow the configuration of existing control functions, with other approaches being usually based on programmable hardware data paths with Field Programmable Gate Arrays (FPGA) network cards. On the contrary, the software-based approach has three different network control and telemetry points: one in the host machine; another one in the software switch; and finally, one inside of the data path of the virtualized network control function. Last, regarding QoS support in the RAN Segment, a RAN controller provides runtime UE monitoring, control, and coordination to support QoS. This allows adapting UP functions after spatio-temporal traffic and network dynamics, with each RAN module (given in further detail in section 5.1 of D2. 3 [5]) being decomposed into the control logic of the radio link and the control action that applies logical decisions.

2 Enterprise Data Plane

2.1 Enterprise Infrastructure

From the connectivity perspective, the enterprise infrastructure can be summarized in three layers:

- Physical networks - Datacenter (DC) gateways, Top of the Rack (ToR) switches, blades Network Interface Card (NIC);
- Underlay networks - based on Virtual Local Area Network (VLAN) and created during the infrastructure provisioning;
- Overlay networks - based in tunnelling mechanisms like Virtual Extensible Local Area Network (VXLAN) or Multiprotocol Label Switching over User Datagram Protocol (MPLSoUDP) and used for traffic isolation between tenants.

The following subchapter presents the general concept of NFV, Management and Orchestration (NFV MANO), and Business Support System (BSS) and Operational Support System (OSS) functional blocks, communication interfaces between them and the functions and procedures that allow the creation of apps. The Management and Orchestration (MANO) framework proposed will be the base of the development and implementation for prototyping IoT platform [8] [9]. This subchapter is made in line with the SliceNet logical architecture proposed and detailed in D2.2 [1] and which is figured in Figure 5.

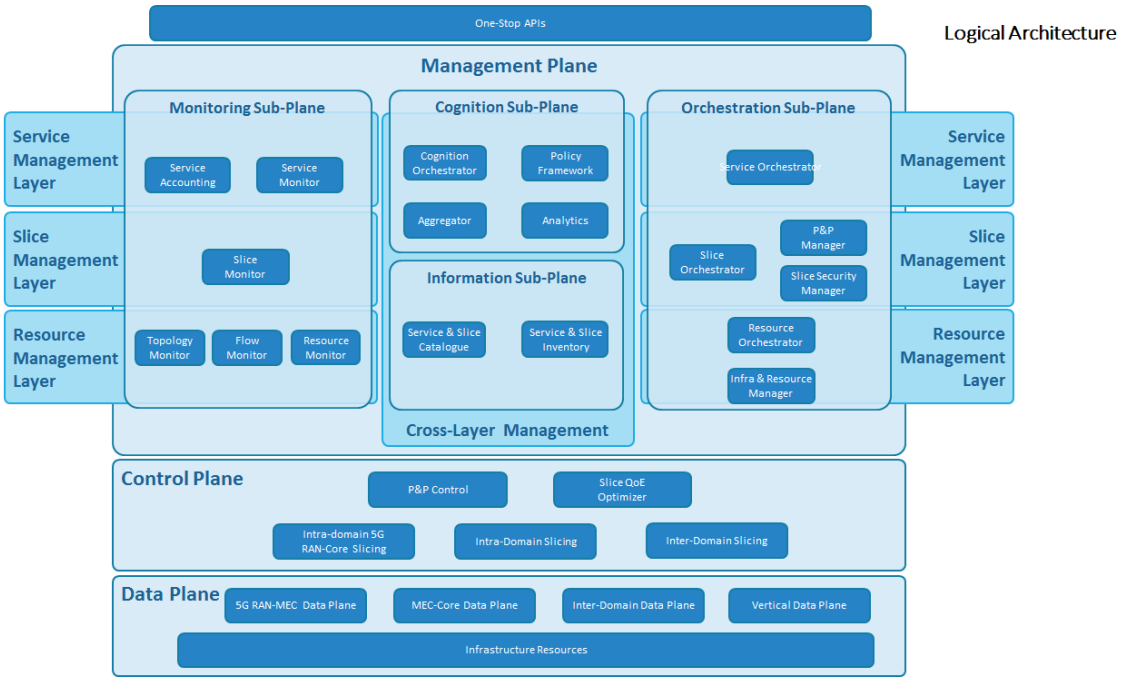

Figure 5 Slicent Logical Arhitecture [1]

NFV provide the aspect of implementation of Network Functions as software only entities that run over the Network Function Virtualization Infrastructure (NFVI) [9]. In Figure 6 it is illustrated the high-level NFV framework.

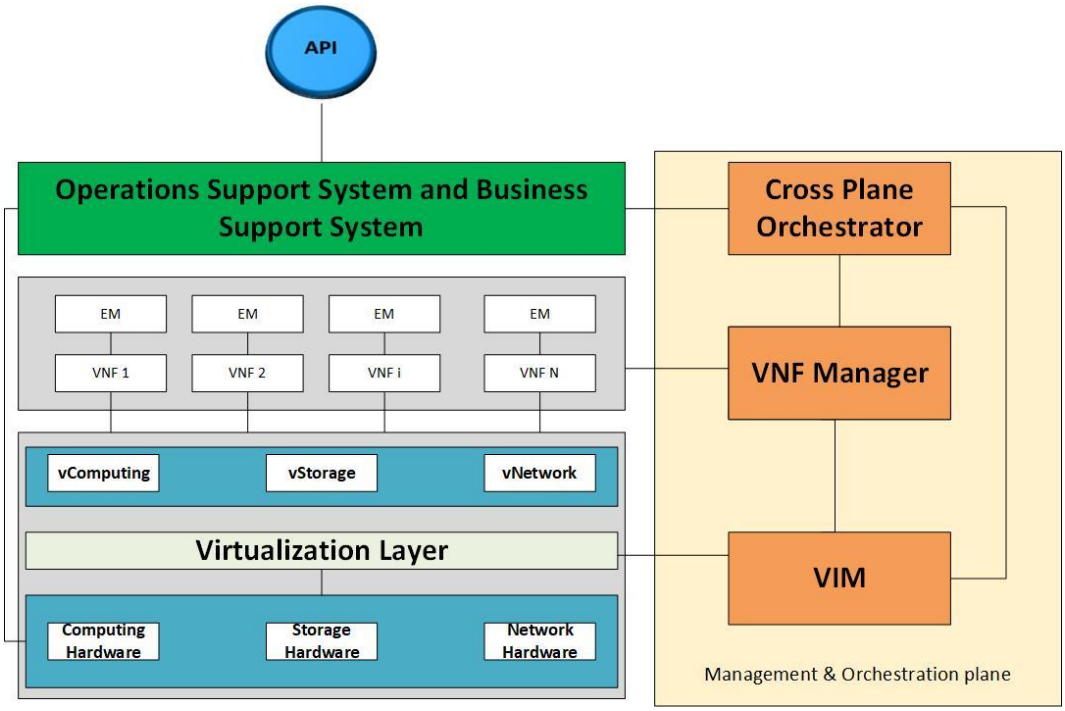

Figure 6 High Level NFV architecture

Can be observed, from Figure 6, four main working domains:

- VNF, as the software implementation of a network function which is capable of running over the NFVI [9].
- NFVI, including the diversity of physical resources and how these can be virtualized. NFVI supports the execution of the VNFs [9].
- Network Function Virtualization Management and Orchestration (NFV MANO), which covers the orchestration and LCM of physical and/or software resources that support the infrastructure virtualization, and the LCM of VNFs. NFV MANO focuses on all virtualization specific management tasks necessary in the NFV framework.
- OSS and BSS. OSS is software, in some cases hardware, apps that support back-office activities, which operate a telecommunication network, provision and maintain customer services. BSS is a software app that supports customer-facing activities. Billings, order management, customer relationship management, call centre automation. BSS may also encompass the customer-facing a thin layer of OSS app such as trouble ticketing and service assurance.

To fulfil our Enterprise Segment based on an ETSI MANO model, we'll implement as follows:

- **Cross Plane Orchestrator** is composed of the following modules: *heat-api-cfn* and *heat-engine*. Heat-api-cfn is a module that exposes an external Representational State Transfer APIs (REST) based api to the heat-engine service. The communication between the heat-api-cfn and heat-engine uses message queue based Remote Procedure Call (RPC) [10]. Heat-engine does all the orchestration work and is the layer in which the resource integration is implemented [11].
- **Management Plane** is composed of three main components, **Metering** which is composed of the *ceilometer* service (ceilometer service is composed of several modules who combine their responsibilities to collect, normalize and redirect the data [12]), **LCM** which is composed of the *heat-api* module, which is responsible to

expose an external REST based api to the heat-engine service and **Configuration Management** which is composed of three modules: *nova-compute*, who is responsible for building a disk image, lunching it via the underlying virtualization driver, responding to calls to check its state, attaching persistent storage and terminating it [13]; *cinder-volume*, who is an REST based api responsible to trigger the creation of a logical volume on the storage node; and the *glance-manage* who is a utility for managing and configuring a glance installation.

- **CP** is composed by two main components: **VIM** and **SDN**. VIM is composed by four modules: *nova-api, nova-conductor, cinder-api and glance-api.* The *nova-api* module is a server daemon that serves the metadata and compute APIs in separate greenthreads, the *nova-conductor* module is a server daemon who provides coordination and database query support for nova service, the module *cinder-api* in OpenStack Ocata (15.0) is deprecated now, and is under *cinder-wsgi* who works under apache service but the main roles are the same, to uses eventelt as a webserver and wsgi application manages. Eventlet provides own wsgi application to provide web server functionality [14]. The module *glance-api* servers the service where users can upload and discover data assets that are meant to be used with other services [15]. SDN is composed by the *neutron* service. Neutron offers the functionality to create and attach interface devices managed by other OpenStack services to networks [16].

The NFV framework enables dynamic construction and management of VNF instances and the relationships between them regarding data, control, management, dependencies and other attributes. To this end, there are at least three architectural views of VNFs that are around different perspectives and contexts of a VNF. Those perspectives are as follow:

- A virtualization deployment/on-boarding perspective where the context can be a VM;
- A vendor-developed software package perspective where the context can be several inter-connected VMs and a deployment template that describes their attributes;
- An operator perspective where the context can be the operation and management of a VNF received in the form of a vendor software package.

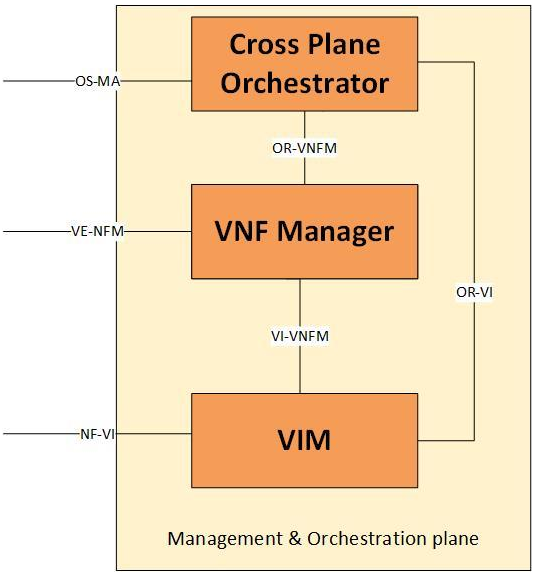

Figure 7 High Level NFV Architecture with the MANO highlight

The Management and Orchestration Plane, Figure 7, which covers the orchestration and LCM is composed by the following architectural framework functional blocks (this approach is in line with the SliceNet logical architecture presented in Figure 5):

- **Cross Plane Orchestrator**, has two main responsibilities:
  - As mentioned in D2.2 [1] the Cross Plane Orchestrator and Resource Orchestrator represents the low level orchestration for slice orchestration.
  - The orchestration of NFVI resources across multiple VIMs for each plane, fulfilling the aspects that the NFVI resources under consideration are both virtualized and non-virtualized resources, supporting VNFs and partially VNFs. Virtualized resources in-scope are those that can be associated with virtualization containers, and have been catalogued and offered for consumption through appropriately abstracted services.
  - The LCM of NSs, fulfilling the Network Service Orchestration which is responsible for the Network Service LCM including the following operations:
    - On-board NS register a NS in the catalogue and ensure that all the templates describing the NS are on-boarded.
    - Instantiate NS, create a NS using the NS on-boarding artefacts.
    - Scale NS, grow or reduce the capacity of the NS.
    - Upgrade NS by supporting NS configuration changes of various complexity such as changing inter-VNF connectivity or the constituent VNF instances.
    - Create, delete, query and update of VNF Forwarding Graph Descriptors (VNFFGs) associated to a NS.
    - Terminate NS, request the termination of constituent VNF instances, request the release of NFVI resources associated to NSs, and return them to NFVI resources pool if applicable.
    - As a set of examples an NS catalogue is contain in the messages exchanged between the Keystone and other services, as the form of links to the other services that are necessary to interact in our use cases. Keystone could give out links for more than one region, depending on the providing configuration. The code from Table 3 will search the catalogue for the compute and NS.

Table 3 The code that search the catalogue for the compute and NS

```
#Find the link to the NOVA API in the service
catalog:
for service in
r.ison()['acccess']['serviceCatalog']:
    if service['type'] == 'compute':
        nova endpoint =
service['endpoints'][0]['publicURL']
        if service['type'] == 'network':
            neutron_endoint =
service['endpoints'][0]['publicURL']
```

In the Table 4 there are provided 2 examples of NS catalogues.

Table 4 Examples of NS catologues [65]

```
"catalog": [
    {
        "name": "Keystone",
        "type": "identity",
        "endpoints": [
            {
                "interface":
"public",
                "url":
"https://identity.example.co
m:35357/"
            }
        ]
    }
]
```

```
"catalog": [
    {
        "name": "Neutron",
        "type": "network",
        "endpoints": [
            {
                "interface":
"admin",
                "url":
"https://network.example.com:
9696/"
            }
        ]
    }
]
```

The main parameters used in the Table 4 are:

1. **Endpoints**, which is an array type and represent a list of endpoint of objects.
2. **Id**, which is a string type and represent the Universally Unique Identifier (UUID) of the service to which the endpoint belongs.
3. **Type**, which is a string type and represents the service type, which describes the API implemented by the service. Its value could be compute, identity, image, network, or volume.
4. **Name**, which is a string type and represents the service name.

- **Virtualized Network Function Manager (VNFM)** is responsible for the LCM of VNF instances. Each VNF instance is assumed to have the same and the only VNF Manager. The VNF manager may be assigned the management of all VNF instance, of the same type or of different types. The non-exhaustive set of functions performed by the VNFM functions. These functionalities may be exposed by means of interfaces and consumed by other NFV-MANO functional blocks or by authorized external entities:
    - VNF instantiation, including VNF configuration if required by the VNF deployment template.
    - VNF installation feasibility checking, if required.
    - VNF instance software update or upgrade.
    - VNF instance modification.
    - VNF instance scaling out or in and up or down.
    - VNF instance-related collection of NFVI performance measurement result and faults or events information, and correlation to VNF instance-related events or faults.
    - VNF instance assisted or automated healing.
    - VNF instance termination.
    - VNF LCM change notification.
    - Management of the integrity of the VNF instance through its lifecycle.
    - Overall coordination and adaptation role for configuration and event reporting between the VIM and the Element Management (EM).

- **VIM** is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain. VIM may be specialized in handling a certain type of NFVI resources (e.g. compute-only, storage-only, networking-only), or may be capable of managing multiple types of NFVI resources (e.g. in NFVI-Nodes). The following list expresses the set functions performed by the VIM. These functionalities may be exposed by means of interfaces consumed by other NFV-MANO functional blocks or by authorized external entities:
  - Orchestrating the allocation or upgrade or release or reclamation of NFVI resources, and managing the association of the virtualized resources to the physical compute, storage, networking resources. Therefore, VIM keeps an inventory of the allocation of virtual resources to physical resources.
  - Supporting the management of VNFFGs by creating and maintaining Virtual Links, Virtual Networks, sub-nets, and ports as well as the management of security group policies to ensure network or traffic access control.
  - Managing in a repository inventory related information of NFVI hardware resources (compute, storage, networking) and software resources (e.g. hypervisor), and discovery of the capabilities and features of such resources.
  - Management of the virtualized resource capacity and forwarding of information related to NFVI resources capacity and usage reporting.
  - Management of software images (add, delete, update, query or copy) as requested by other NFV-MANO functional blocks (e.g. Cross Plane Orchestrator). While not explicitly shown in the NFV-MANO architectural framework, the VIM maintains repositories for software images, in order to streamline the allocation of virtualized computing resources. A validation step, performed by VIM, is required for software images before storing the image.
  - Collection of performance and fault information of hardware resources (compute, storage and networking) software resources, and virtualized resources.
  - Management of catalogues of virtualized resources that can be consumed from the NFVI. The elements in the catalogue may be in the form of virtualized resources that can be consumed from the NFVI. The elements in the catalogue may be in the form of virtualized resource configurations.

NFV MANO reference points describe the interface between each functional block and are described in the following lines.

- **OSS/BSS – Cross Plane Orchestrator (OS-MA-NFVO):** This reference point is used for exchanges between OSS/BSS and Cross Plane Orchestrator and supports the following Network Service Descriptor (NSD) and VNF package management, Network Slice Instance (NSI) LCM, VNF LCM, policy management and/or enforcement for NS instances, VNF instances and NFVI resources, querying relevant NSI and VNF instance information from the OSS/BSS and forwarding of events, accounting and usage records and performance measurement results regarding network service instances, VNF instances, and NFVI resources to OSS/BSS, as well as and information about the associations between those instances and NFVI resources.
- **Element Management (EM) – VNF Manager (VE-VNFM-EM):** This reference point is used for exchanges between EM and VNF Manager, only in case when the EM is aware of virtualization, and supports the following procedures, VNF instantiation, VNF instance query, VNF instance update, VNF instance scaling out or in, and up or

down, VNF instance termination, forwarding of configuration and events from the EM to the VNFM, and forwarding of configuration and events regarding the VNF from the VNFM to the EM.

- **Cross Plane Orchestrator - VNF Manager (OR-VNFM):** This reference point is used for exchanges between Cross Plane Orchestrator and VNFM, and supports the resource related requests by the VNFM, sending configuration information to the VNFM, so that the VNF can be configured appropriately to function within the VNFG in the NS and collecting state information of the VNF necessary for NS LCM.
- **Cross Plane Orchestrator – Virtualized Infrastructure Manager (OR-VI):** This reference point is used for exchanges between Cross Plane Orchestrator and VIM, and supports the resources reservation and/or allocation requests by the Cross Plane Orchestrator and virtualized hardware configuration and state information exchange.
- **Virtualized Infrastructure Manager – VNF Manager (VI-VNFM):** This reference point is used for exchanges between the VIM and VNFM, and supports the resources allocation requests by the VNFM, and virtualized hardware resources configuration and state information exchange.

The NFVI can be defined as the totality of all hardware and software components which build up the environment, in which VNFs are deployed, managed and executed. From the VNFs point of view, the virtualization layer and the hardware resources looks like a single entity providing them with desired resources.

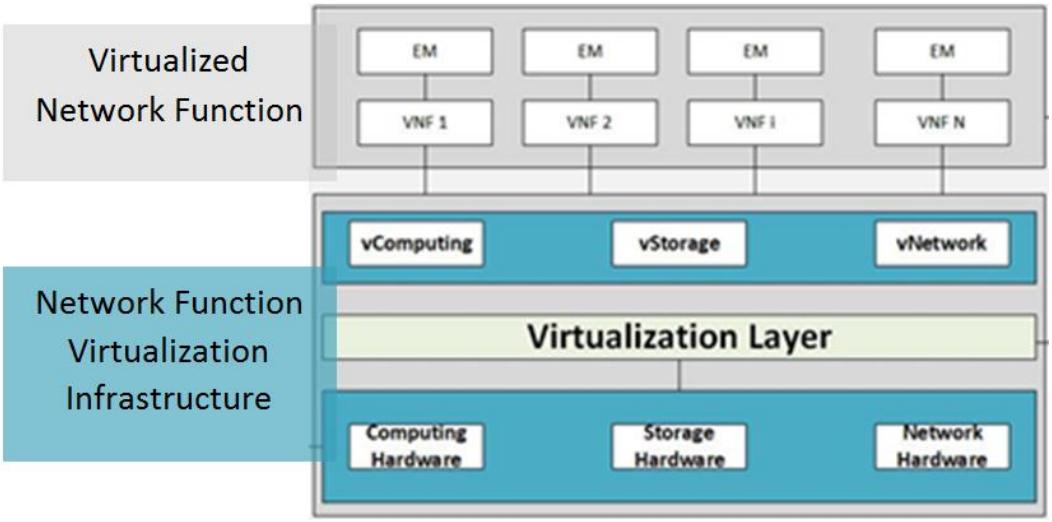

Figure 8 High Level NFV Architecture with the NFVI and VNFs highlight

The hardware resources displayed on the bottom of the Figure 8 include computing, storage and network that provide processing, storage and connectivity to VNFs through the virtualization layer. Storage resources can be differentiated between shared network attached storage and storage that resides on the server itself.

The virtualization layer abstracts the hardware resources and decouples the VNF software from the underlying hardware, thus ensuring a hardware independent lifecycle for the VNFs. The virtualization layer has the responsibilities for abstracting and logically partitioning physical resources, commonly as a hardware abstraction layer, enabling the software that implements the VNF to use the underlying virtualized infrastructure and providing virtualized resources to the VNF.

The architectural view of the NFVI and NFV is presented in Figure 6, where the virtualization layer in the middle of the NFVI ensures that the VNFs are decoupled from hardware resources and therefore, the software can be deployed on different physical hardware resources. In the scope of this deliverable this virtualization layer will be managed by the open-sources software OpenStack based.

2.1.1 Smart City IoT segment

The enterprise infrastructure is using the Openstack framework with all relevant services in order to provide Infrastructure as a Service (IaaS) type of solution for hosting the IoT platform. Bellow diagram, from Figure 9, depicts the Openstack services and their interaction with respect to Enterprise Segment overall proposal.

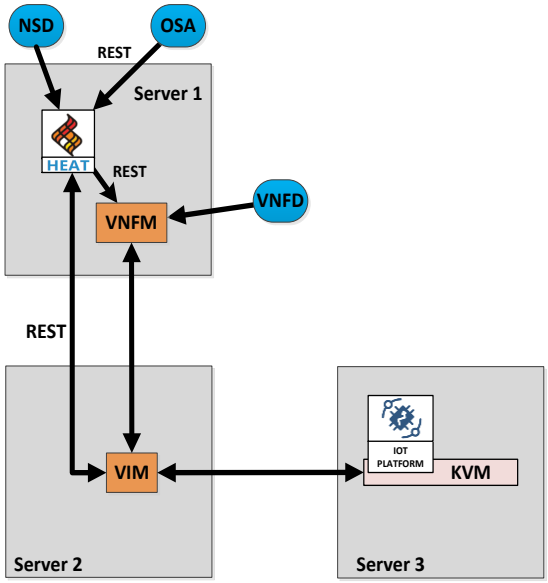

Figure 9 Openstack services and their interaction with respect to Enterprise Segment

The provisioning flow of Smart City service would be based on following steps:

1. A request from OSA is sent to Heat orchestration engine containing the NSD identifier.
2. Heat is analysing the request and the NSD.
3. Heat sends a request to VNFM Murano for IoT platform instantiation.
4. Murano analyses the IoT platform requirements based on associated VNFD.
5. Murano requests the granting of LCM to Heat.
6. Heat authorizes the request and transmits to Murano the associated VIM details.
7. Murano is sending a request to VIM in order to allocate network.
8. VIM creates the network using the Neutron service and confirms the allocation to Murano.
9. Murano is sending a request to VIM in order to allocate a compute host.
10. VIM creates a VM according to the compute requirements of IoT platform.
11. VIM confirms the instantiation to Murano.
12. VNFM confirm the deployment to Orchestrator.

These steps described above could be observed from Figure 10.

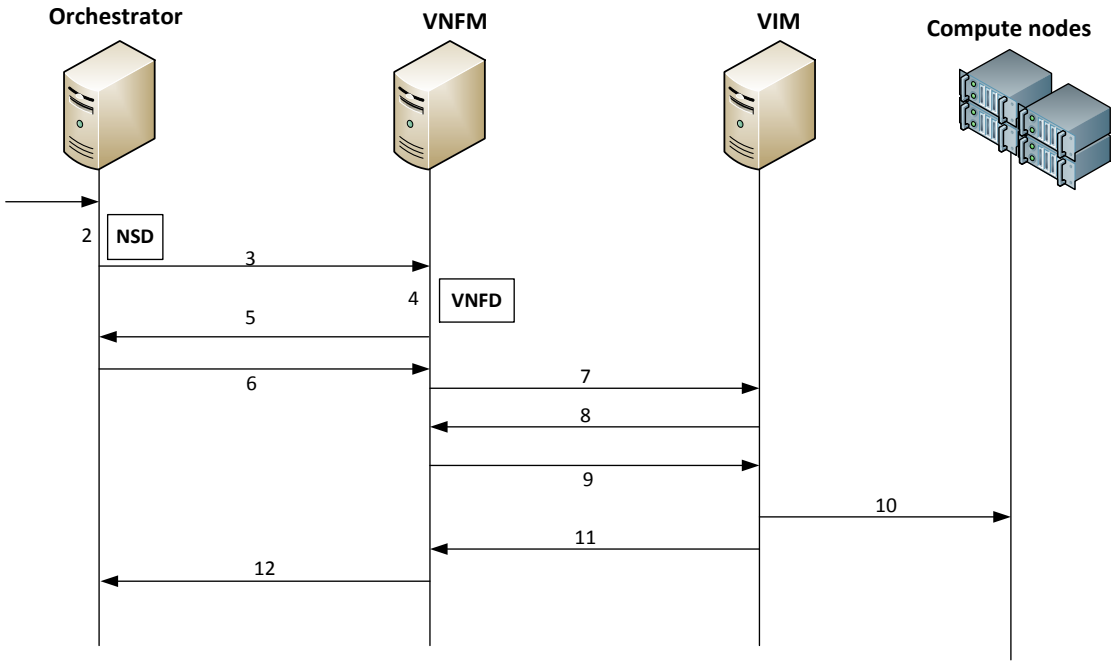

Figure 10 IoT platform instantiation

The physical architecture for Smart City IoT segment is composed by:

- two routers which act as gateways to provide demarcation between Wide Area Networks (WAN) and Enterprise infrastructure;
- two aggregation switches in order to allow seamless expansion of the infrastructure in the future;
- two ToR switches used for servers' connectivity;
- three servers to host the above depicted infrastructure.

Based on the calculations detailed in Subchapter 8.2.1, each server has the following hardware capabilities:

- processors, 12 core/processor @ 2.4 GigaHertz (GHz);
- 128 Gigabyte (GB) RAM;
- TB Hard Disk;
- Network adapters of 1 Gigabits per second (Gbps).

Having the target to build a resilient infrastructure, bellow diagram from Figure 11 depicts the physical components and their connectivity.

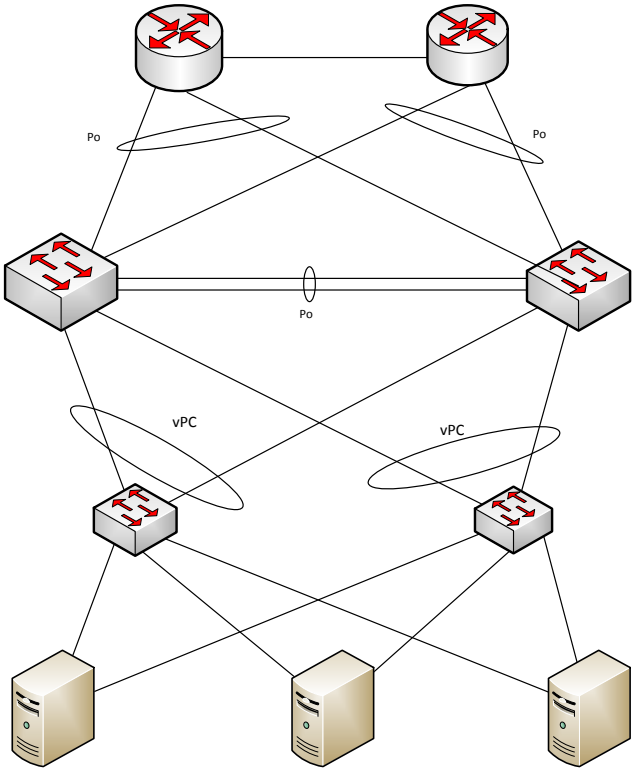

Figure 11 Enterprise physical architecture

The underlay networks are used for transport of the remote management, Control and Data planes traffic and relay on VLANs for segmentation. Following the Figure 12, the administrative subnet is used for remote management access to all nodes of the infrastructure. The internal subnet is used for communication between nodes, being accessed by the REST of the Openstack framework and being reachable only within the infrastructure nodes. Traffic subnet is used to communicate with external networks and will allow sensors data to reach the IoT platform. The scope of the overlay subnet is to provide communication between internal IoT platform components and is dynamically created based on Neutron inputs.

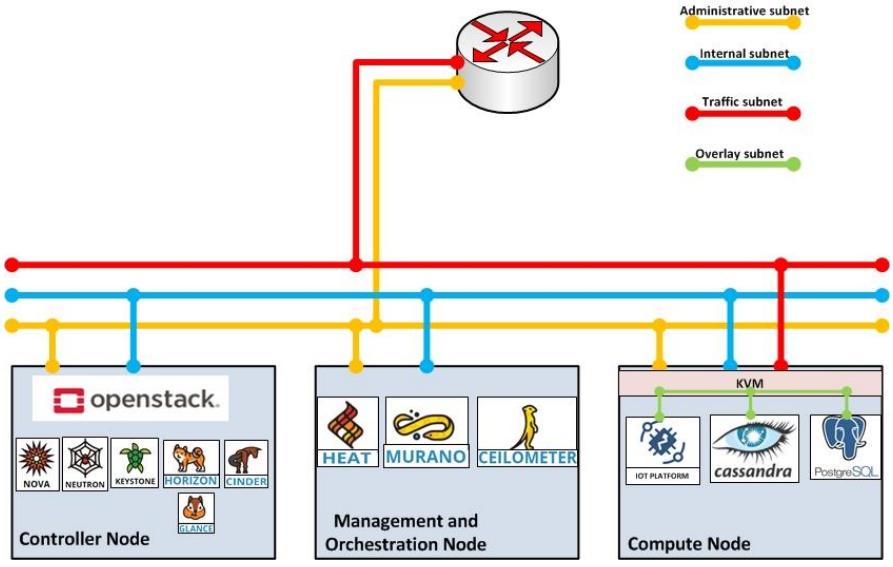

Figure 12 Connectivity in Enterprise Infrastructure

2.2 Enterprise Private Cloud

Enterprise Private Cloud Infrastructure is a type of cloud computing infrastructure that has the same characteristics and advantages as a public cloud including even more benefits but at a specific cost. The main difference is that this type of cloud infrastructure is a private one, it is developed, installed and administrated for dedicated needs of one and specific private organization. This offers connectivity and all additional services only for one company/institution being able to share it in special cases.

Even if this approach comes with multiple benefits (like security, resource management, architectural choices, etc.) has also two drawbacks. The first one is strongly related to the entire cost of the project with all necessarily hardware equipment provisioning and installation. This could generate a big investment and a grow on CAPEX side. The other one is related to the implementation time and also the time spent on trainings with every member of the team involved in the project. It is more cost effective to buy a subscription to an existing cloud platform than to implement this entire project.

In general, there are three types of deployment for cloud infrastructure:

- Public cloud
- Private cloud
- Hybrid cloud

In the following sentences it is described shortly every type of it.

Public cloud infrastructure represents a type of virtualized environment mainly owned by a third party, a company/service provider with many resources behind which they are held one or many physical DCs and operational teams that can assure availability of the services. The whole variety of services and access to it follow the model of multitenant with sharable resources at hardware level, but logical distributed and separated such that can assure and secure the data of any client. Some benefits of this type of approach are:

- Payment model: typically, almost all providers offer a pay-as-you-use model, where the client pays only for the compute resources he uses. This could be a very economical way for some of use cases.
- Self-management: the user can deploy, decommission, upgrade, downgrade, start or shutdown anytime any machine without a big and skilled team.

At this moment, the top five cloud computing vendors are:

- Microsoft
- Amazon
- IBM
- Salesforce
- SAP
- Followed by Oracle, Google, VMWare and many others.

Private Cloud Infrastructure is a type of environment with a single tenant in which case all hardware resources (compute, storage), network are dedicated (and in many cases owned by) to a single client or company and all data is protected behind a firewall [16]. In general, this type of approach is dedicated and chosen by middle and large size company/enterprise businesses.

An important benefit of private cloud is that single organization that has access to resources, can allocate however it is necessarily and this operation could lead to an efficiency of cost for energy and for operation entire cloud. In addition, flexibility is a big plus. Every private organization can choose own technologies, architecture and software such that can secure and isolate apps and use cases.

A third model, the **hybrid cloud** is maintained by both internal and external providers. In effect, a hybrid cloud is a combination of public and private cloud services, with orchestration and connectivity between the two [17]. This model is attractive because it enables organizations to take the benefits of the public cloud, while maintaining their own private cloud for sensitive, critical or highly regulated data and applications.

In Table 5 are summarized the main characteristics of a cloud infrastructure.

Table 5 Main characteristics of cloud infrastructure

Type	Flexibility	Capacity	Cost	Security	Payment	O&M	Provisioning	Site
Private	Yes	Limited	Initially high	High	Initial payment + support	Yes	Instant (after cloud is up)	Yes
Public	Yes	High	Low (limited usage)	Low	Pay-as-you-use	No	Instant	No

2.2.1 Private Cloud for Smart City

To start development and implementation of entire Enterprise Cloud Infrastructure dedicated to the Smart City use case and all network components, it is necessarily to describe the needs and all blocks that make up entire service. All elements and components from private cloud network will be described in the following words and this environment will be developed in Orange Romania laboratory on physical infrastructure (servers and connectivity). A high level view of Smart City Apps could be observed from Figure 13.

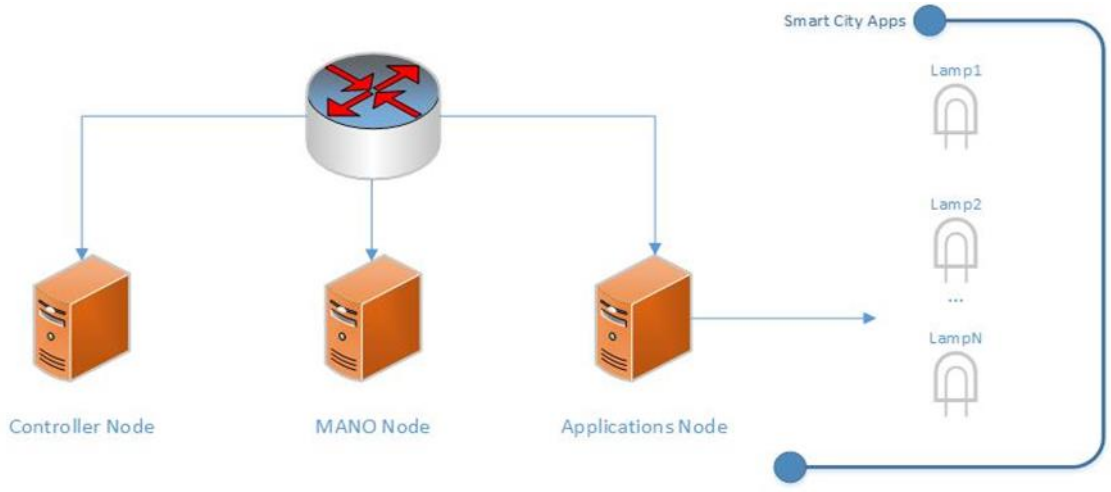

Figure 13 High level view of Smart City apps

Using this entire hardware infrastructure, we will instantiate a Private Cloud Infrastructure based on OpenStack open source software collections.

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a DC, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface [18]. In Figure 14 it is presented a high level design for Openstack [18].

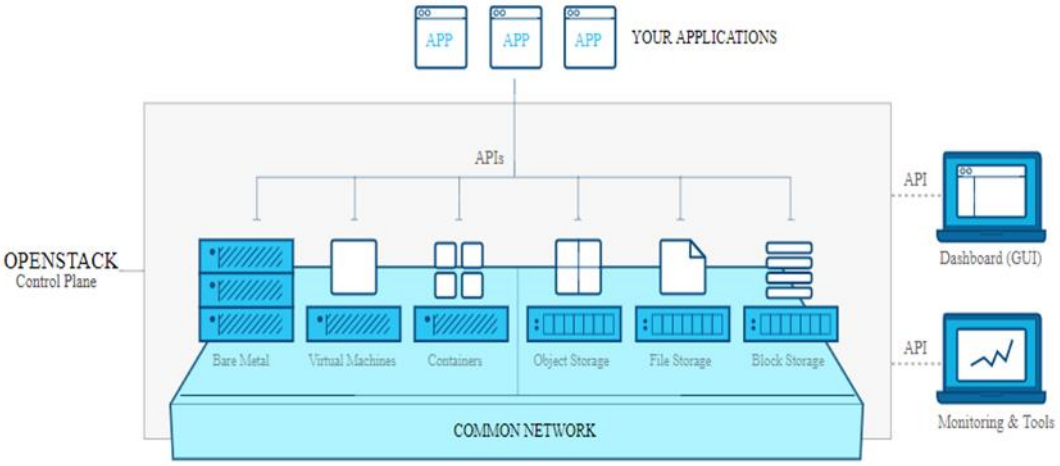

Figure 14 High level design for Openstack [18]

OpenStack is not a hypervisor, but it is designed to work with a number of different hypervisors. Users have the option of deploying a hypervisor on the machine or an OS that has a built-in hypervisor, like Linux Kernel-based Virtual Machine (KVM). With the OpenStack bare-metal provisioning, users can push VMs onto bare-metal servers.

There are many OpenStack components; some of them that will be used in this project are listed below.

Nova (compute) includes the controller and compute nodes. These get VM images from OpenStack's image service and after that it will create a VM on the specific server.

Neutron (networking) creates virtual networks and network interfaces, and attaches to many proprietary vendor networking products.

Keystone (identity storage) grants users and processes access to different OpenStack tools based on an authentication token that keystone generates [19].

Besides controller node, MANO and Apps servers will use over bare metal, KVM as hypervisor. KVM is open source software that can enable a full virtualization solution. Using it, we can deploy and manage multiple VMs with different OSs images (Linux-based, Windows, etc.). KVM Virtualization Environment could be observed from the Figure 15.

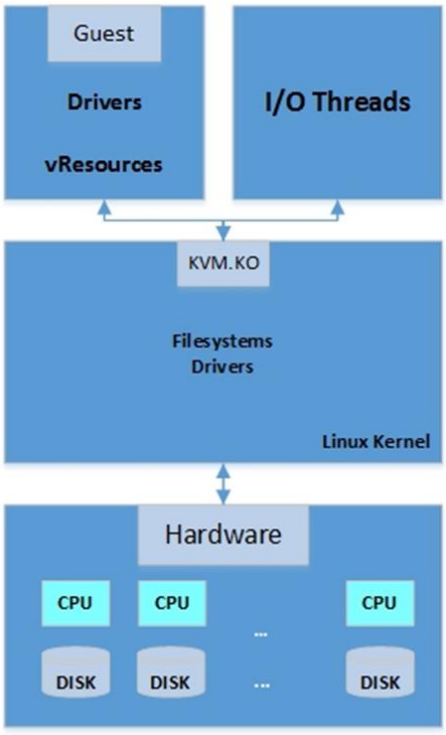

Figure 15 KVM Virtualization Environment

KVM converts Linux into a bare-metal hypervisor. All hypervisors need some operating system-level components—such as a memory manager, process scheduler, input/output (I/O) stack, drivers, security manager, a network interface to run VMs. KVM has all these components due the fact that it's part of the Linux kernel. Every VM is implemented as a regular Linux process, scheduled by the standard Linux scheduler, with dedicated virtual hardware like a network card, graphics adapter, Central Processing Unit (CPU), memory and disks [20].

Above, in section 2.3.1., is presented entire Enterprise Segment with all VMs created on this infrastructure, there functions and how can be them connected to build upon a solid infrastructure for Smart City that can serve one or more cities.

Putting together all this elements it can be developed a sustainable environment for the Smart City use case with smart lighting system and also available to be customizable for more solutions.

3 Services plane: Enterprise Services

3.1 SliceNet vision about 5G Connected Services

NSs can be provisioned on demand and deployed over a virtualized infrastructure, allowing the transition from nowadays vertical dedicated networks to share and customizable horizontal networks. Slicing can maximize resource-sharing, e.g. via multiplexing, both across and within domains. The purpose of a SliceNet slice, in particular, is for enterprises to be able to consume logical (sub-) networks, hence allowing having no dedicated network infrastructures in order to support enterprise environments.

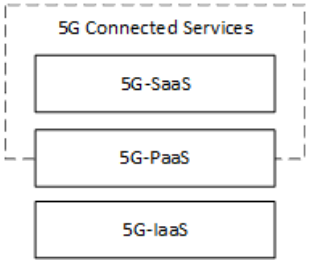

Figure 16 5G-Slicing offered as a SaaS over PaaS over IaaS

5G-IaaS provides a programmable infrastructure; 5G-PaaS extends IaaS with support for control, orchestration and virtualization; finally, 5G Software as a Service (SaaS) allows the connected enterprise services to consume and control the underlying customized network via P&P control apps.

Figure 16 shows how the Service Plane is organised. A 5G Connected Service is either a NS or a value-added service such as a video optimizer for an enterprise vertical like Netflix or a QoS optimizer in an e-health use case. The 5G-IaaS approach presented on top hierarchical level of Figure 16 provides a programmable infrastructure (e.g., software-defined radio or an x86-based infrastructure) and hosts the RAN service over the top, with the latter being either commercial (e.g., Amarisoft) or open-source (e.g., OpenAirInterface (OAI)). From a provider's point of view, 5G-IaaS is a function that can be logically sliced and provided to each slice as its own logical function. However, management remains the same, including the management lifecycle. From the point of view of a slice, IaaS is perceived as a dedicated 5G control function which is deployed as if there is no slicing.

Whereas the 5G-IaaS level is used as a common infrastructure for many slices, a 5G-PaaS level sitting on top of the 5G-IaaS allows not only the creation of a slice based on SLA/QoS requirements, but also the customization of the network functions as per enterprise needs. Essentially, a 5G-PaaS level extends the underlying IaaS with support for control, orchestration and virtualization via providing open APIs for a slice-friendly development environment to enable an ecosystem for network applications. For instance, FlexRAN and RAN runtime belong to this category.

A 5G SaaS level allows the connected enterprise services to consume and control the underlying customized network comprised by both the 5G-PaaS and the 5G-IaaS. 5G-SaaS embodies customized services through P&P control apps that are operating on the top of the customized network such a video optimization, analytics, etc. At this level, the provider provides an E2E service and not a VNF, which is perceived by slices more like a (micro-) service mesh composition with the internals of the 5G NSS being opaque. Enterprise services

can cognitively consume services like Radio Resource Management (RRM) and user handover to provide the control logics through control apps. The development of these apps rely on the provided Software Development Kit (SDK) and API to enable new control and management services in an agile and flexible manner. Note, however, that 5G connected apps are not only limited to the enterprise control apps, but also they can interact via app-to-app communication with other control apps from other domains.

Finally, that enterprise services can also span to the 5G-PaaS level. The latter level establishes both shared and dedicated CP and/or User Plane (UP) functionalities such as mobile handover algorithms or RRM policies. This way an enterprise service slice perceives 5G-PaaS as a new control function that supports slightly different APIs or, alternatively, management can be similar to standard VNF with some options disabled.

3.2 Smart City Applications

As mentioned in the section above, all apps from the use case are using to export data, different APIs an SDK and other applications to process and gather all information in one place.

On the third physical server, from Orange Romania laboratory will be instantiate 3 VMs that will take the role of apps and will be the core of the data processing and storage for all information that can be obtained from Smart City infrastructure of sensors and actuators. Besides these 3 virtual servers, we can, on demand, instantiate new VMs for another third party apps or data provider.

On the first VM, will be deployed all the components that cover IoT platform functionalities and on the other two will be deployed one relational database and one no Structured Query Language (SQL) database where can be stored all kind of data gathered from city solutions. Both databases represent key points in entire architecture and help the IoT platform to be more efficient and to process and stored data in correct manner for every type of client or service.

Types of apps from enterprise segment:

1. Thingsboard.IO – open source IoT platform
2. PostreSQL – open source relational SQL database
3. Cassandra NoSQL – open source non-relational database

The entire concept of Smart City can be understood as all devices deployed in city areas that provides data and communication technologies used to transport all data to cloud or to any platform that can analyse it, process it and integrate in visualizations or any form that can offer key information in running the cities towards an economical, secure, efficient and green environment for all citizens. In this picture, the Smart City Apps use all components of enterprise infrastructure to serve all kind of use cases. For Orange Romania, the Smart City use case is Smart Lighting that forms apps layer of city and gather all information to offer intelligence such that a city can have an economical and efficiency lighting system. An endpoint as a web portal is offered to end user to can monitor and control the entire system.

Forwards, it will be described those 3 apps listed before.

IoT platform needs to be compliant, for Smart City use cases, with some technical capabilities listed below. Besides that, will have to connect IoT devices via telecom networks

(with/without SIM cards – depends on use case) with reduced power and act like an IaaS that provides hosting space and processing power for all services.

IoT Platform should integrate a set of tools to facilitate the interconnection between devices and business apps, with the following components:

- Connectivity Interfaces (public and private) to collect data send command or notification from/to IoT/ Machine to Machine (M2M) devices;
- Device Management (supervision, configuration, resources, firmware, etc.);
- A specific and configurable policy for message and commands routing between devices and services;
- Data management system and data storage with different storage periods and variable access, processing speeds;
- Web portal for user's management and visualization tools;

IoT Platform architecture, presented in Figure 17, is composed by following complementary levels:

- Connectivity: manages the communications with the client devices and apps;
- Bus: a set of message-oriented programs allowing exchanges between all client software modules;
- A set of rules or a bus that can allow messages/commands exchange in asynchronous manner;
- Service: various modules supporting the high level functions (device management, data processing and storage, etc.).

In Figure 17 it is proposed architecture of an IoT platform, taking into account all the aspects listed above.

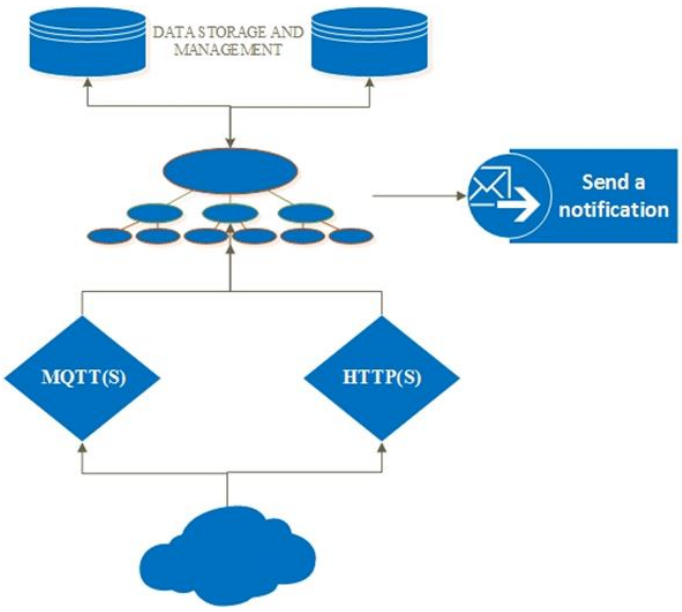

Figure 17 Proposed architecture for IoT platform

In the following paragraphs it will be described in more details every component of the platform.

a) Connectivity Interfaces:

IoT Platform must allow a set of standard and unified public interfaces allowing connecting any programmable devices, gateway or functional IoT backend. The public interfaces must support the following protocols:

- Message Queuing Telemetry Transport (Secure) -MQTT(S) interface to communicate and also over websocket interfaces;
- Hypertext Transfer Protocol Secure (HTTPS) interface;
- Optional other protocols useful in IoT segment, such as Constrained Application Protocol (CoAP).

It handles communications from specific families of devices with defined protocol (over IP) and translates them as standardized messages available on platform.

b) Device management:

IoT Platform must offer various functions dedicated to all kind of devices:

- Supervise devices connection and disconnection to/from the SaaS;
- Manage devices configuration parameters;
- Send command to devices and monitor the status of these commands.

c) Event processing:

Event processing service is aimed at detecting notable single event from the flow of data messages. Based on processing rules that you define, it generates fired events that your business apps can consume to initiate downstream action(s) like alarming, execute a business process, etc.

d) Data management:

IoT Platform allows storing the collected data from any connectivity interfaces. These data could be then retrieved by using Hypertext Transfer Protocol (HTTP) interface. A full-text search engine must be provided in order to analyse the data stored. Data management must be based on:

- The store service which is aimed to store data messages from IoT things (devices, gateway, IoT app collecting data, etc.) as time-series data streams;
- The search service.

The data messages sent to the IoT platform can be encoded in a customer specific format. For instance, the payload may be a string containing a hexadecimal value or a csv value. The data decoding feature enables you to provision your own decoding grammar. On receiving the encoded message, the IoT Platform will use the grammar to decode the payload into plain text JavaScript Object Notation (JSON) fields and record the JSON in the store service.

e) Security: API keys are used to control the access to the SaaS for devices/app and users to authenticate.

f) Visualization tool: Provide a simple visualization tool web-based for friendly usage.

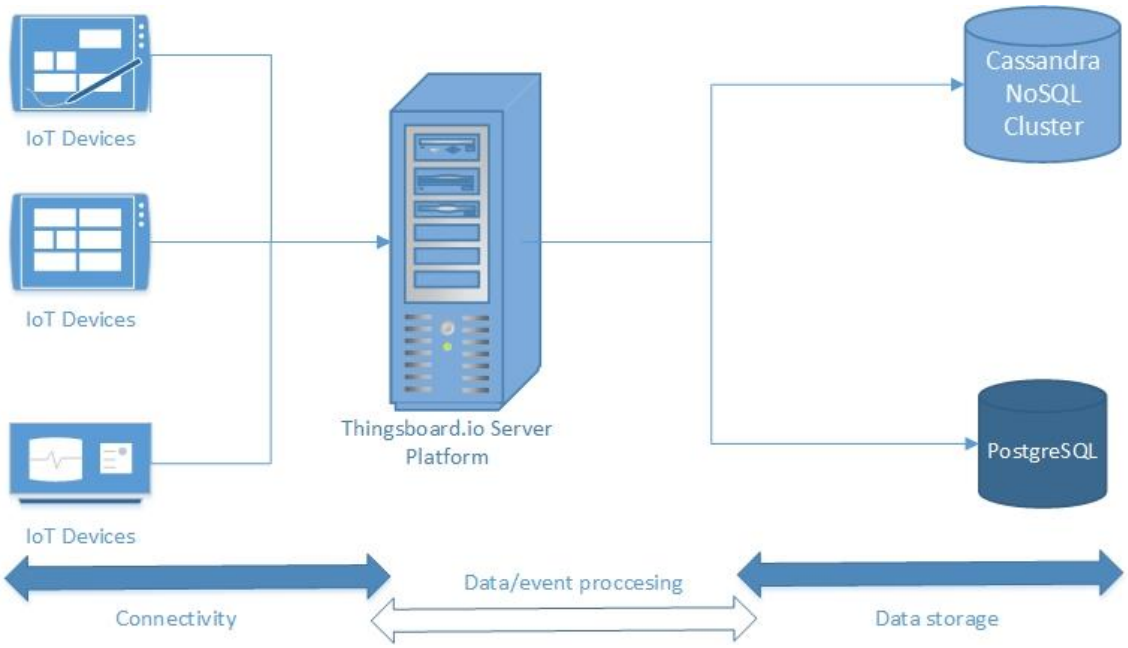

Figure 18 IoT Platform integration

IoT platform need to be connected with two types of databases (relational and non-relational) such that can store in efficient way all messages and documents. All these elements figured in Figure 18 are part of apps ecosystem and will be deployed, configured and tested in engineering laboratory on virtual infrastructure build above a physical server.

Every function and functional mode will be presented in chapter 8, where entire Enterprise Orange prototype will be described.

3.3 Smart City Services

The apps depicted in section 3.2 will be utilized to run the services for Smart City use case in Alba Iulia. For beginning, as it was also described in D2.1 [4] we intend to implement the SmaLi-5G SLICENET use case. After implementing those services it is expected to extend this implementation to other services that could be optimum offered by a 5G Smart City, like: metering solutions (water, gas), environmental monitoring, (pollution, temperature, humidity, noise), connected parking, real-time traffic information and control, connected buildings, smart home, connected household appliances and public safety alerts for improved emergency response times. The target, in the future, is to transform Alba Iulia in a Smart City powered by 5G services.

Services delivered by smart cities should be easy to use, responsive, efficient, open and environmental sustainable. Citizens expect high quality public services meant to improve their daily quality of life. Nowadays cities are under pressure to enhance urban services management, provide more efficient infrastructures and services, often for less cost.

SmaLi-5G will be considered in the scope of the 5G Massive machine-type communication category where the challenge is to accommodate the massive number of connected actuators/controllers. Another service requirement to be met by the SmaLi-5G use case is to assure ultra-high network reliability and availability, while low-power, context awareness and location awareness requirements for managing the connected actuators/controllers over the access and transport layers can further improve the solution cost efficiency. This

will be especially important during the daytime when the smart streets lighting poles infrastructure is supposed to remain powered to facilitate other city services (e.g. public safety surveillance, air quality monitoring, public Wi-Fi hotspots, advertising) [4]. The goal of the use case is to implement smart lighting services within the Smart City platform, over a 5G enabled architecture, with slicing support, from IoT devices to the smart lighting cloud app.

The knowhow of a sensor network is mainly reflected in provision of real time information and in the fact that the realtime sensor data might be integrated with other type of information such as environmental modelling and control. The increased penetrations of fixed and wireless networks and the expected introduction of 5G allow that such ecosystems to be connected to distributed processing data centres. The 5G Smart City enables the sensors to feel the city systems, and process the sensing information by cloud native apps while integrating components from the cyber space and internet of things.

Smart city services and apps are focusing on how to shape future 5G based services and apps from a smart city perspective. The design, implementation and approval of innovative 5G internet based services and apps are mandatory prerequisites considering the challenges of advanced connected cities. The generation of data is not restricted to a particular location, and the resulting products are typically delivered through the network. Smart city services are enabled by services oriented enterprise architecture including web services and cloud native apps.

On this deliverable it will be described and prototyped a 5G Connected Virtualized Enterprise Infrastructure that could support the Smart Lighting service for the Smart City platform following by the adoption of a wide range of services.

4 Control Plane for Enterprise

4.1 SliceNet vision about Control Plane for Enterprise

A slice QoE Optimizer module in the CP regards the functionality of a per-slice optimisation actuation framework with the scope of QoE guarantees. The functionality offers optimisation algorithms tailored for the infrastructure resources and the network functions to be deployed, re-configured or released after the characteristics of the NSI, the current state and utilization of the underlying physical and virtual data plane. Essentially, the QoE Optimizer is a dedicated control function within the CP, adapting the function split of 4G/5G architectures that separate control functions into either common or dedicated. Common control functions are transversal for all NSI intra- and inter-domain slicing functions (discussed later in this section). Dedicated control functions are on the other, are specifically tailored for every NSI, such as the P&P control (discussed in the next paragraph) and the Slice QoE Optimizer, providing enterprises with the flexibility of customized functionality tailored slice-specific QoE requirements.

In general, P&P control functionalities provide on-demand composition, integration and abstraction of NSI control and management to the enterprise slice consumers. SliceNet P&P regards per-slice sets of control functions consumed and augmented by slice consumers, hence being tailored to the enterprise needs. In order for the enterprise to consume services, there is service registry, an inventory and a discovery module to support the necessary control and management functions for that purpose. These control and management functions include (i) a very basic monitoring option (e.g., for monitoring performance, resource availability, etc.), (ii) a limited control option that allows the enterprise as a slice consumer to gain access to a limited set of SDN and NFV control and configuration primitives, and (iii) an extended control option allowing to access the slice instance LCM.

Last, QoS assurance is an important factor for the enterprise. For this purpose, handling of virtual and physical network resources can support multi-tenancy and resource isolation for the sake of performance or security among different instances of slices and services consumed by the enterprise. Therefore, the enterprise has to be offered with different levels of resource isolation and sharing so as to customize the CP and to increase the resource utilization.

4.2 Smart City Control Plane

SliceNet CP architecture, as described in D2.3 [5] is composed by different domains and covers different technologies. There are identified some specific SliceNet functions, as P&P, QoE Optimizer, service chaining. It is also expected to support the QoS and the SLA for each sliceable vertical apps, in this case the Smart City lighting apps. The approach is to build and manage a single domain inside the Network Service Provider (NSP) that also plays the role of Digital Service Provider (DSP). The Enterprise will be part of the E2E smart lighting slice, within the same management domain, that contains the RAN and Core part, with the specificity that these are not the subject of the task, the slice being composed by the sub-slices. The Smart City use case is considered to be treated as a “common slice” with respect of the slice requirements, that reflect to both Physical Network Functions (PNFs) and VNFs functions, including also the CP defined actuators.

The SliceNet CP high level architecture includes the Service-Based Architecture (SBA) approaches and has the origins of implementation based on ETSI MANO, with main focus on the VIM, integration within the Enterprise, acting as a resource orchestrator, can be observed from Figure 19 [5].

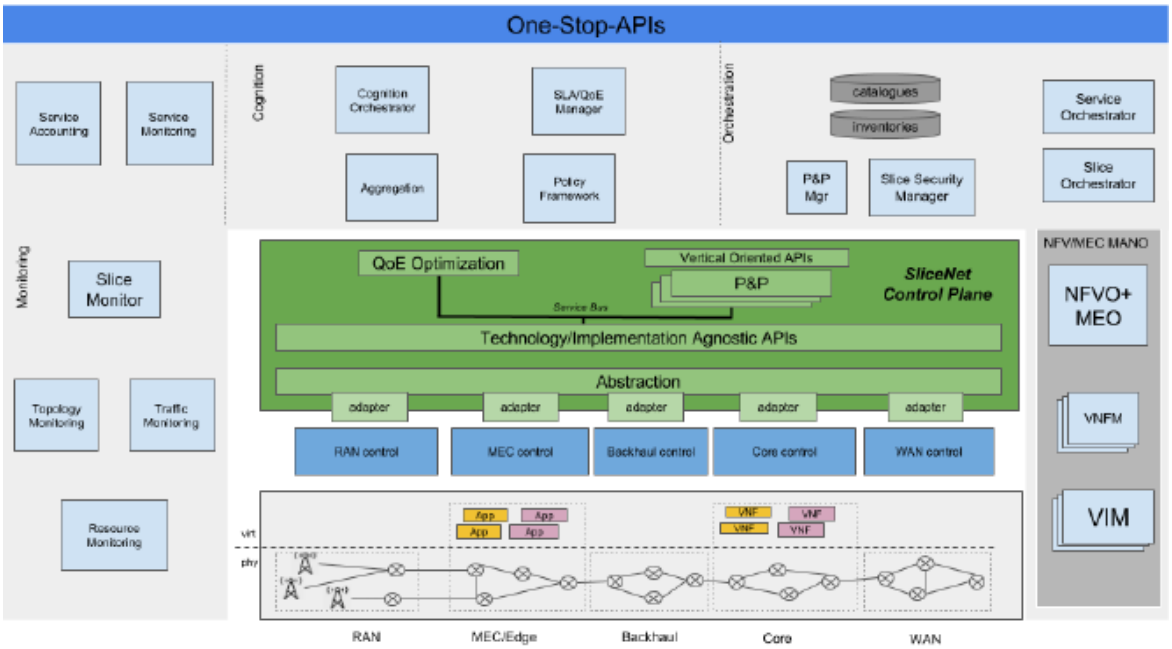

Figure 19 SliceNet CP high level view

4.2.1 VIM

VIM is managing the NFVI and serves as a conduit for control path interaction between VNF and NFVI. The main task for VIM is to inventories, provisions, de-provisions and manages virtual compute, storage and networking while also communicating with the underlying physical resources. The VIM is responsible for operational aspects such as logs, metric, alerts, root cause analysis, policy enforcement, service assurance etc. In Figure 20 is shown the interaction responsibilities of the VIM with the management and orchestration functional block, and SDN controller.

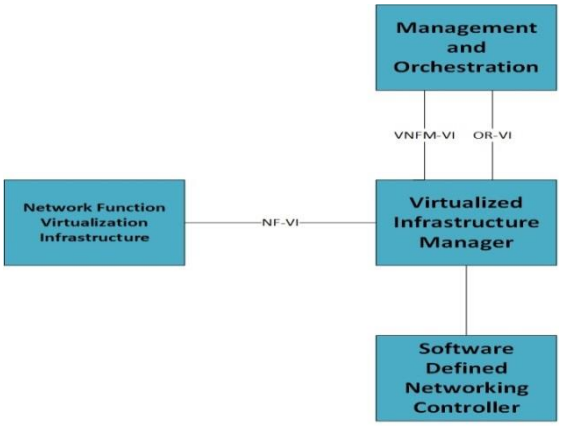

Figure 20 Interaction between VIM and SDN Controller

VIM comes in the form of complete software stacks. On the market right now there are only two major software VIM stacks prevalent in NFV: OpenStack and VMware.

To achieve the goal of building the Enterprise platform, we choose a more open source approach, and we get to handle on OpenStack, which is growing in popularity among the NFV deployment at carriers worldwide.

The most important aspect of OpenStack pertaining to its usage as a private cloud platform is the tenant model. Every virtual or physical object governed by the OpenStack system exists within a private space referred to as a tenant or project [21].

The main purpose is to use one of the computing nodes as a Management and Orchestration where we intend to deploy alongside Cross Plane Orchestrator, VNFM and VIM also a Tacker server. The main purpose of the Tacker server deployment in this deliverable infrastructure is to perform the scaling procedures and optimize all the physical pool resources of the server, with future perspective in case of managing multiple OpenStack sites without having the need to deploy Tacker server on each of these sites.

After installation of OpenStack it needs to initialize the flavors that the platform will support. Most Tacker sample Topology and Orchestration Specification for Cloud Applications (TOSCA) templates will ask Tacker to create Flavor on demand [22]. If not, the specified flavor in templates must exist in OpenStack. Tacker repository's sample TOSCA templates are referring to cirros image named "cirros-0.3.5-x86_64-disk", so this image should be uploaded into OpenStack before Tacker uses it.

TOSCA represents a specification that aims to standardize how it describe software apps and everything that is required for them to run inside a cloud based app. TOSCA provides a way to describe not only an app, but also its dependencies and supporting cloud infrastructure. TOSCA contains two building blocks references: nodes and relationships.

Management and Orchestration descriptors are detailed in the table attached in the Annex A. For example, to deploy a real VDU, we'll use the following TOSCA template: tosca.nodes.nfv.VDU.Tacker [23], which contains the information from the Annex A.

TOSCA parser will be updated to handle VNFD, Virtual Link Descriptor (VLD) and Connection Point for NSD and is detailed in Annex A. The process of parsing is represented by analysing a string input made out of a sequence of tokens to determine its grammatical structure with respect to a given formal grammar. The parser then builds a data structure based on the tokens. This data structure can then be used by a compiler, interpreter or translator to create an executable program or library.

All the signalling represented in the Figure 21 is made throw specific points, called connection points that will be exposed as part of VNF. An NSD needs to be created to instantiate NSs. The method of creating NSD follows the TOSCA template scheme presented in the Annex A.

In Figure 21 it can be noticed three parts of communication flow, first one is established between the EM and VNFM establishing a VNF LCM scheme, the second one is establishing the permissions from the orchestrator point and the last communication flow is invoking the resource management operations handled by the VIM.

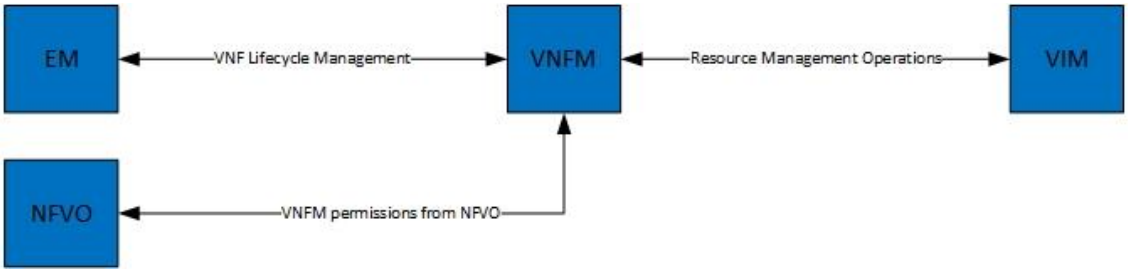

Figure 21 Simplified flow for VNF related resource management

As a conclusion of this chapter, VIM is the software resource block responsible for ensuring that physical and virtual resources work smoothly in any condition. Compared with the more traditional Operating System, VIM comes with extra features of collecting resources logs from many other machines at the same time.

4.2.2 SDN Controller

SDN controller represents the control point of the SDN network and its scope is to control the flow rules of the overlay forwarding path entities (vSwitches or vRouters) and might also have the capability to manage the physical network equipment.

Under the perimeter of Enterprise cloud, the SDN controller is responsible for performing network functions and to provide on-demand connectivity for apps and services while keeping isolation between different tenants.

In the Enterprise cloud implementation, Neutron Openstack service is responsible for enabling networking and controlling the vSwitches embedded in the compute node as depicted in Figure 22.

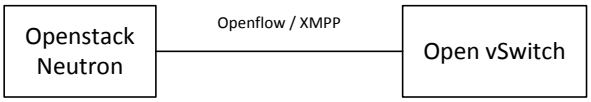

Figure 22 Neutron control to Open vSwitch

In order to provide network dynamicity for discovery and allocation of virtual networks and compute resources, the SDN controller is bound to the VIM. This interaction is based on Modular Layer 2 (ML2) plugin and RPC service for bidirectional agent communication.

As depicted in Figure 23, Neutron agents which run in Enterprise infrastructure are split as follows:

- On Controller Node will run neutron-server, neutron-dhcp-agent, neutron-l3-agent and neutron-plugin-agent;
- On Compute Node will run neutron-plugin-agent (neutron-openswitch-agent).

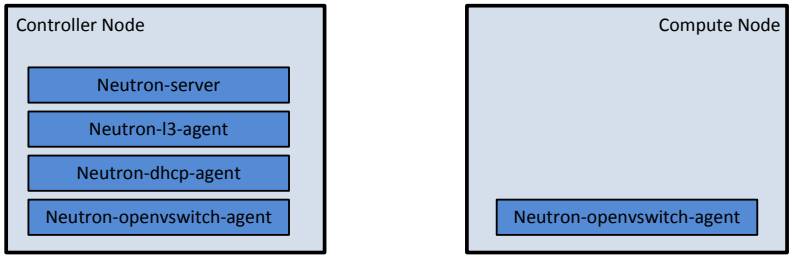

Figure 23 Neutron agents

Table 6 specifies the scope of each Neutron agent in order to allow the provisioning of virtual network interfaces and resources.

Table 6 Neutron agents usage

Agent	Usage
Neutron-server	Provides REST API exposure
Neutron-l3-agent	Provides Layer 3 (L3) routing
Neutron-dhcp-agent	Provides Dynamic Host Configuration Protocol (DHCP) services
Neutron-openvswitch-agent	Provision network resources

Neutron will help on VM booting process by wiring the VM port and providing IP address to it using DHCP agent. Neutron will also help when communication between different VMs or to external physical world is needed.

5 Management Plane for Enterprise

5.1 SliceNet vision about Management Plane for Enterprise

The SliceNet Management Plane contains four internal sub-planes; namely, the (i) Information, (ii) Orchestration, (iii) Monitoring and (vi) Cognition. The Information sub-plane is logically centralized, so as to avoid replication and incoherent information (service/slice templates and registries). It acts as a central point of interaction all SliceNet (sub) planes. Then, the Orchestration is a sub-plane for ensuring that actions take place in a correct order to deploy and/or configure the logical abstractions in SliceNet, namely the resources, slices and services. The Monitoring sub-plane collects filters and enriches network information such as events/alarms or counters from the running network resources, slices and services. Last, the Cognition sub-plane enables proactive performance and fault management for the network slice and services. It is fed with information from the monitoring subplane, which it processes to allow automation policy-based procedures to run.

Regarding orchestration, in particular, for coordinating the needed actions in order to deploy and configure a service or a slice and their (possibly cross-domain) resources, there is a series of six components discussed next. First off, a (i) Service Orchestrator coordinates the uninterrupted service delivery of the slice/service to the individual customers in the sliced multi-tenant environment, exposing the necessary interfaces to customers, who can consult available services and request new ones, or update and monitor the existing ones. Slices in SliceNet are composed of isolated virtual infrastructures. It is at this point where a (ii) Slice Orchestrator component is needed to coordinate one or more technology-independent virtualization domains, so as to build isolated and unified slices, as well as to deploy and manage the slices at runtime. The third component, a (iii) Resource Orchestrator, does the lowest level of orchestration within the Cross-Plane Orchestration, providing the building blocks (e.g. VNFs) for the previously discussed Slice Orchestration. It includes ETSI MANO management components and SDN components for the apps LCM.

In addition, there is a special (iv) P&P Manager that does the P&P control, providing management primitives and APIs that allow slice consumers to have direct run-time control over their slice instances. In a nutshell, the P&P Manager offers all needed customization and configuration interfaces required to expose to the verticals. Related to the former, a (v) Infrastructure and Resource Manager offers a flexible RAN slicing runtime environment [24] for multiple virtualized RAN instances. It enables slice owners and infrastructure providers to manage slices, RAN and Core Network (CN) lifecycles, and to enforce custom control (e.g., for handover decisions). As a result, different levels of sharing and isolation across resources and network functions are feasible with fine-granularity on a per slice basis. Last, there is a (vi) Slice Security Manager component, which triggers the deployment of de/encryption functions and interacts with other components in order to share de/encryption keys for the E2E encrypted network paths within a slice.

5.2 Smart City Management Plane

Regarding Management Plane of a Smart City Enterprise architecture (in line with deliverable D2.4. [6]) from a management perspective, 3GPP defines a Network Slice Management Function (NSMF) as responsible for the LCM of network slice instances, linked and interconnected to Network Slice Subnet Management Function (NSSMF) for LCM of

subnets instances. A potential relation between 3GPP and NFV functions can be assured such that SliceNet Management Plane can be defined and development in context of Smart City Use Case.

All management and orchestration blocks from a network slice (in this case Smart Lighting slice) should include and operate, such that can facilitate the proper functioning of the service, the following functions: provisioning, optimization and performance, monitoring and maintain entire NSI in parameters in which it was designed.

The approached scenario, as also described in D2.4 [6] for the Smart City use-case is based on a single management domain, is a combined DSP and NSP function, as the both roles are played by the operator, including in this case, under the same umbrella, the Enterprise implementation.

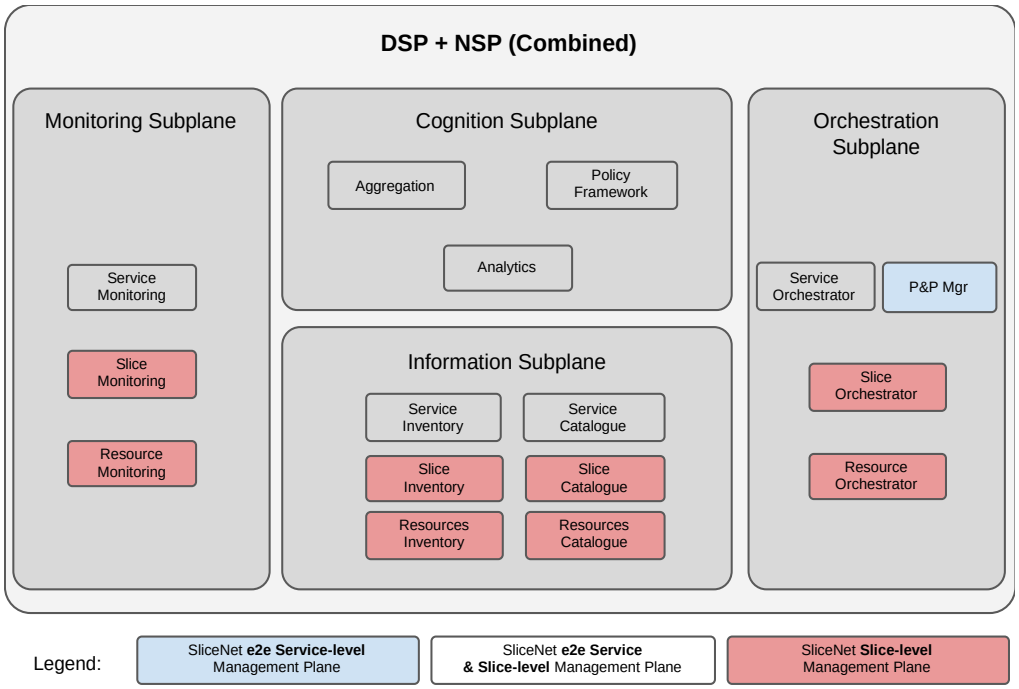

Figure 24 Management Architecture Components – DSP & NSP Combined Perspective [6]

In the following paragraphs, will be described main management blocks and functions regarding management of a slice dedicated to a smart city solution or service which is offered, design, provisioned, configured, monitored and released.

5.2.1 Monitoring

Starting from the architecture Figure 24, the Monitoring Sub-Plane is the functional block who is responsible to perform the architecture sensing functionalities, collection, filter and enrichment of counters, events and alarms retrieved from the network resources that compute the slice delivered to each vertical. The functional block, Monitoring Sub-Plane, from the Figure 24 is responsible of collecting, filtering and enriching network information that will be use to understand the performance and usage of the network slice and services. It gathers information, such as counters, events and/or alarms form the running physical/ virtual network resources, slices and services.

Monitoring scope is to provide information to the intelligence procedures deployed in the Cognition Sub-Plane functional block, as well to allow the network operator administrator and/or DSP/Digital Selective Calling (DSC) to view the slice and service status.

Aspects that are expected to be monitored will be split as follows:

1. **Host-Based Metrics:**
   The information here extracted are the bottom of the hierarchy of primitive metrics, represents the host-based indicators. All of those metrics collected from this layer of the hierarchy will provide the perspective for evaluating the health or performance of an individual machine, disregarding for the upper layers in the hierarchy. These are mainly comprised of usage or performance of the operating system on the bare metal or hardware performance, like:

   - CPU
   - Memory
   - Disk Space
   - Processes

2. **Application Metrics:**
   The next layer in the hierarchy of metrics is application metrics. Those are concerned with units of processing or work that depend on the host-level resources, like services or apps. The specific types of metrics at this layer depend on what the service is providing, what dependencies it has, and what other components it interacts with. In this deliverable slice network this can be consisting of a VM who is performing the app and it desired to monitor the metrics at the application layer considering the health, performance, or load of an app. So, in this case we should consider the following metrics:

   - Error and success rate
   - Service failures and restart of the service
   - Performance and latency of responses
   - Resource usage

3. **Network and Connectivity Metrics:**
   When it is necessary to deal with horizontally scaled infrastructure (like in enterprise development), another layer of infrastructure is needed, so will be possible to monitor the higher level of extrapolation of application and server metrics, but the resources in this case are homogeneous servers instead of machine-level components. This layer of metrics summarizes the health of collections of servers using the following metrics:

   - Pooled resource usage
   - Scaling adjustment indicators
   - Degraded instances

4. **External Dependency Metrics:**
   Considering the distributed system that is necessary to develop and the fact that most of the slice will communicate throw APIs. Tracking these within your own system – as well as your actual interactions with the service – can help to identify problems with your provider that may affect your operations. Some of the items that might be applicable to track at this layer of hierarchy are the following metrics:

- Service status and availability
- Success and error rates
- Run rate and operational costs
- Resource exhaustion

The NFV inherent multi-layer infrastructure requires an evolved monitoring approach. Which can rely on the real-time analytics capability, i.e. it must be able to correlate data from all the layers involved to understand the root-cause analysis in case of a problem occurring in the top layers correlating data from top and from bottom layer. And E2E active service monitoring, i.e. it should possibly use active monitoring by continuous testing for end user service layer, also relying on service activation automated testing systems used during automated deployment by Network Functions Virtualization Orchestrator (NFVO).

The monitoring sub-plane at the management plane constantly monitors and feed the slice KPIs to the LDE. The LDE checks that the level of packet loss is higher than the desired level, checks what is the threshold for TCP SYN retries per minute and it has to identify if the device has reached maximum throughput.

As a conclusion of this subchapter, it will be a good practice to expose all the monitoring metrics throw APIs which are categorized as presented above to further deployments in order to obtain a more deep perspective about all the performance of the system and service. All those monitoring metrics exposed throw APIs could be forwarded also to the VNFM in order to concentrate all the metric information's into one API to the Cross Plane Orchestrator in order to take the decision to optimize all the system parameters so that the services provided offer a high yield.

5.2.2 Life-Cycle Management

The LCM proposed is based on ETSI NFV MANO [25] and ETSI TS 128 526 [26] related to VNF creation instantiation, configuration, scaling, auto-scaling, on-boarding and termination. The LCM include also the NS instance procedures for instantiation, termination, scaling, modification and lifecycle changes

In this case the main functions of LCM are:

- Create & Instantiate VNFs
- Operate VNFs
- VNFs chaining
- Update VNFs
- Terminate VNFs
- Delete VNFs

The LCM module is used to provide agility to enterprise infrastructure, facilitating the spawning of VMs, scaling, healing, modification or termination of them. The LCM module is based on descriptors of the targeted infrastructure, such as templates or blueprints. One template describes the list of virtual nodes which need to be instantiated, their flavours, configurations, and policies and is written in TOSCA.

The proposed module for LCM of enterprise services is Murano, Openstack project. Murano will provide a catalogue of apps and services, including the rules and configurations of them contained in templates. Network connectivity between virtual instances can be also described in NS templates. LCM module performs basic functions such as commissioning,

scaling or terminating of NSs and provides a framework for implementing advanced functions such as healing, patching or upgrades.

LCM module will interact with other components of enterprise infrastructure as depicted in Figure 25.

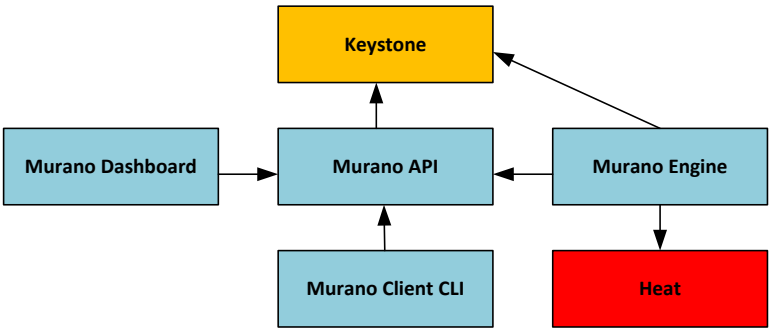

Figure 25 Murano components and interaction with other services

As a conclusion, the advantages of using a LCM module are:

- Fast deployment of services compared to manual methods;
- Avoid human errors in manual methods;
- Scaling capabilities to meet demand.

5.2.3 Configuration Management

Configuration Management is the part of the management plane that includes mechanism for: design, deployment, provisioning, configuration (input for CP), monitoring, de-provisioning of resources, including the slice and the service.

The configuration management is intended to support the enterprise network slice (through OSA) ordering, by interacting with the CP, and includes functions such as configuration, update, upgrade/rollback, resource configuration.

- **Configuring VNF/PNF/NF | Service configuration**

Proper configuration input information must be provided to the CP. The correct amount of virtual resources (e.g.: vCPU, vRAM, Virtual Storage (vStorage)), communication protocol information (e.g.: MQTT, CoAP, HTTP, etc.), associated technology data particularities (e.g.: OS type, storage file system type, associated middleware and plug-ins), associated VNF/PNF (vFirewall, vDHCP server, vIDS, etc.) must be taken into account, stored into templates and deployed accordingly.

Due to change of traffic or load patterns, the slice, comprised of VM, VNF and PNF elements, allocated as resources, can be scaled up/down, as a resource optimization process. A new service instance may require reconfiguration on an existing VNF or new VNFs. If the service evolves, new VNFs or resource allocation blueprints need to be defined.

VNF/PNF/NF configuration can be either manually triggered from the Horizon Web Interface or by active configuration management templates stored in HOT (native to Heat) or TOSCA templates (accessible to Heat via heat-translator).

The below steps are to be followed for enabling flexible, auto-scaling apps:

1. Define a HOT / TOSCA template (TOSCA templates can be translated in Heat with a special plugin), specifying: min-max virtual instances, compute, storage and networking resources to be allocated and a key pair: control access to instances.
2. Call Heat with a cloud app HOT / TOSCA template.
3. Heat creates the cloud app stack with scale group and scale policy.
4. Heat creates alarm definitions and scaling notifications in Ceilometer.
5. Ceilometer starts to monitor the scaling group elements (auto-scaling setup is done).
6. When the allocated resource pool reaches the alarm thresholds, Ceilometer detects and generates an alarm.
7. Ceilometer signals the app (VM/VNF) through Heat.
8. Heat gives the scale up /scale down command to the app, which can have two outcome types:
   a. allocating /removing additional VMs, VNFs, or adding PNFs to the running app;
   b. creates another app instance, having the same VM/VNF/PNF resources (vCPU, VRAM, vStorage, number of IPs, etc.) as the initial one.

If properly configured, once set up and instantiated, an app can automatically scale by following steps 6 – 8.

These steps are depicted in the below diagram:

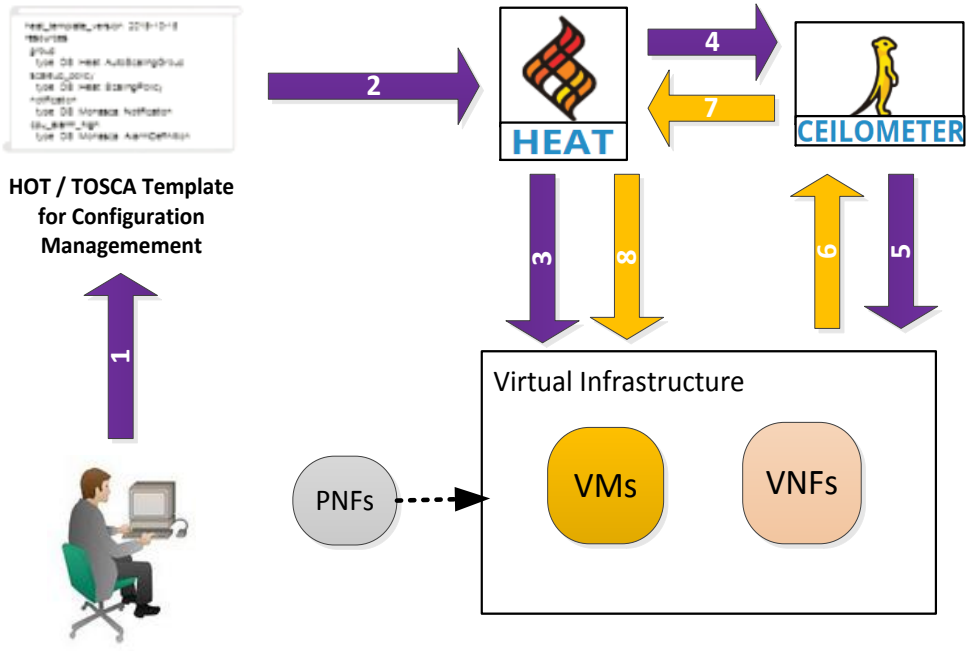

Figure 26 Configuringuration Management for Manual and Auto Scaling

- **Update VNF/PNF/NF**

Updating a VM/VNF/PNF in this context refers to the update of compute (vCPU, vRAM, vStorage) and networking (vFirewall, vRouter, vCDN, etc.) or add /removing physical resources in the mix.

Tools such as Tacker can be used for Auto-Scaling in a NFVI. Tacker is an OpenStack project for NFV Management and Orchestration that deploys and operates VNFs and Physical NSs on an NFV Platform. Tacker is also compatible with TOSCA templates and can be used for: VNF LCM, monitoring, auto-scaling and self-healing.

Monitoring activities must to be done on a recurrent basis, in order to scale network and computing resources as per Enterprise Slice needs. Thresholds must be set in order to scale networking and computing resources up or down as per slice needs. One of the benefits obtained by NFV approach is auto-scaling, which is the ability to dynamically extend or reduce resources allocated to the VNF as needed and at run-time.

Scaling can be of two types (self optimization process capabilities) as is presented in Figure 27:

- **Vertical Scaling** | scaling up/down - ability to add or remove allocated resources for existing computing nodes, such as memory, CPU capacity or storage;
- **Horizontal Scaling** | scaling out/in - ability to scale by add/remove instances, such as VMs (instantiate new VNFs quickly, enabling creation and delivering new services quickly).

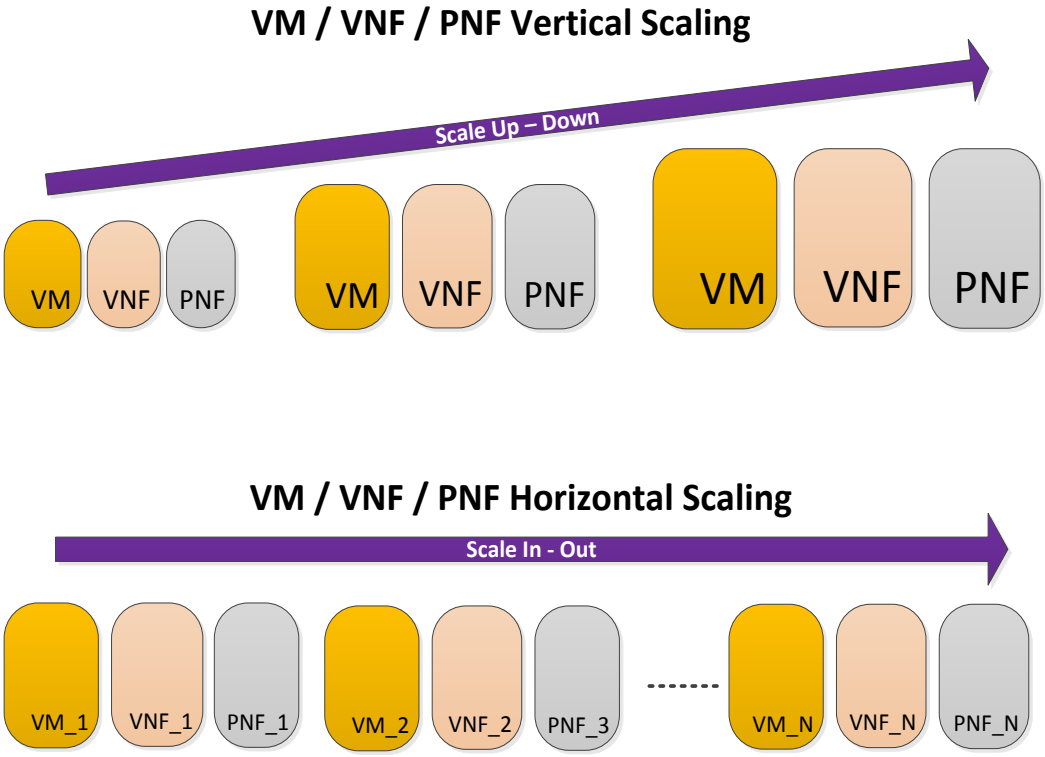

Figure 27 Vertical and Horizontal Scaling

When a certain threshold is hit, more resources can be committed (more bandwidth will be allocated, more vCPU, vRAM or vStorage will be allocated), the VMs could be moved to a superior storage plan automatically (E.g.: from Hard Disk Drive (HDD) to Solid State Disk (SSD), to satisfy high I/O ratios).

One such example is a vCPU reaching a certain set MAX threshold, (e.g. 80%). At that point, a Ceilometer alarm is triggered, which, in turn, initiates the instantiation of a vCPU allocation VNF Template, which will supplement the existing vCPU resources. The supplementary resources usually appear as a new instance, through the Scale In – Out method.

Similarly, when the vCPU of a certain Smart City slice reaches a certain set MIN threshold (e.g. 20%), at which point a Ceilometer alarm is triggered, which in turn, initiates the deletion of a supplementary vCPU VNF template or a decrease in vCPU usage in a VM that is running – the latter implies a restart.

The PROs and CONs of each type of scaling are listed in Table 7 below:

Table 7 Differences between Scale Up – Down and Scale In – Out methods

	Scale Up - Down	Scale Out- In
PROs	- low power consumption - good solution for single VM apps	- easier to manage - no need for VM / VNF instance restart - used as the default scaling technique by most TELCO VNFs - can also be used with VM resources
CONs	- - less scalable - - not used by most TELCO VNFs - - restart must take place for resource allocation	- - more networking resources are needed for each supplementary instance - - its flexibility is limited as it depends on the VM Flavor size and Virtual Deployment Unit which is statically configured

Even critical errors (like Out of Memory (OoM)) should trigger a "re-apply" of some templates, restarting crucial services in order to assure that the service is not affected. Also, different VNFs might apply, considering these thresholds (a different protocol for handling higher bandwidth or a different number of machines in the network).

- **Upgrade/ Rollback VNF/PNF/NF**

There might be some cases in which the VM /VNF templates might not apply correctly. These might be triggered during automatic scaling or a planned upgrade.

First the system has to identify the nodes on which scaling has been done or a planned upgrade has been attempted. TOSCA templates work with interfaces. An interface defines the types of operations that can be performed on nodes. Operations like upgrade and rollback can be defined in the same workflow and soft coded on the same interface.

In case of an auto-scale trigger or VNF upgrade, the upgrade operation is passed and the VNF is upgraded. If the VNF template is up to date, then the upgrade request is ignored. In addition to upgrading, it can save the current configuration for potential rollbacks.

The rollback operation simply takes the previous VNF template and restores it. Multiple rollback levels are supported and should be implemented.

- **Resource configuration**

As stated in 4.2.1, in VIM section, Resource Management Operations are done between the VNFM and the VIM. The resources lie in the NFVI, managed by the VIM. The VNFM (Murano) must give commands to the VIM, in order to allocate resources.

According to ETSI [27], NFV-MANO architecture must be able to support a service composed of VNFs and PNFs and implemented across multivendor environments. Interaction between VNF and PNF is restricted to connectivity functions.

All VNFs managed by one VNFM will use the same virtualized resource management model. This means that resource reservation models can be done at NFVO (Heat) level.

There are multiple resource commitment models. All of them will be uses, according to the Sliced Network needs:

1. **Reservation model** – this is applied on TOSCA template initial definition. Once the template is defined, certain virtual and physical resources are booked at VNF template first run.
2. **Quota /Allowance model** – which implies that a limited amount of resources can be allocated per service or per service slice. These quotas can be specified at VIM level, in the VNF template for the specific service (slice) or at the entire service level.
3. **On demand model** – which implies that resources are allocated on availability, not limiting a service (slice) to its VNF/PNF initially set resource boundaries.

Resource management between VNFM and VIM is depicted in the below figure:

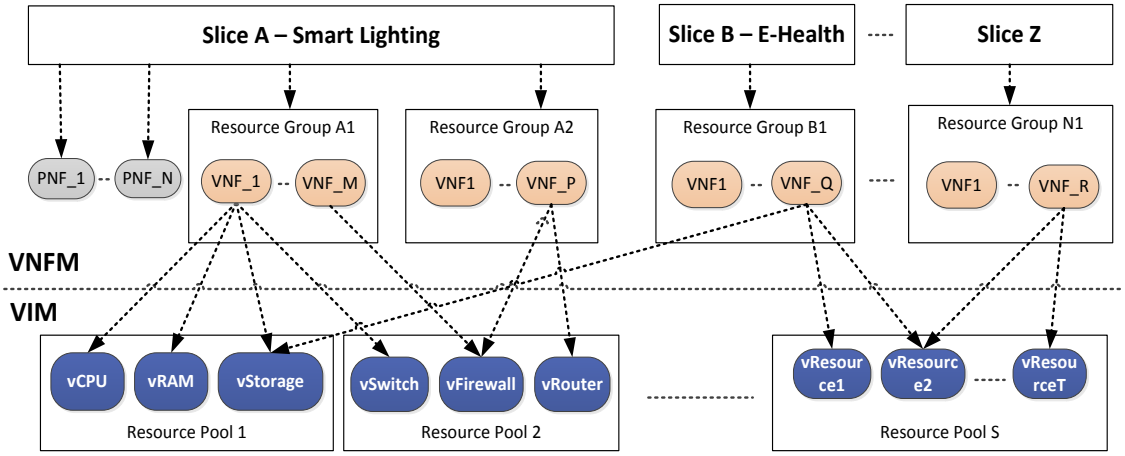

Figure 28 Resource management between VNFM and VIM

In this diagram, each Smart City slice is a “tenant” in the VNFM, to which resource “tenants” (e.g.: vCPU, vRAM, vStorage, vDHCP, vFirewall, vRouter, vSwitch) are allocated.

6 Cross-Plane Orchestration for Enterprise

The general purpose of the cross-plane orchestrator is to provide, as defined in D2.2 [1], the capability of orchestration through several logical layers (virtual infrastructure, resources, slice and service) within the scope of E2E slicing. The scenario is based on the assumption that in this case the DSP and the NSP are combined, under the single domain administration.

6.1 SliceNet vision about Cross-Plane Orchestration for Enterprise

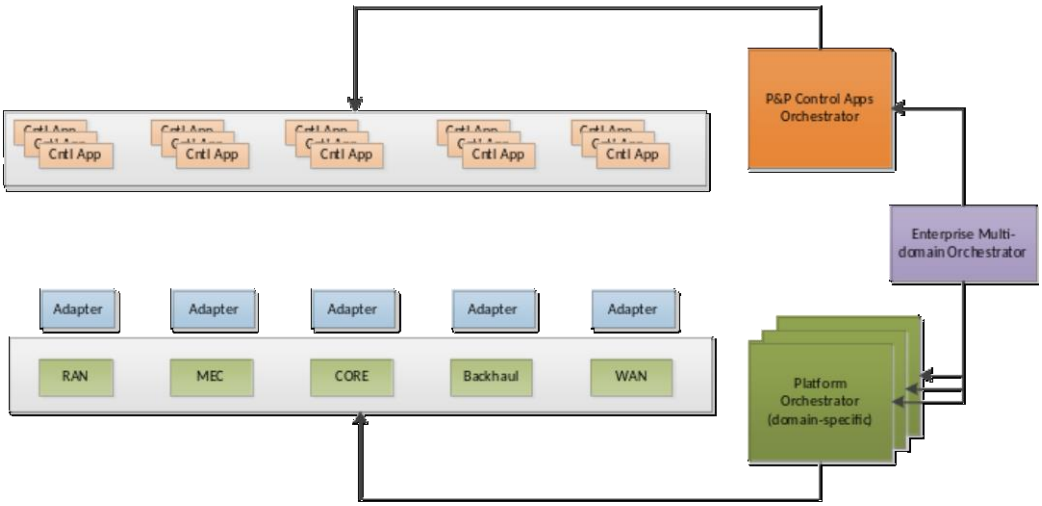

Figure 29 Enterprise orchestrator example, with a two-level orchestration

There is not a unique orchestration recipe from an enterprise perspective. On one hand an enterprise wants to integrate well its existing infrastructure services like specific P&P services, but on the other hand to have its own VNFs for managing NFs. Figure 29 shows an example of enterprise orchestration that employs two levels. First, a dedicated VNF orchestrator, in this example, for P&P Control Apps and for managing enterprise services via control apps.. Second, a Platform Orchestrator for the different domains (e.g., RAN and Mobile Edge Computing (MEC)), e.g. the different segments, technologies, vendors, etc., on top of which there are control adapters exploiting common and technology-independent control primitives. Notice that there can be multiple, domain-specific orchestrators in the latter case, contrary to the single orchestrator in the first level, which manages the control apps of the existing infrastructure. Also, notice that there can be a (meta-) Enterprise Multi-domain orchestrator hiding the two different levels of orchestration.

A single orchestrator may be able to orchestrate and manage both NFs and Network Applications (NAs). However, in SliceNet we not only envision different NF and NA orchestrators, as already considered by European Telecommunications Standards Institute Multi-access Edge Computing (ETSI MEC) [28] and depicted in Figure 2.2 of deliverable D3.1 [2]. But we also envision the differentiation of Common and Dedicated Functions, the earlier aimed for the infrastructure provider and the latter for the slice owners. While the Dedicated Functions serve slice owners towards optimizing slice resource usage, the Common Functions are there to impose cross-slice policies and resolve conflicts between the different dedicated functions, with the goal of optimization from a network perspective rather than a slice perspective.

Regarding the orchestration tools landscape, most of the current solutions are open-source. The most important ones are:

- Open Source MANO (OSM) [29]: OSM is published under Apache v2 license and includes the Service Orchestrator, the Resource Orchestrator and a configuration manager. In one realization of the architecture, the Juju framework [30] is used to provide the VNFM functionalities and Riftware [31] to support orchestration.
- Tacker [32]: Tacker is an official OpenStack project targeting data centre environments, building a VNFM and a NFVO to deploy and operate NSs and VNFs on the OpenStack infrastructure platform.
- Open Network Automation Platform (ONAP) [33] [34]: ONAP brings together Open Enhanced Control, Orchestration, Management and Policy (ECOMP) and Open Orchestrator Project (Open-O) [35] as a platform for real-time, policy-driven orchestration and automation of both physical and virtual NFs. It is supported by Linux Foundation.
- Open Platform for NFV (OPNFV) [36]: realizes the ETSI MANO framework, supported by the Linux Foundation. OPNFV integrates OpenStack [18] as the supporting cloud management system and also considers for a number of SDN controllers. OpenStack is used in the cloud orchestrator role.
- SONATA [37] and 5GEx (5G Exchange) [38]: SONATA and 5GEx comprise two new types of orchestrator following ETSI MANO. JOX (defined below) employs an architecture that is similar to the SONATA design, exploiting a message bus to support the plugin communication subsystem.
- Open-Baton NFVO [39]: Open-Baton is designed and implemented following the ETSI MANO specification. It uses message queueing for the communication with the VNFM, however both OpenBaton and SONATA do not inherently support life-cycle management of network slices.
- JOX [40]: JOX is an open-source, event-driven orchestrator for the virtualized network which can be included in this list as a validation/prototyping tool. It supports network slicing natively for the MEC platform and its app as well as for RAN and CN segments. From the implementation perspective, JOX is tightly integrated with the Juju VNFM framework.
- HEAT: main orchestration project from Openstack; provides the capability to launch multiple cloud applications using templates written in TOSCA.
- Other frameworks: there are other framework such as CloudNFV, Puppet, Chef and Cloud Foundry, the Unify solution that describes a multi-layer service orchestration in a multi-domain network, Central Office Re-architected as a Datacenter (CORD/XOS), Gigaspaces Cloudify, etc.

6.2 Smart City Cross-Plane Orchestration

In Enterprise environment, the purpose of the Orchestration engine is to automate, optimize and abstract the operational tasks for the deployment of apps and services. Orchestration module is opening its lower level components by the use of an open REST API which, integrated in the SliceNet framework, is consumed by OSA to trigger the slice instantiation in enterprise environment.

Smart City cross-plane orchestration engine is represented by HEAT, the main orchestration project developed in Openstack. Heat relies on three components:

- heat-engine – main component which allows orchestration tasks;
- heat-api – REST API which listens for API requests and send them to heat-engine;
- heat – command line interface (CLI) which uses heat-api.

Using a template based model, HEAT implements an orchestration engine to allow the instantiation of Smart city apps. HEAT is interacting with VIM in order to orchestrate the physical layer (which provides infrastructure resources like servers, volumes and floating IPs) and is also interaction with VNFM in order to orchestrate the application layer. As depicted in Figure 30, HEAT is also interacting with the monitoring service in order to provide auto-scaling capabilities.

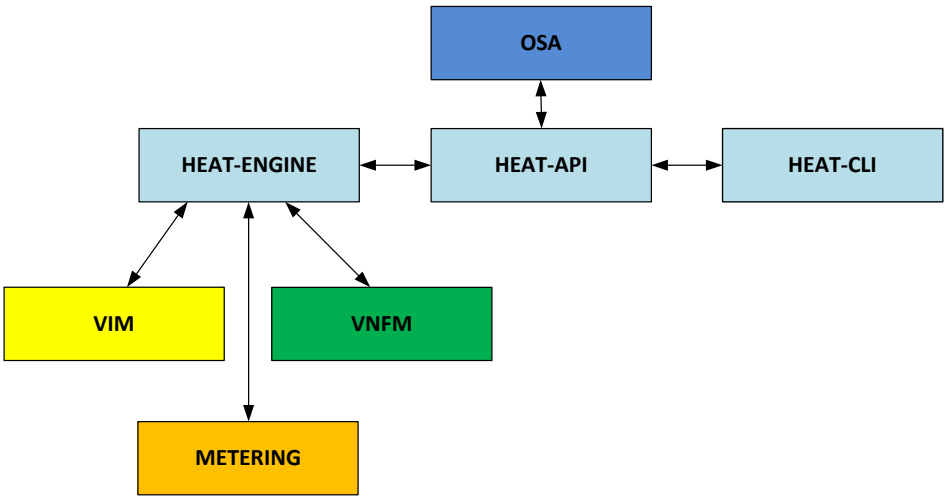

Figure 30 HEAT integration and interaction

6.3 Enterprise integration in SliceNet E2E architecture

An innovative cross-plane orchestrator must be able to automate the different functionalities (FCAPS) associated to the automation of the slice management. The orchestrator will coordinate the operations of the data plane, CP, management plane and service plane for optimised network slicing, and deploy NFV actions for operational requirements on demand related to QoE sensors /actuators to monitor /optimise the QoE of a use case service. The other task is to prototype the demanding vertical business use cases.

The idea of Cross-Plane Orchestration for Enterprise is firstly to provide a set of enabling automated mechanisms which can perform the cross-plane configuration of all the architectural components involved in the layers from diagram presented in Figure 4 (CP, Management Plane, Services Plane and Applications Plane) in order to do an efficient slice management. Another purpose of the Cross-Plane Orchestration is to provide a set of coordination functions across several logical layers and constructs (e.g. service, slice, resource, and infrastructure) with the aim orchestrating the provisioning of E2E slices.

It was defined in the deliverable D2.2 [1], subchapter 6.3, that are two types of orchestration defined in SliceNet: horizontal orchestration and vertical orchestration (see Figure 31). It is called horizontal orchestration when it refers to multiple domains, either administrative or technologic, and the orchestration is performed within the same layer or for the purpose of the same logical construct. Additionally, it is called vertical orchestration when it refers to multiple layers, e.g. a single network provider performs vertical orchestration when deploying a service instance where the service is composed by one or more slices and each

slice is composed by multiple resources. Given the three layers considered in SLICENET, e.g. services, slices and resources; three layers of orchestration are therefore considered the building blocks of the Cross-Plane Orchestration: service orchestration, slice orchestration and resource orchestration [1].

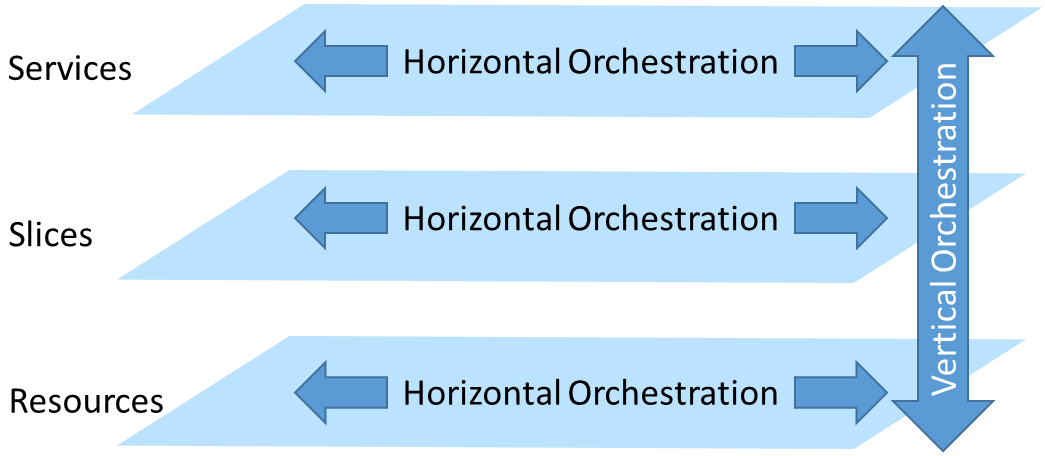

Figure 31 Horizontal and Vertical Orchestrations [1]

So tacking into account the aspects presented above, it could be identified in the Virtualized Enterprise Segment the both models of orchestration, vertical and horizontal. The vertical orchestration is referring to the multiple layers involved (service, slice, resource, and infrastructure) and the horizontal is composed by: one slice orchestrator, one service orchestrator for the Intelligent Lighting service and one resource orchestrator who is represented in this case by VIM.

Overall, referring to Enterprise integration in SliceNet E2E architecture it is important to assure the early integration of slicing control, slice management and orchestration for Smart City use case, Intelligent Lighting.

7 One Stop-API

7.1 OSA for Enterprise

OSA enables verticals to develop and deploy specialized 5G services on top of the SliceNet platform. The OSA can reside on top of the Open Data API, shown in Figure 32, which works as an interface for communicating with the underlying Control Apps. The figure presents an abstraction scheme for supporting monitoring, control and programmability by means of high-level technology-agnostic level APIs (depicted in purple colour, 2nd level northbound APIs). In addition, there are low-level APIs (depicted in green colour, forming a 1st level northbound API). The corresponding SDKs, namely, the Platform SDKs at the first level and the App SDKs, decouple the control logic from the Data plane actions following SDN principles. They also enable the extract aggregated and structured network configuration, status and topology information in the form of instantaneous /current network graphs that allow for a better data analysis and decision-making. Moreover, the SDKs facilitate the development of extendable network Control Apps that coordinate with one another.

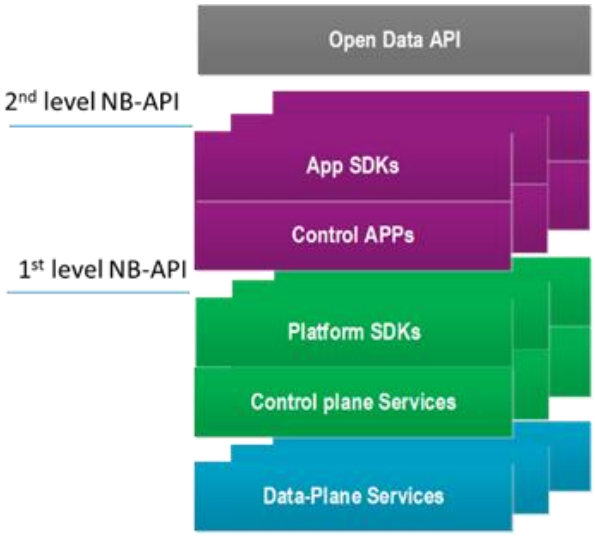

Figure 32 Abstraction scheme

From the enterprise point of view, the OSA appears as an abstraction layer. Below that layer, there can be a different scheme, depending on the underlying needs. In order to perceive this concept, it is presented in Figure 33 below, which portrays an abstract approach to content access. Note that the OSA lies on top of App SDK level, with the latter serving as an interface layer for what lie beneath. The content-access example shows two extreme cases. On the left side, there is a bundled OSA API model on top of a unified App SDK, with all Control Apps being tightly coupled and interdepending on one another. The latter implies the Control Apps in this case need to coordinate. However, on the right side, there is an OSA API model that is fragmented into different SDKs and corresponding underlying loosely coupled Control APPs. In this latter case, the Control Apps are not necessarily depended on one another; hence, they do not need to coordinate.

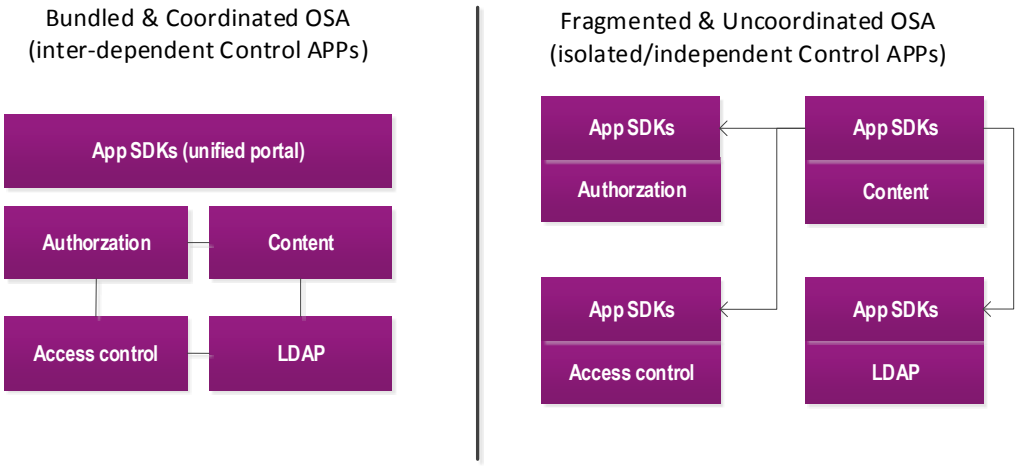

Figure 33 OSA paradigm with respect to two extreme cases: a bunddled OSA API model sitting on top of a unified App SDK (on the left), and a fragemented OSA API model (on the right).

7.2 OSA integration in SliceNet E2E architecture

OSA allows verticals to express their request for specialised 5G services in a technology-agnostic form of a service template, which describes the creation of a slice with specific QoS characteristics. Then, the request is passed to an appropriate Control App, which comprises a 2nd level northbound API (see Figure 32). The Control App is responsible for translating this technology-agnostic request to a policy for slicing used as input by the appropriate Platform SDK and CP services (1st level northbound API portrayed in Figure 32). The latter yields a technology dependent format (a.k.a adaptors) which can be understood and implemented by the underlying Data Plane Services. Without loss of generality, Figure 34 below presents the case of RAN slicing as an example, which is along the lines of what was described above:

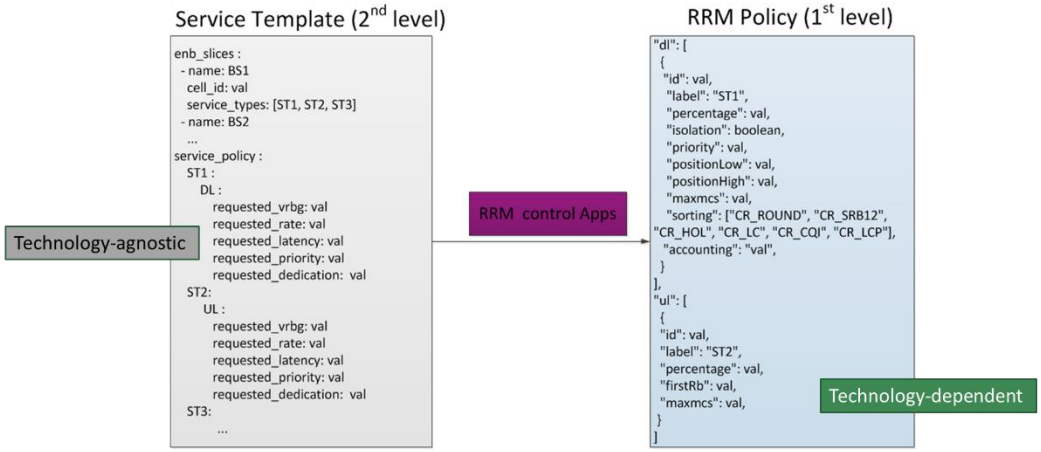

Figure 34 RAN slicing example

A request for slicing the RAN resources follows an abstract service template. It gets passed to an RRM Control App which can create a policy for the CP Services.

Finally, apart from being an interface for the OSA communication with the underlying Control Apps, the Open Data API is a kind of a Pub/Sub communication layer that allows Control Apps to interact via exchanging capabilities, commands and information about their current status. This means that Control Apps may further provide APIs to other Control Apps.

Most of the times, the Control Apps are part of the network functions of an operator, yet via the Open Data API, third parties can access the limited network status information to optimize their services or manage the network slices dedicated to them via the OSA.

7.3 OSA for Smart City

OSA is the innovative one-stop shop implementation for vertical's customized services creation, as a rapid, efficient and scalable deployment, in relation with the P&P function, as a dynamic slices configuration, enabled for vertical's specialized services. The Smart City services are OSA exposed through an aggregation of NSI/NSSI/NF selectable features that will offer the requested service, through slice creation for a specific use-case implementation of smart lighting apps, based on series of specific KPIs.

The OSA will provide to the enterprise the one-stop-shop information for creation the service /app with specific requirements, based on service templates catalogue. The OSA may provide in this case both vertical management (app based for service selection) and horizontal management for administrative roles.

For Smart City vertical slicing configuration at the enterprise level, through the OSA the request are for configuration of specific VNFs (with specific parameters), as an app request from the smart lighting owner (local city administration) by providing the information for the smart lighting network slice creation (number of devices, location, communication needs, bandwidth and delay requirements, availability) or administrative implementations as software upgrades.

8 Prototyped friendly network segment for Smart City

The SliceNet approach for the current network infrastructure is to remove the limitations of current technical approaches to a software infrastructure, 5G ready, including elements of control and management, oriented to vertical services orchestration. The innovation proposed is also extended to the Enterprise segment development, as a friendly sliced implementation over the dedicated enterprise network infrastructure (virtualized implementation), within a single network domain, ready and capable of SSaS that enables the vertical to select, deploy and consume the proper service slice. The slicing friendly network segment of enterprise contains the key elements of 5G programmability, software defined networks, cloud computing, QoS and QoE awareness, with respect of vertical KPIs and SLA.

8.1 Smart city segment description

The Smart City segment is included in the IoT of connected objects that requires specific communication services, as described in D2.1 [4] and D2.2 [1]. The service slice is composed by different sub-slices that correspond to the access (RAN) and core part (virtual Evolved Packet Core (vEPC)) or Next Generation Core Networks -NGCN) and in addition is terminated, from a vertical perspective on top of the enterprise.

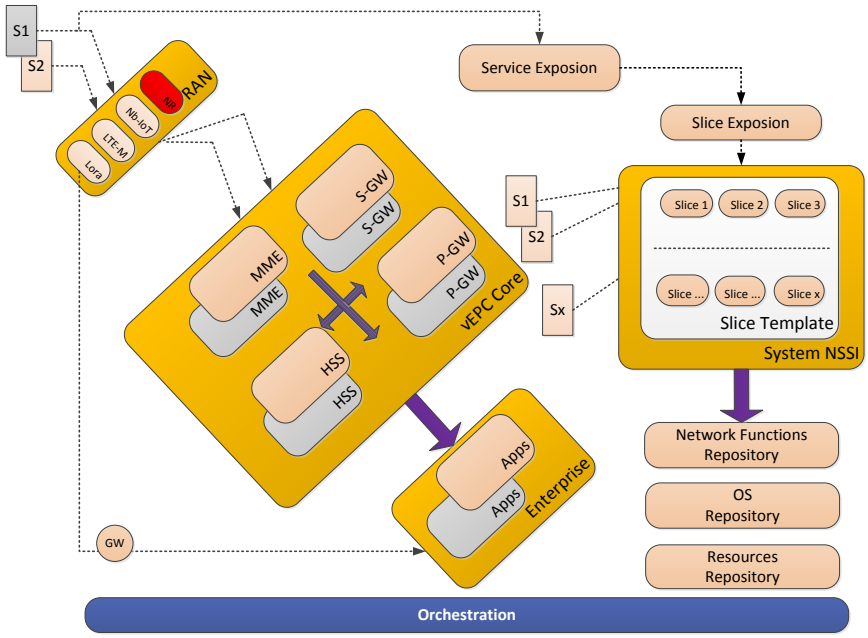

Figure 35 Smart City High Level Architecture

As presented in WP2 [41] deliverable, the use-case proposes smart lighting services, to be consumed by the vertical (City Hall or Lighting Company), split into three main domains:

- RAN domain, step-by-step approach
  - Long range, low power wireless platform (Lora); Long Term Evolution for Machines (LTE-M); Narrowband Internet of Things (Nb IoT); 5G New Radio (NR) (sliceable)
- Core domain (sliceable)
  - vEPC
  - NGCN

- Enterprise domain
  - Virtualized infrastructure

The S1 and S2 represent two different services, used by two consumers, within the Smart City apps, each consumers requiring and using a different service and slice type, chosen from a template catalogue of services. In this scenario, as the Network Operator (NO) plays the combined roles of DSP and NSP, the service is offer by the NO that also ensures the communication with the Enterprise Lighting Apps, in this particular case under the NO administrative domain.

8.2 Prototyped Deployment plan

The deployment plan of prototyping envisages all enterprise infrastructure adaptation and services, ready to run into the 5G virtualized, programmed and automated context, opened to a SBA approach, as presented in D2.2 [1], with respect of the smart lighting and business model requirements from D2.1 [4], including the concepts of CP and Management Plane, as referenced into D2.3 [5] and D2.4 [6].

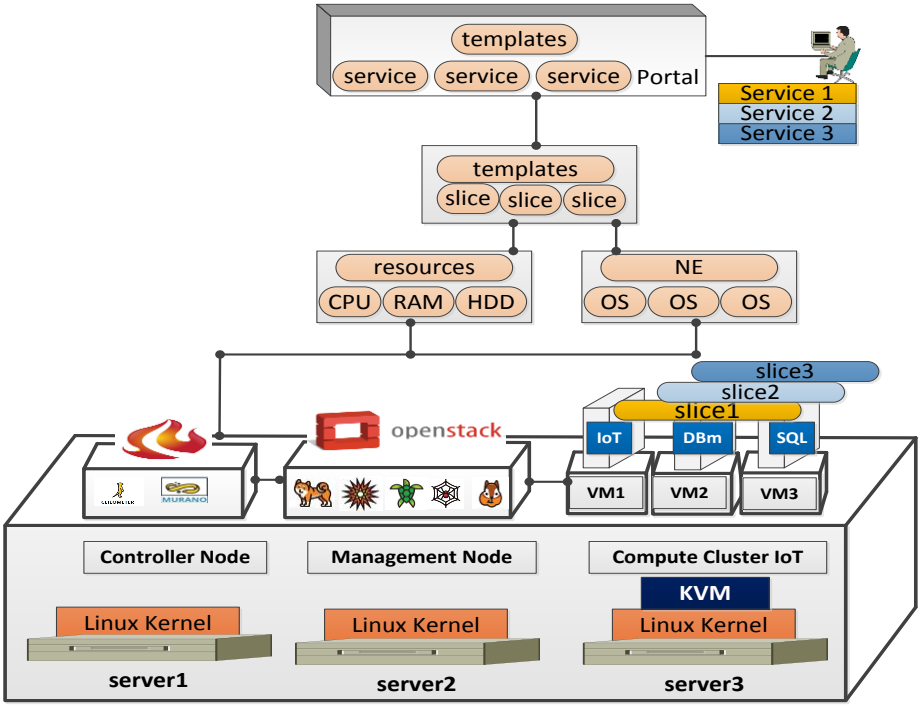

Figure 36 High Level Enterprise Infrastructure

As described in Figure 36, the prototyping model plans to enable vertical consumers, the DSC, to benefits of the communications resources and apps exposed by the infrastructure for IoT Smart City services, into a friendly manner. The system proposed is flexible and may cope with different use cases, beyond the Smart Lighting apps, that are demonstrated in the task. The Enterprise prototype proposed is intended to use and integrate de logical functional blocks and architectures proposed in WP2 [41], WP4 [42] and WP5 [43] and to be used as a functional SliceNet component in WP7 [44] and WP8 [45].

Detailed deployment plan is addressed into the next chapters, starting from physical and resource layers, to the virtualization implementations based on open tools, including resource control, resiliency and availability aspects, management components, up to the

specific apps for Smart City use-case that are instantiated automatically, as a set of pre-defined VNFs, services chained, and then the services are exposed to the consumer.

8.2.1 Resources modelling and design for Enterprise Infrastructure

After it was presented briefly all the functional blocks inside the ETSI NFV architecture framework in Subsection 2.1, it is necessary to answer to an important question in the mission to deploy a functional NFV service: What are the certain hardware requirements to sustain a stable 200 VM using an OpenStack environment?

To answer to this question, several assumptions must be made, like:

1. It is necessary to sustain 200 VMs
2. No CPU oversubscribing
3. Number of GHz per core: 2.2GHz
4. Hyper-threading supported: use a factor of 2
5. Number of GHz per VM (average compute units): 2GHz
6. Number of GHZ per VM (maximum compute units): 16GHz
7. Intel Xeon E5-2680L v2 core CPU: 12 cores per CPU [46]
8. CPU socket per server: 2 CPU socket per server blade

- Number of CPU cores per VM [46]:

$$\frac{\max GHz}{(number\ of\ GHz\ per\ core\ \cdot 1.3\ for\ hyper-threading)} = \frac{16\ GHz}{2.2\ GHz\ \cdot 2}$$
$$= 3.6\ CPU\ cores\ per\ VM$$

- Total number of CPU cores [46]:

$$\frac{Number\ of\ VMs \cdot number\ of\ GHz\ per\ VM(average\ compute\ units)}{number\ of\ GHz\ per\ core} = \frac{200 VMs \cdot 2\ GHz}{2.2\ GHz}$$
$$= 181\ CPU\ cores$$

- Number of core CPU sockets [46]:

$$\frac{Total\ number\ of\ CPU\ cores}{number\ of\ cores\ per\ CPU} = \frac{181\ CPU\ cores}{12\ cores\ per\ CPU} = 15\ sockets$$

- Number of socket servers [46]:

$$\frac{Total\ number\ of\ sockets}{number\ of\ sockets\ per\ server} = \frac{15\ sockets}{2\ socket\ per\ server} = 7\ dual\ socket\ servers$$

- Number of VMs per server [46]:

$$\frac{Number\ of\ VMs}{number\ of\ servers} = \frac{200\ VMs}{7\ dual\ socket\ server} = 29\ VMs\ per\ server$$

To summarize the results presented above it was created Table 8.

Table 8 CPU calculations results for our case

Nr. Crt.	Item calculated	Results
1	Number of CPU cores per VM	3.2 CPU cores per VM
2	Total number of CPU cores	160 CPU cores
3	Number of core CPU sockets	13 sockets
4	Number of socket servers	7 dual socket servers
5	Number of VMs per server	29 VMs per server

Based on the previous results, we can conclude that could be deployed 29 VMs per compute node. Also, memory sizing is also important to avoid making unreasonable resource allocations. Next will be proceeding to memory calculations.

An assumption list for memory calculations:

1. 2 GigaByte (GB) of RAM per VM
2. 8GB of RAM maximum dynamic allocation per VM
3. Compute nodes supporting slots of: 2, 4, 8 and 16 GB sticks

- RAM available per compute node [46]:

$$\frac{Total\ available\ of\ RAM\ installed\ on\ server}{Max\ available\ of\ RAM - stick\ size\ mounted} = \frac{128\ GB}{16\ sticks} = 8\ GB\ of\ RAM\ available\ per\ compute\ node$$

To accomplish the target to build the enterprise infrastructure it is necessary to achieve the best performance and networking experience. The following values can be assumed for that:

1. 200 Megabit (Mbits) per second is needed per VM
2. Maximum network latency

To accomplish all of those assumptions made previously it is necessary to use a 10GB link for each server, which will give [46]:

$$\frac{10000\ Mbits\ per\ socond}{29\ VMs} = 344\ Mbit\ per\ second$$

This value is very satisfying, so the resources modelling planning could go further to identify highly available network architecture. To solve this task, it is an alternative to use two data switches with a minimum of 29 ports for data. If we assume a future growth of the network, it is proper to consider using a switch aggregation that uses the Virtual Link Trunking (VLT) technology between the switches in the aggregation. This feature allows each server rack to divide their links between the pair of switches to achieve powerful active-active forwarding while using the full bandwidth capacity with no requirement for a spanning tree.

Taking in consideration all the previous assumptions and results, it is necessary to plan for an initial storage capacity per server that will serve 29 VMs. Considering the storage calculations, several assumptions must be taken into account in this case also:

4. 100GB for storage for each VMs drive
5. The usage of persistent storage for remote attaching volumes to VMs

A simple calculation shows for 200VMs a space of $200\,VMs \cdot 100GB = 20TB$ of local storage used. It can be assigned 150GB of persistent storage per VM to have $200\,VMs \cdot 150GB = 30TB$ of persistent storage.

Therefore, the final question is how much storage should be installed by the server serving 29 VMs, 51 $GB \cdot 29VMs = 1479\ GB = 1.5\ TB$ is the result.

In conclusion, it is necessary 1.5 TB per server to serve 29 VMs.

8.2.2 Prototyped Physical Layer

In the Physical Layer Implementation, it was chosen in the virtualized infrastructure the physical resources displayed in Table 9.

Table 9 Physical resources

Compute node role	No of processors [CPUs]	Memory [GiB]	Hard Disk Capacity [GiB]	No of NICs
Controller node	2	128	600	6
Management & Orchestration node	2	128	1024	6
Compute node	2	128	1800	6

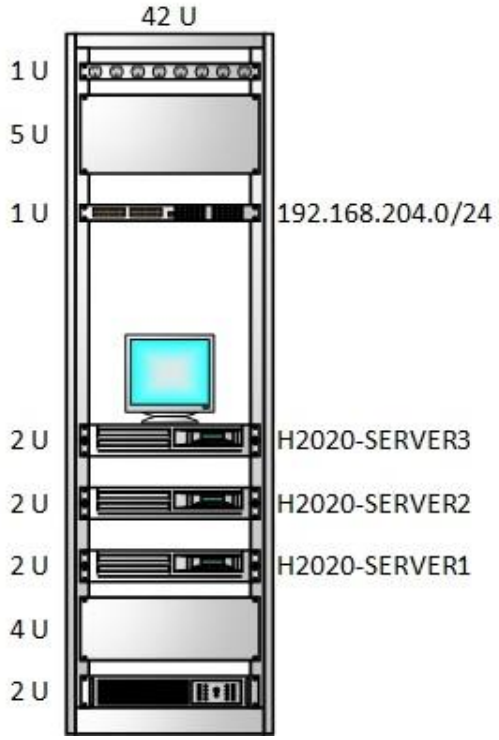

Figure 37 Physical representation of infrastructure

Represented in Figure 37, is the test bed set-up integrated in order to prove the technological capabilities of the distributed virtualized enterprise app. As it can be seen in Figure 37 and also mentioned in Table 9, the virtualized infrastructure is distributed on tree servers. All the physical system capabilities are mentioned in Table 9 and, regarding the network interface configuration is presented in Annex B.

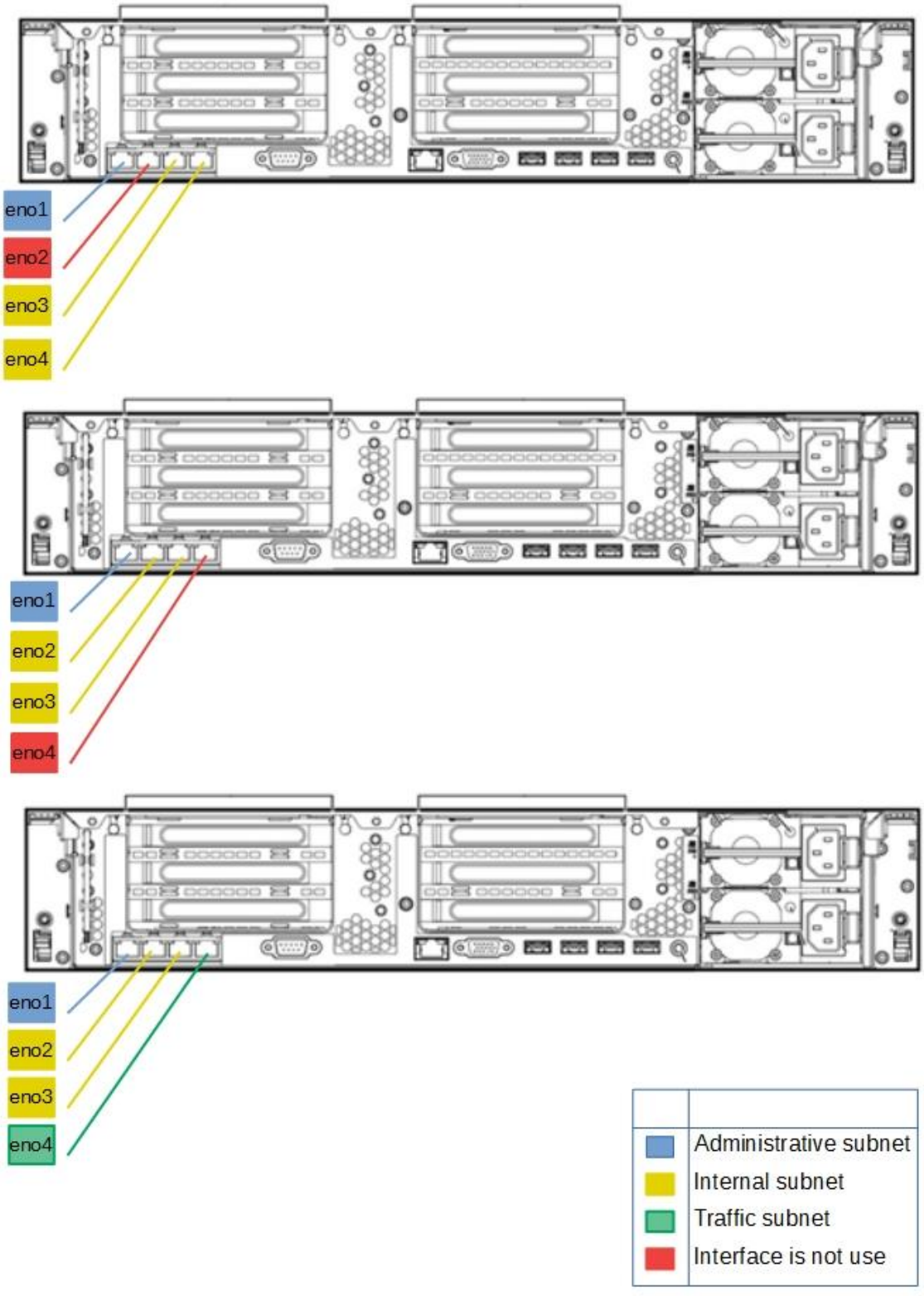

Figure 38 Backend view of the physical infrastructure

Figure 38 represents the backend perspective of the physical infrastructure and each of the physical interfaces and the use of it. As it shows above some of the interfaces are used in different subnets and the configuration each of them are detailed in Annex B and configure name resolution in Annex C.

As a conclusion of this subchapter, it was presented the physical layer implementations and the configurations necessary to sustain the good functionality of the distributed virtualize system and all the apps above this. From the physical point of view all the interconnection is working smoothly.

8.2.3 Prototyped Enterprise cloud

As it has been described in section 2.4 of this document, there are three main types of clouds: public, private and hybrid. In the presented prototyped segment, was followed to develop and build from scratch an enterprise cloud platform based on Openstack Ocata version. All Openstack tools have been configured on different physical machine, as follows: control node, orchestration node and compute node.

In the Figure 39, entire part represented as “Engineering Lab” is composed of three physical servers and in this section will be detailed server number 3, known as compute node.

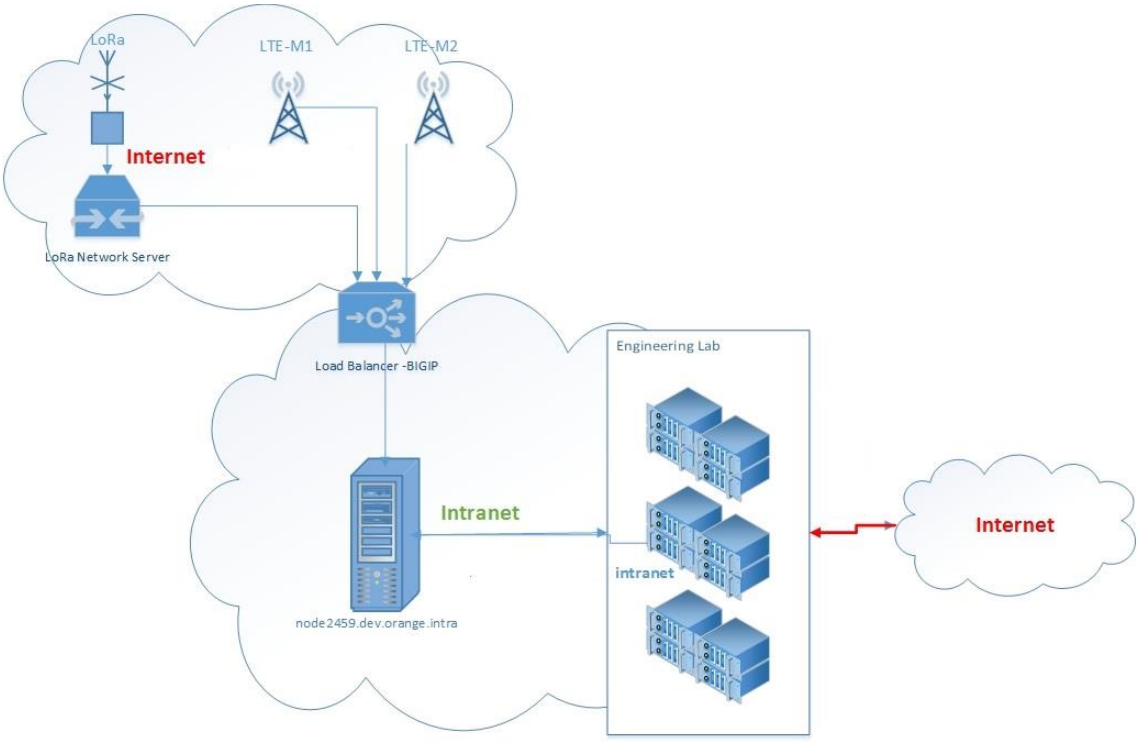

Figure 39 The topology used in laborator for the prototype

First component that has been installed and configured on it was KVM, who is a hypervisor that manages the link between virtual and physical resources. After this, were installed all necessary tools and has also been mounted necessarily volume for storage usage.

```
h2020@h2020-server3:~$ kvm --version
QEMU emulator version 2.8.0(Debian 1:2.8+dfsg-3ubuntu2.9~cloud3)
Copyright (c) 2003-2016 Fabrice Bellard and the QEMU Project developers
```

Figure 40 KVM Version on Ubuntu Server 16.04

- **Volume for storage resources (Figure 41):**

```
h2020@h2020-server3:~$ sudo vgdisplay
--- Volume group ---
  VG Name               h2020-server3-vg
```



```
[…]
--- Volume group ---
  VG Name               cinder-volumes
  System ID
  Format                lvm2
  […]
  VG Size               1.55 TiB
  PE Size               4.00 MiB
  Total PE              405189
  Alloc PE / Size       404480 / 1.54 TiB
  Free  PE / Size       709 / 2.77 GiB
  VG UUID               muVJUu-A8jl-JI8A-iJYn-il2x-tGjd-c6uoLX
```

Figure 41 List volumes group from compute node

In the following paragraphs, it was shown how could be created new VMs. There are two possibilities: the first one is to use Horizon, friendly web interface and the second one involves the use of CLI.

1. Web interface (HORIZON): http://192.168.204.15/horizon/auth/login/?next=/horizon/

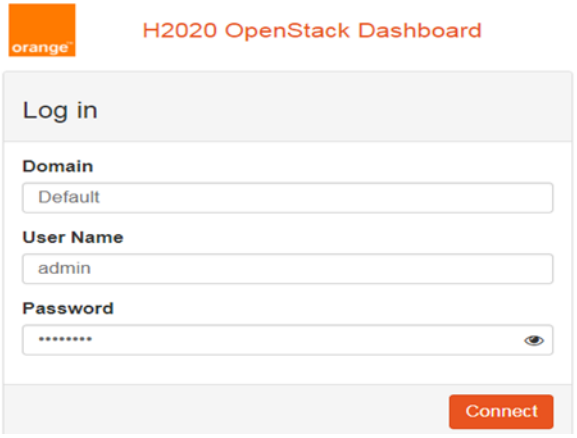

Figure 42 Login page of ORO Openstack Platform

After the login (Figure 42), we can go to “Project -> Compute -> Instances” and hit the “Launch Instance” button. In this page, we have to define a list of parameters such as: Instance Name, Availability Zone, Source, OS, Flavor, Networks Ports and Interfaces, Security Groups, and so on.

On the other hand, it has been also installed on server number 2, Heat, a tool that helps to orchestrate entire infrastructure. So, we will continue to detail instantiation mode through orchestration flow.

It has to be accessed “Project -> Orchestration -> Stacks” and choose a stack that has been previously defined.

For instantiation of a VM with IoT flavor, we will use the template from Figure 43.

```
description: Launch a basic instance with UBUNTU image using the ``IoT_platform_high_version``
  flavor, ``mykey`` key,  and one network.
heat_template_version: '2015-10-15'
outputs:
```

```
  instance_ip:
    description: IP address of the instance.
    value:
      get_attr: [server, first_address]
  instance_name:
    description: Name of the instance.
    value:
      get_attr: [server, name]
parameters:
  NetID: {description: Network ID to use for the instance., type: string}
resources:
  server:
    properties:
      flavor: IoT_platform_high_version
      image: ubuntu
      key_name: mykey
      networks:
      - network: {get_param: NetID}
    type: OS::Nova::Server
```

Figure 43 IoT platform template

2. CLI: ssh h2020@192.168.204.15

To launch a new instance from CLI, just run the following command from Figure 44.

```
openstack server create --flavor IoT_platform_high_version --image ubuntu --nic net-id=f6291e27-7476-41b4-9453-5769a32c0fcf --security-group default --key-name mykey ThingsBoardIoT_Platform
```

Figure 44 Launch new instance from CLI of controller node

In Figure 46 are listed all available VMs from our infrastructure and in Figure 45 network topology of all machines from cloud.

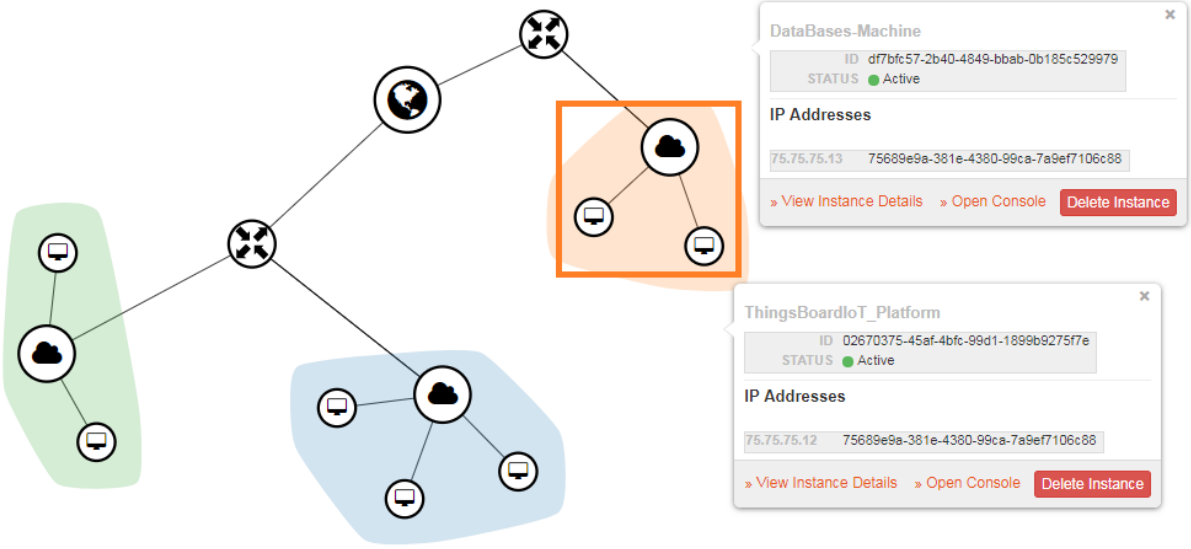

Figure 45 Network topology – Cloud Infrastructure

The output from Figure 46 represents all available VMs in the Openstack infrastructure.


```
h2020@h2020-server1:~$ openstack server list
+----+------------------------------+--------+--------------+----------------------+
| ID | Name                         | Status | Networks     | Image Name           |
+----+------------------------------+--------+--------------+----------------------+
| ## | DataBases-Machine            | ACTIVE | selfservice5 | ubuntu               |
| ## | stack_2-server-fefbk33tjgo3  | ACTIVE | selfservice3 | windows-server-2012  |
| ## | ThingsBoardIoT_Platform      | ACTIVE | selfservice5 | ubuntu               |
| ## | server3                      | ACTIVE | selfservice1 | ubuntu               |
| ## | stack-server-2chti6z6exxp    | ACTIVE | selfservice1 | ubuntu               |
| ## | server2                      | ACTIVE | selfservice1 | ubuntu               |
| ## | server1                      | ACTIVE | selfservice3 | ubuntu               |
+----+------------------------------+--------+--------------+----------------------+
```

Figure 46 Available VMs

For the moment, main interest is on VMs with number 1 (DataBases-Machine) and number 3 (ThingsBoardIoT_Platform).

In the following lines, it will be describe how the databases and IoT platform has been installed and configured such that can settle up the entire (E2E) cloud infrastructure.

In the Figure 47, it is represented a simple imagine of VMs and host server placed above KVM hypervisor.

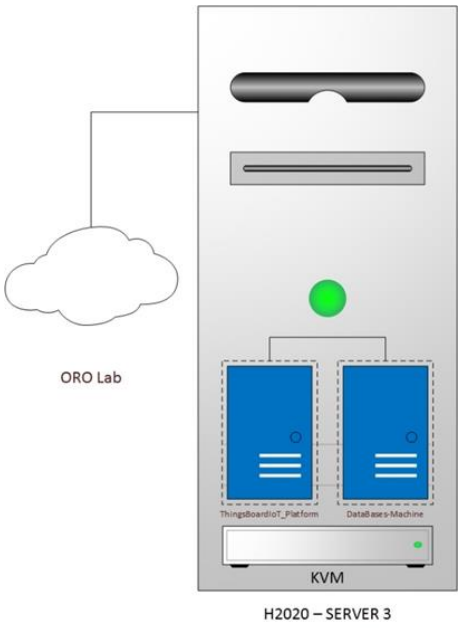

Figure 47 IoT VMs

- **Installing IoT Platform:**

Prerequisites and informations about the system are listed in Figure 48:

```
ubuntu@ThingsBoardIoT-Platform:~$ uname -a
Linux ThingsBoardIoT-Platform 4.4.0-124-generic #148-Ubuntu SMP Wed May 2 13:00:18 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux

#virtualization layer
ubuntu@ThingsBoardIoT-Platform:~$ sudo dmidecode -s system-product-name
OpenStack Nova
ubuntu@ThingsBoardIoT-Platform:~$ java -version
```

```
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)

#Update ThingsBoard memory usage and restrict it to 256MB in /etc/thingsboard/conf/thingsboard.conf
export JAVA_OPTS="$JAVA_OPTS -Xms256M -Xmx256M"
```

Figure 48 Prerequisites

To install the platform, a package has to be downloaded, unzip it and after that edit some configuration files and start the entire platform like in Figure 49.

```
wget https://github.com/thingsboard/thingsboard/releases/download/v1.4/thingsboard-1.4.deb
sudo dpkg -i thingsboard-1.4.deb
```

Figure 49 Download the packets and install

To start the platform, use the command from Figure 50:

```
#start the platform
sudo service thingsboard start
#status of the platform
service thingsboard status
Redirecting to /bin/systemctl status  thingsboard.service
● thingsboard.service - thingsboard
  Loaded: loaded (/usr/lib/systemd/system/thingsboard.service; enabled; vendor preset: disabled)
  Active: active (running) since Thu 2018-03-22 18:51:06 EET; 1 months 29 days ago
 Main PID: 15390 (thingsboard.jar)
  CGroup: /system.slice/thingsboard.service
        ├─15390 /bin/bash /usr/share/thingsboard/bin/thingsboard.jar
        └─15404 /usr/bin/java -Dsun.misc.URLClassPath.disableJarChecking=true -Dplatform=rpm -jar /usr/share/thingsboard/bin/thingsboard.jar

Mar 22 18:51:06 ThingsBoardIoT-Platform systemd[1]: Started thingsboard.
Mar 22 18:51:06 ThingsBoardIoT-Platform systemd[1]: Starting thingsboard...
Mar 22 18:51:08 CentOS72x86-apps thingsboard.jar[15390]: ================================================
Mar 22 18:51:08 ThingsBoardIoT-Platform thingsboard.jar[15390]: :: ThingsBoard ::  (v1.4.0)
Mar 22 18:51:08 ThingsBoardIoT-Platform thingsboard.jar[15390]: ================================================
```

Figure 50 IoT platform status in Linux environment

- Installing and configuring Postgres SQL Database and Cassandra NoSQL Database (Figure 51):

```
root@databases-machine:/home/ubuntu# systemctl status postgresql.service
● postgresql.service - PostgreSQL RDBMS
  Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
  Active: active (exited) since Wed 2018-05-16 15:36:28 UTC; 4min 13s ago
 Main PID: 21927 (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/postgresql.service
May 16 15:36:28 databases-machine systemd[1]: Starting PostgreSQL RDBMS...
May 16 15:36:28 databases-machine systemd[1]: Started PostgreSQL RDBMS.
May 16 15:36:37 databases-machine systemd[1]: Started PostgreSQL RDBMS.
```



```
May 16 15:40:37 databases-machine systemd[1]: Started PostgreSQL RDBMS.
ubuntu@databases-machine:~$ sudo -i -u postgres
postgres@databases-machine:~$ psql -d postgres -W
Password:
psql (9.5.12)
Type "help" for help.
postgres=# CREATE DATABASE thingsboard;
postgres=# \list
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
thingsboard | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)
```

Figure 51 Postres database configuration

- Installing and configuring Cassandra NoSQL Database (Figure 52):

```
# Add cassandra repository
echo 'deb http://www.apache.org/dist/cassandra/debian 311x main' | sudo tee --append
/etc/apt/sources.list.d/cassandra.list > /dev/null
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-get update
## Cassandra installation
sudo apt-get install cassandra
## Tools installation
sudo apt-get install cassandra-tools

ubuntu@stack-db-server-dhd2vdfa7i55:~$ sudo service cassandra status
sudo: unable to resolve host stack-db-server-dhd2vdfa7i55
● cassandra.service - LSB: distributed storage system for structured data
   Loaded: loaded (/etc/init.d/cassandra; bad; vendor preset: enabled)
   Active: active (exited) since Thu 2018-05-17 10:27:28 UTC; 14min ago
     Docs: man:systemd-sysv-generator(8)

May 17 10:27:28 stack-db-server-dhd2vdfa7i55 systemd[1]: Starting LSB: distributed storage system
for structured data...
May 17 10:27:28 stack-db-server-dhd2vdfa7i55 systemd[1]: Started LSB: distributed storage system
for structured data.
May 17 10:37:51 stack-db-server-dhd2vdfa7i55 systemd[1]: Started LSB: distributed storage system
for structured data.
```

Figure 52 Cassandra database configuration

In section 8.3, will be presented the services which can used the platforms and infrastructure depicted above.

8.2.4 Prototyped SDN and VIM integration

8.2.4.1 Prototyped SDN

The OpenStack Ocata (v15.0) module responsible with the functions of the SDN in the enterprise solution is Neutron. This module manages networking virtualization that runs on top of the OpenStack engine.

From the networking point of view, the setup used is presented in the Figure 53, below.

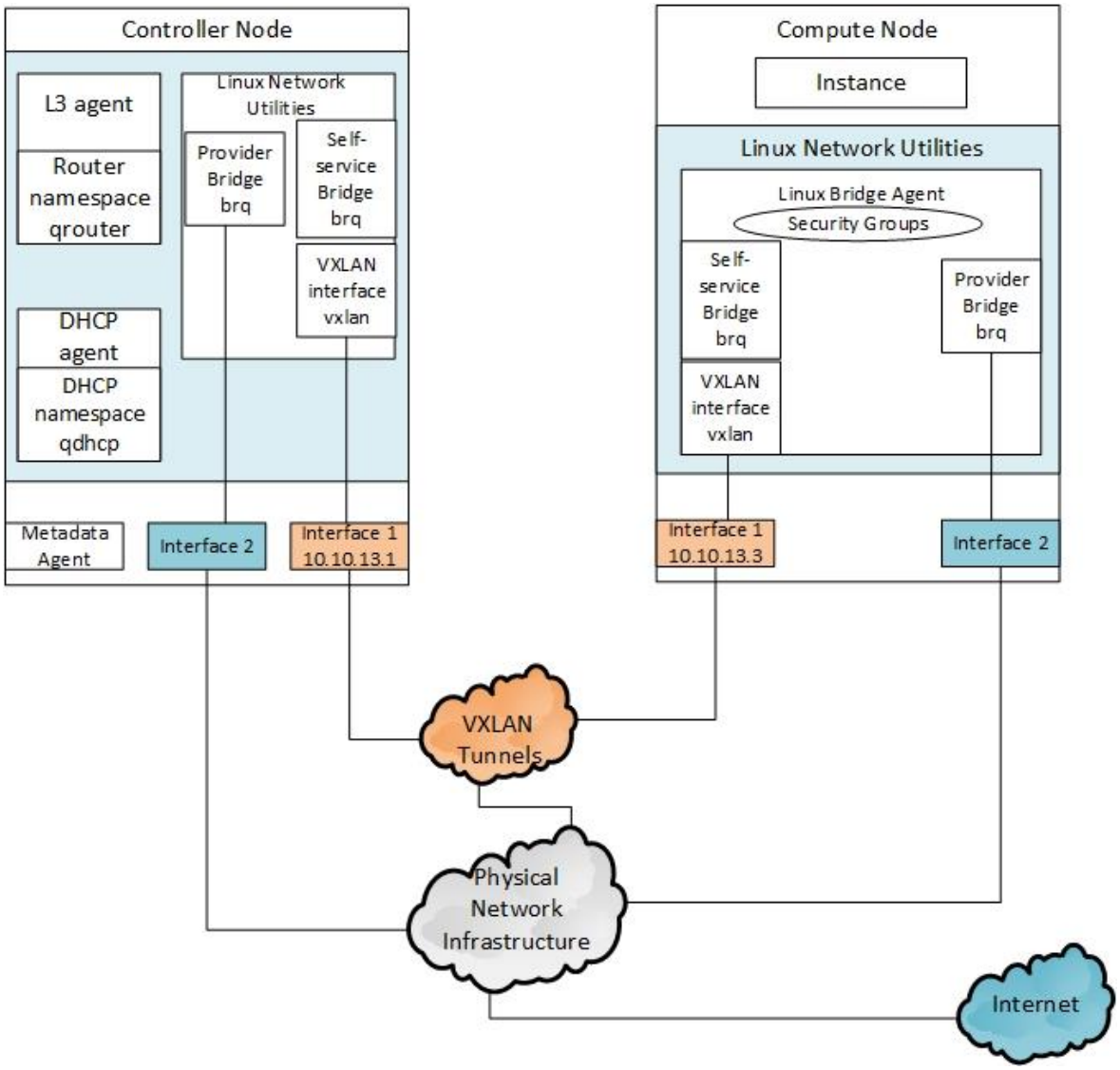

Figure 53 Self-Service Network architecture

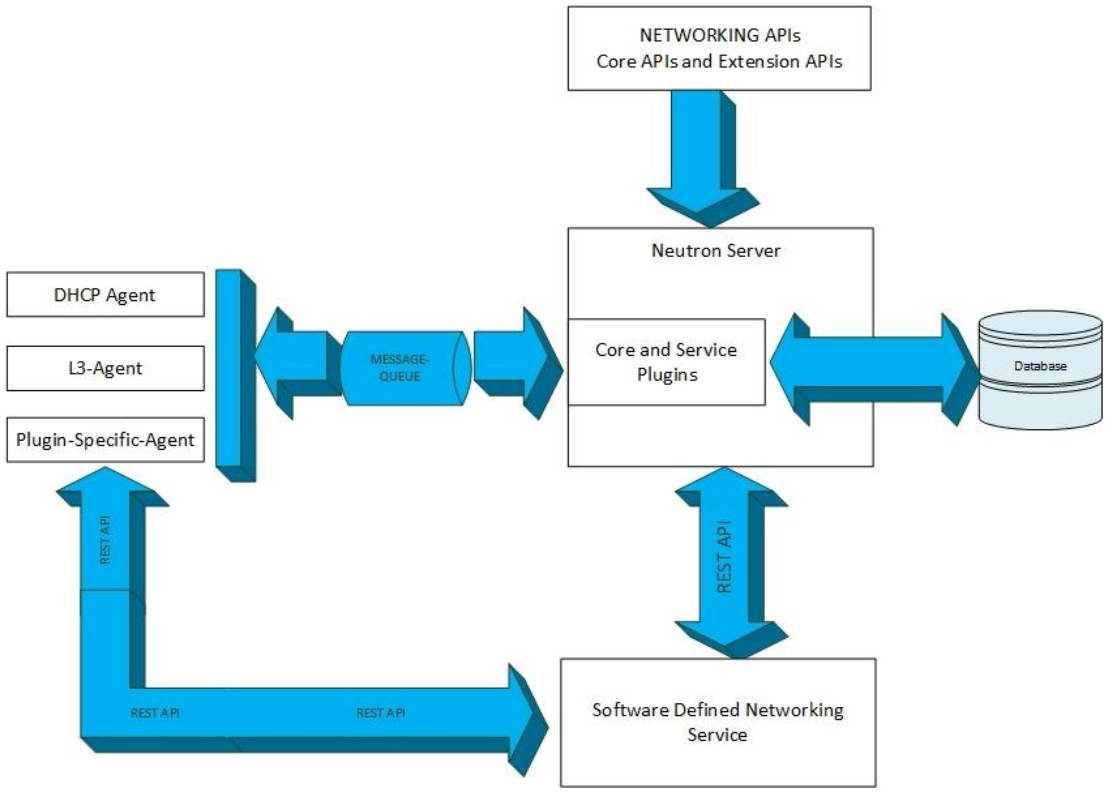

Figure 54 Openstack Neutron Architecture

The elements from Figure 54 will be presented as follows:

1. Plugin Agent

   Those agents are specific to the Neutron plugin being used. They run on the compute node, in this use case implementation is the h2020-server3 node and communicate with the Neutron plugin to manage virtual switches. These agents are optional in many deployments and perform local virtual switch configuration on each hypervisor.

2. Message Queue

   This is an OpenStack component, which is including in Neutron as well, who is used as an advanced message queue protocol (AMQP) for internal communications. In this use case implementation it is used the RabbitMQ who sits between any two internal components of Neutron and allows them to communicate in a loosely coupled fashion (e.g. Neutron components use remote procedure call and it can be identified in the configuration files as RPC parameter).

3. Database

   This logical function block is almost in all plugins, because all of them need a database to maintain a persistent network model.

4. DHCP Agent

   This agent is a part of Neutron and provides DHCP service to tenant networks. It maintains the required DHCP configuration and is the same across all plugins.

5. L3 Agent

This agent is responsible for providing layer 3 and Network Address Translation (NAT) forwarding to gain external access for VMs on tenant networks.

Another plugin used in this enterprise solution is the modular layer 2 core plugin, or just ML2, which is a neutron core plugin. ML2 is a plugin framework allowing OpenStack Neutron to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centers.

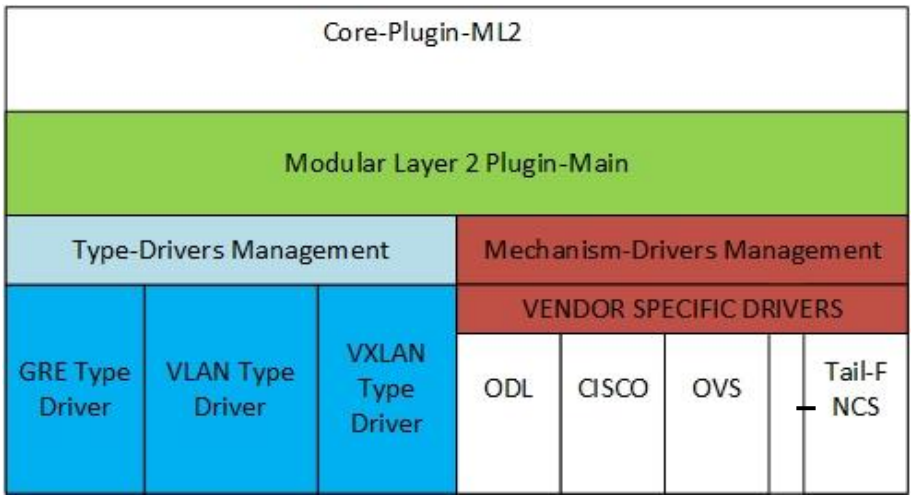

Figure 55 Modular Layer 2 plugin Architecture

This Plugin achieves the modularity through its driver mode. As we can see in the Figure 55, it includes two categories of drivers: type and mechanism. Type drivers (such as flat, VLAN, Generic routing encapsulation (GRE) and VXLAN) define a particular L2 type, where each available network type is managed by a corresponding type driver. The driver maintains type-specific state information and realizes the isolation among the tenant networks long with validation of provider networks.

On the other hand, the mechanism drivers, which are vendor specific (such as open virtual switch, and drivers from ODL, Cisco, NEC, Huawei, Juniper, etc.), based on the enabled type, support creating, updating and deletion of network, subnet and port resources. Is nice to mention the aspect that a hardware vendor might implement a completely new plug-in solution, which might be similar to ML2, or just implement a mechanism driver.

In Figure 53 it is represented the self-service network architecture that was included in the enterprise solution. The neutron software modules in this enterprise implementation are as follows:

- On Compute Node: *neutron-linuxbridge-agent* and *neutron-linuxbridge-cleanup*.

On this node, it is necessary to create the link between the compute OpenStack service and networking OpenStack service as presented below, in Figure 56 [47].

```
h2020@h2020-server3:~$ sudo vi /etc/neutron/neutron.conf
[DEFAULT]
transport_url = rabbit://openstack: ************@h2020-server1
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://h2020-server1:5000
auth_url = http://h2020-server1:35357
```



```
memcached_servers = h2020-server1:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = neutron
password = ************
```

Figure 56 Configuration of Compute Node

In order to send all the Message Queue as presented in Figure 56 from the Compute node, it is necessary to configure this *transport_url* parameter in the *neutron.conf* file. The *keystone_authtoken* section represents the identity service access based on the neutron user and a password that has been obfuscated.

In Figure 57 is represented the configuration file of the *linuxbridge agent [47]*.

```
h2020@h2020-server3:~$ vi /etc/neutron/plugins/ml2/linuxbridge_agent.ini
[DEFAULT]
[agent]
[linux_bridge]
physical_interface_mappings = provider:eno1
[securitygroup]
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
[vxlan]
enable_vxlan = true
local_ip = 192.168.204.17
l2_population = true
```

Figure 57 Configuration file of the linuxbridge agent

- On the Management and Orchestration Node, it wasn't installed any of the neutron software modules.
- On the Controller Node, there were installed the following software modules:
  - *neutron-dhcp-agent [48] (Figure 58)*

The entire configuration regarding the OpenStack DHCP agent is located in the *dhcp_agent.ini* file that provides DHCP services for virtual networks.

```
h2020@h2020-server1:~$ vi /etc/neutron/dhcp_agent.ini
[DEFAULT]
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
 [agent]
availability_zone = nova
[ovs]
```

Figure 58 OpenStack DHCP agent

- *neutron-l3-agent [48] (Figure 59)*

The entire configuration regarding the OpenStack Layer 3 agent is located in the *l3_agent.ini* file that is responsible for packet forwarding including routing from one node to another.

```
h2020@h2020-server1:~$ vi /etc/neutron/l3_agent.ini
[DEFAULT]
interface_driver = linuxbridge
 [agent]
availability_zone = nova
[ovs]
```

Figure 59 OpenStack Layer 3 agent

- *neutron-linuxbridge-agent [49] (Figure 60)*

The entire configuration regarding the OpenStack linux bridge agent is located in the *linuxbridge_agent.ini* file.

```
 h2020@h2020-server1:~$ vi /etc/neutron/plugins/ml2/linuxbridge_agent.ini [DEFAULT]
 [agent]
[linux_bridge]
physical_interface_mappings = provider:eno1
 [securitygroup]
enable_security_group = true
[vxlan]
enable_vxlan = true
local_ip = 192.168.204.15
l2_population = true
```

Figure 60 OpenStack linux bridge agent

- *neutron-linuxbridge-cleanup*
- *neutron-metadata-agent [60] (Figure 61)*

The entire configuration regarding the OpenStack metadata agent is located in the *metadata_agent.ini* file which is responsible for providing configuration information such as credentials to instances.

```
h2020@h2020-server1:~$ vi /etc/neutron/plugins/ml2/linuxbridge_agent.ini
[DEFAULT]
# IP address used by Nova metadata server. (string value)
nova_metadata_ip = 192.168.204.15
metadata_proxy_shared_secret = ************
[agent]
[cache]
```

Figure 61 OpenStack metadata agent

- *neutron-server*

On the table below we are listing all the module functions implemented on the Compute and Controller nodes.

Table 10 All the module functions implemented on the Compute and Controller nodes

Server node	Software module	Functionalities
Compute Node	neutron-linuxbridge-agent	Configures a Linux Bridge to realize neutron's network, port and attachment [50].
	neutron-linuxbridge-cleanup	Automated removal of empty bridges has been disabled to fix a race condition between the Compute (nova) and Networking (neutron) services [50].
Controller Node	neutron-dhcp-agent	Allocates IP addresses to VMs that run on the network [51].
	neutron-l3-agent	Uses the Linux IP stack and ip tables to perform Layer 3 forwarding and NAT.
	neutron-linuxbridge-agent	Configures a Linux Bridge to realize neutron's network, port and attachment [50].
	neutron-linuxbridge-cleanup	Automated removal of empty bridges has been disabled to fix a race condition between the Compute (nova) and Networking (neutron) services [50].
	neutron-metadata-agent	Provides metadata services for instances [49].
	neutron-server	Enforces the network model and IP addressing of each port. Requires indirect access to a persistent database [52].

To configure the compute service to be able to use the networking service, it is necessary to configure the neutron section in the *nova.conf* file (Figure 62) so the link between those two OpenStack services is enabling.

```
h2020@h2020-server1:~$ sudo vi /etc/nova/nova.conf
 [neutron]
url=http://h2020-server1:9696
auth_url = http://h2020-server1:35357
auth_type = password
project_domain_name = Default
user_domain_name = Default
region_name = RegionOne
project_name = service
username = neutron
password = ************
service_metadata_proxy = true
metadata_proxy_shared_secret = ************
```

Figure 62 nova.conf file

In the section above it was presented only the neutron section that is creating the link between the compute and networking services.

8.2.4.2 VIM integration

OpenStack Ocata (v15.0) embodies the Virtual Infrastructure Manager, and the main software modules are:

1. Nova

2. Cinder
3. Keystone
4. Glance

Considering the Nova software module in this enterprise implementation, it was installed:

- On Compute node, nova-compute, nova-manage and cinder-volume were installed.
- On Management and Orchestration Node, it wasn't installed any software modules mentioned previously.
- On Controller node, were installed the following software modules (Figure 63):
  - *nova-api*
  - *nova-conductor*
  - *nova-consoleauth*
  - *nova-novncproxy*
  - *nova-scheduler*

```
h2020@h2020-server1:~$ vi /etc/nova/nova.conf
 [DEFAULT]
transport_url = rabbit://openstack:************@h2020-server1
my_ip = 192.168.204.15
user_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
image_service=nova.image.glance.GlanceImageService
compute_driver=libvirt.LibvirtDriver
allow_resize_to_same_host=true
instance_name_template=instance-%08x
dhcpbridge_flagfile=/etc/nova/nova.conf
public_interface=eno1
vlan_interface=eno1
flat_network_bridge=br100
flat_interface=eno1
dhcpbridge=/usr/bin/nova-dhcpbridge
force_dhcp_release=true
state_path=/var/lib/nova
enabled_apis=osapi_compute,metadata
network_manager=nova.network.manager.FlatDHCPManager
dhcpbridge_flagfile=/etc/nova/nova.conf
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
[api]
auth_strategy = keystone
[api_database]
connection = mysql+pymysql://nova: ************@h2020-server1/nova_api
[cells]
enable=False
mute_weight_multiplier=-10.0
ram_weight_multiplier=10.0
offset_weight_multiplier=1.0
scheduler_weight_classes=nova.cells.weights.all_weighers
[cinder]
os_region_name=RegionOne
[database]
```



```
connection = mysql+pymysql://nova: ************@h2020-server1/nova
[vnc]
enable = True
vncserver_listen = $my_ip
vncserver_proxyclient_address = $my_ip
[wsgi]
api_paste_config=/etc/nova/api-paste.ini
```

Figure 63 Controller node config

- *cinder-scheduler (Figure 64)*

```
root@h2020-server1:/etc/cinder# vi cinder.conf
[DEFAULT]
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_confg = /etc/cinder/api-paste.ini
iscsi_helper = tgtadm
volume_name_template = volume-%s
volume_group = cinder-volumes
verbose = True
debug = True
auth_strategy = keystone
state_path = /var/lib/cinder
lock_path = /var/lock/cinder
volumes_dir = /var/lib/cinder/volumes
tranport_url = rabbit://openstack: ************@h2020-server1
my_ip = 192.168.204.15
[database]
# ...
connection = mysql+pymysql://cinder: ************@h2020-server1/cinder
[keystone_authtoken]
# ...
auth_uri = http://h2020-server1:5000
auth_url = http://h2020-server1:35357
memcached_servers = h2020-server1:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = cinder
password = cinder
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
[lvm]
#iscsi_helper = tgtadm
#volume_group = cinder-volumes
#iscsi_ip_address = 192.168.204.15
#volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
#volume_dir = /var/lib/cinder/volumes
#iscsi_protocol = iscsi
#volume_backend_name = lvm
```

```
#volume_clear = none
[oslo_messaging_notifications]
#...
driver = messagingv2
```

Figure 64 cinder-scheduler

- *keystone (Figure 65)*

```
root@h2020-server1:/etc/keystone# vi keystone.conf
[DEFAULT]
admin_token = abcd1234
debug = true
verbose = true
log_file = /var/log/keystone/keystone.log
[database]
connection = mysql+pymysql://keystone: ************@h2020-server1/keystone
```

Figure 65 keystone

- *glance-api (Figure 66)*

```
root@h2020-server1:/etc/glance# vi glance-api.conf
[DEFAULT]
debug = True
[database]
sqlite_db = /var/lib/glance/glance.sqlite
connection = mysql+pymysql://glance: ************@h2020-server1/glance
[glance_store]
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
auth_uri = http://h2020-server1:5000
auth_url = http://h2020-server1:35357
memcached_servers = h2020-server1:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = ************
[paste_deploy]
flavor = keystone
```

Figure 66 glance-api

- *glance-registry (Figure 67)*

```
root@h2020-server1:/etc/glance# vi glance-registry.conf
[DEFAULT]
debug = True
verbose = true
transport_url = rabbit://openstack:************@h2020-server1
[database]
```



```
connection = mysql+pymysql://glance:************@h2020-server1/glance
sqlite_db = /var/lib/glance/glance.sqlite
backend = sqlalchemy
[keystone_authtoken]
auth_uri = http://h2020-server1:5000
auth_url = http://h2020-server1:35357
memcached_servers  = h2020-server1:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = ************
[matchmaker_redis]
[oslo_messaging_amqp]
[oslo_messaging_kafka]
[oslo_messaging_notifications]
driver = messagingv2
[oslo_messaging_rabbit]
[oslo_messaging_zmq]
[oslo_policy]
[paste_deploy]
[profiler]
```

Figure 67 glance-registry

On the Table below are listed all the module functions implemented on the Compute and Controller nodes.

Table 11 Module functions implemented on the Compute and Controller nodes

Server node	Software module	Functionalities
Compute Node	nova-compute	It handles all processes relating to instances [53].
	nova-manage	Controls cloud computing instances by managing shell selection, vpn connections, and floating IP address configuration [54].
	cinder-volume	Responds to requests to read from and write to a block storage database to maintain state by interacting with other processes, like cinder-scheduler, through a message queue, and to act directly upon block-storage providing hardware or software [55].
Controller Node	nova-api	Serves the metadata and compute APIs in separate greenthreads [56].
	nova-conductor	Provides coordination and database query support for nova service [57].
	nova-consoleauth	Provides authentication for nova consoles [58].
	nova-novncproxy	Websocket proxy that is compatible with openstack nova noVNC consoles [59].

	nova-scheduler	Determines how to dispatch compute requests [60].
	cinder-scheduler	Schedule volume on a different back-end from a set of volumes [61].
	Keystone	Provides authentication credential validation and data about users and groups [62].
	glance-api	Provides services for discovering and retrieving VM images [63].
	glance-registry	Provides services for registering the VM images [63].

The interworking of all the software modules presented previously and their interdependency is presented in the Figure 68. The Figure is inspired by one from [64] but is adapted accordingly with this scenario.

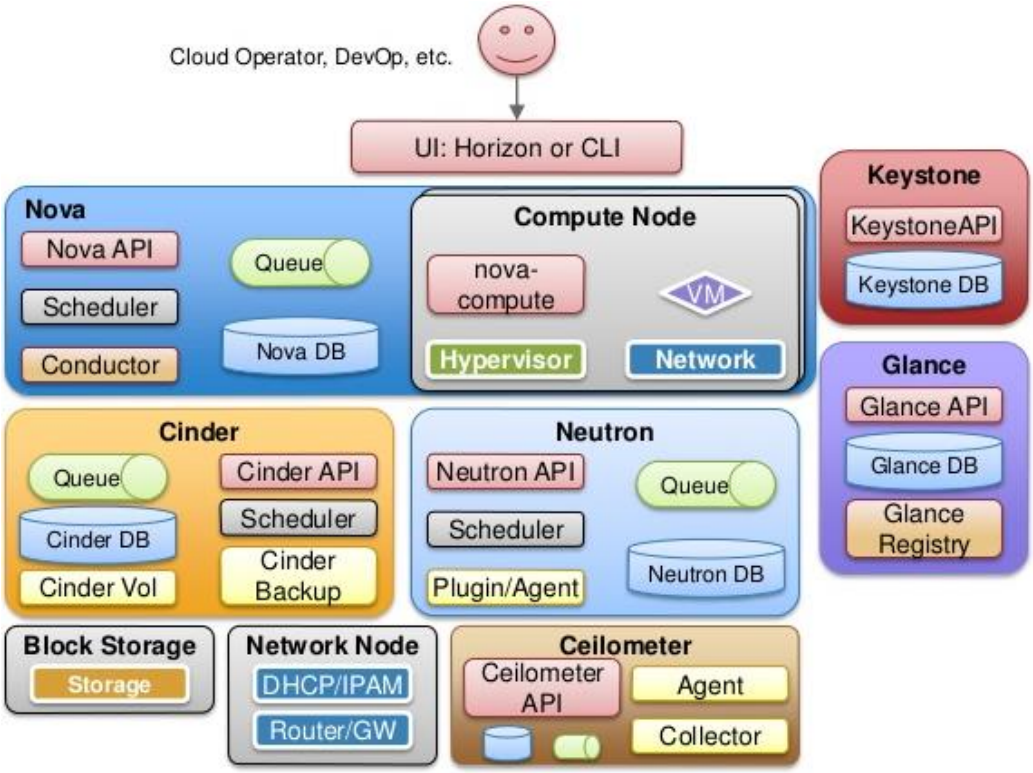

Figure 68 The interdependency of the OpenStack Ocata software modules [64]

The information presented in the Annex D represent the OpenStack software modules that were installed according to the Table 11 and it get as inputs all the configuration file highlighted above in the text.

As a conclusion of this subchapter, the implementation of the VIM and SDN based on OpenStack services is working smoothly to provide all the services necessary to facilitate the access of the enterprise platform to the lightning modules.

8.2.5 Prototyped LCM

The OpenStack Ocata module responsible with LCM of enterprise services and apps is Murano. Running on top of the aforementioned Openstack cloud, Murano will manage the creation, instantiation, updates and termination of Smart City apps and services.

Murano is composed of four services:

- murano-engine – workflow execution organizer;
- murano-API – programmatic interface used for interaction with Murano;
- murano-client CLI – CLI used for managing environments, packages and categories;
- murano-dashboard – Horizon add-on for Murano service.

While Murano assures the life cycle of apps, it is integrated with Heat in order to orchestrate the physical resources. This integration is based on REST APIs between Murano-engine and Heat.

In order to create a new Smart City app composed of multiple VMs, it was defined the service metadata. These metadata specify the properties and the necessary steps for deploying the Smart City service. Murano provides the possibility to define the smart city apps under an app catalogue.

Murano services were installed in Management and Orchestration Node and are listed in Figure 69:

```
h2020@h2020-server1:~$ openstack endpoint list --service murano
+----------------------------------+-----------+--------------+-------------------+---------+-----------+---------------------------+
| ID                               | Region    | Service Name | Service Type      | Enabled | Interface | URL                       |
+----------------------------------+-----------+--------------+-------------------+---------+-----------+---------------------------+
| 362006838e1344f988152069951c9a49 | RegionOne | murano       | application-catalog | True    | admin     | http://h2020-server2:8082 |
| 7179b2b972e647b29dfa7f9144dfef99 | RegionOne | murano       | application-catalog | True    | internal  | http://h2020-server2:8082 |
| bec3fd588db74db99d054952f4f8be0f | RegionOne | murano       | application-catalog | True    | public    | http://h2020-server2:8082 |
+----------------------------------+-----------+--------------+-------------------+---------+-----------+---------------------------+
```

Figure 69 Murano endpoints

8.3 Prototyped services and applications

As described in section 3 of this document regarding Smart City apps and services, in this current section it will be described how IoT platform has been configured such that can deliver and manage smart city services, in this case smart lighting service.

After all components of enterprise cloud infrastructure have been configured and all physical elements installed in place must ensure E2E connectivity so as the service can deliver at the expected level.

The starting point to create the entire service is to add in Thingsboard.io Platform (Open Source IoT Platform in this scenario) a new customer, a new tenant and user (or users) such that can deploy and manage every element (lamps, controllers, gateways) as can be seen from Figure 70.

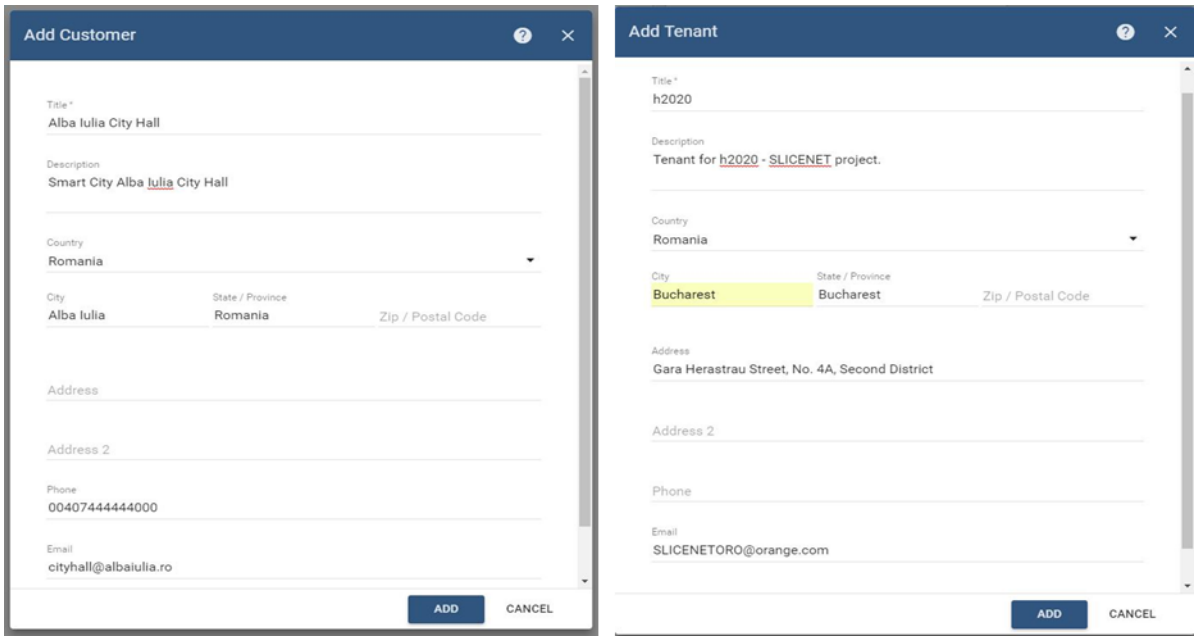

Figure 70 Creation process of customer and tenant

Every customer available in the platform has created a tenant who owns all connected devices and the data produced. Based on this, a service offered to a customer can be distinguished from another. (e.g. smart lighting service for City of Alba Iulia versus smart lighting service for City of Bucharest). All settings made below, has been made using SLICENETORO user.

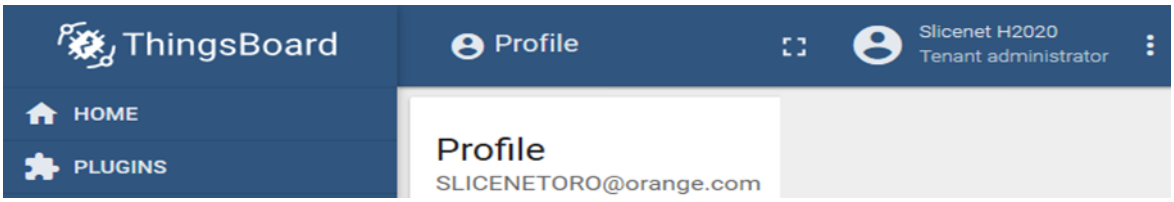

Figure 71 SliceNetORO user account

After all of this has been set up, all devices have to be declared in IoT platform panel. In the Figure 72, are listed six lighting elements (two of them are using LTE-M and the other three LoRa WAN and one 5G-ready) connected throw this platform. All devices are associated to the customer previous defined.

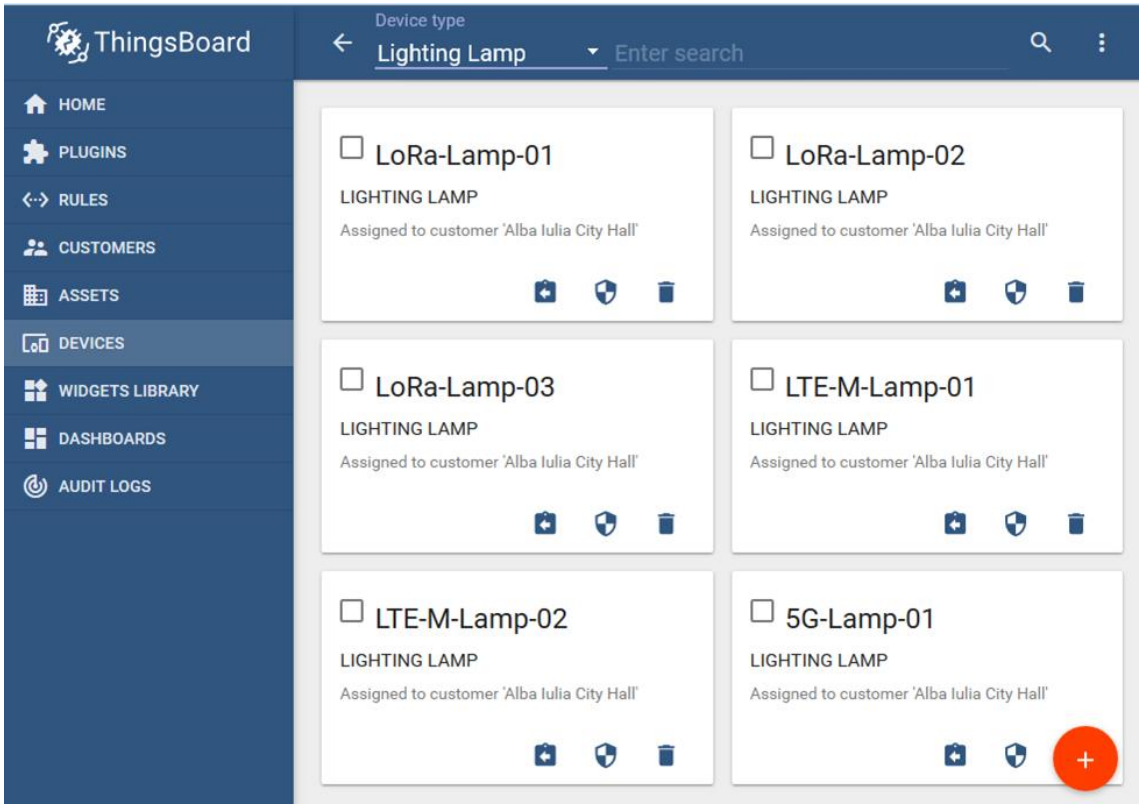

Figure 72 List of devices

To pair every device, an authentication method must be used: based on access token or based on X.509 certificate. Within this use case, an access token based authentication has been used (Figure 73).

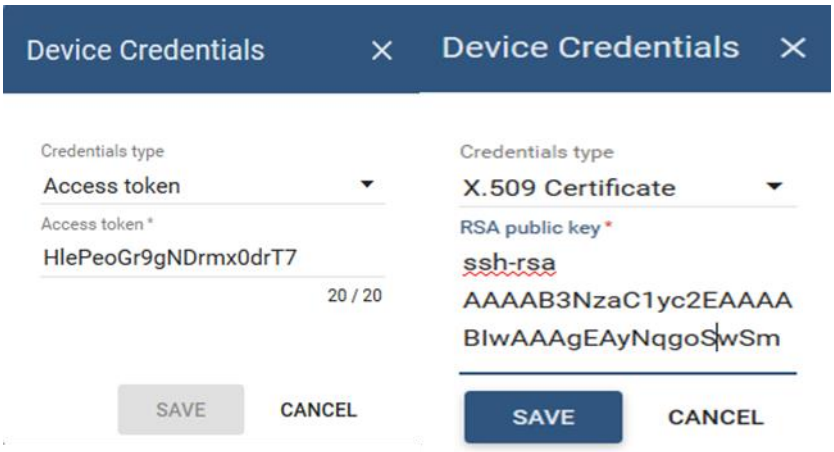

Figure 73 Authentication method (Token left; certificate right)

After all the lamps has been connected and correctly configured, they started to send first messages – the "hello message". The payload of this message contains all technical parameters and attributes regarding the client itself (the client in this understanding is the device). In the Figure 74, all device attributes are listed: name of device, type, panel, pillar, address and so on. For better and compact visualization has been created dashboards for every important part from device configuration menu.

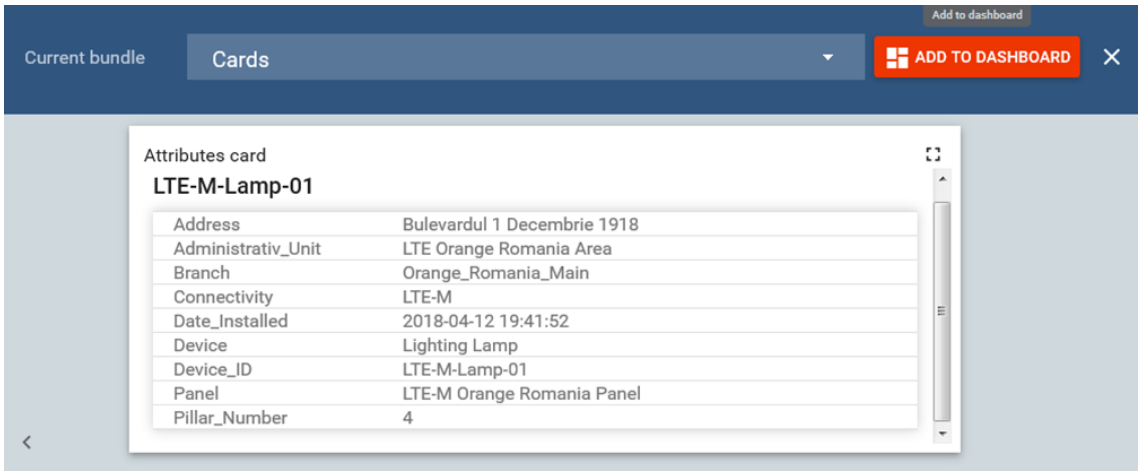

Figure 74 Parameters of a LTE-M lamp

A list with all dashboards generated for operation and maintenance of entire service is described in the figures (Figure 75 and Figure 76) below:

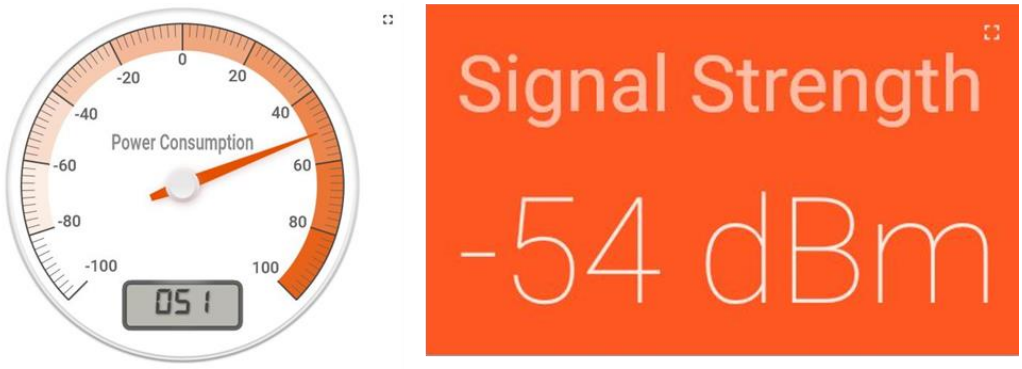

Figure 75 Power consuption and signal strenght of LTE-M Lamp 01

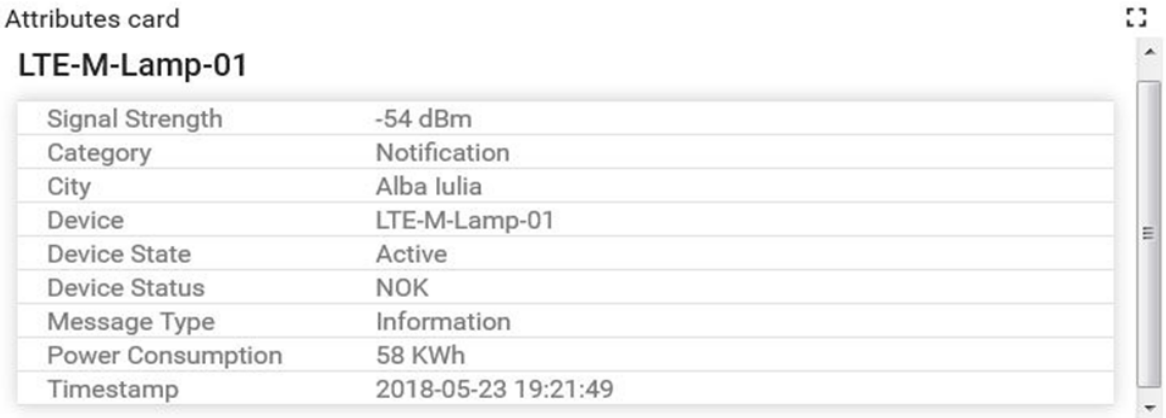

Figure 76 General status of LTE-M Lamp 01

The following dashboard from Figure 77 has been generated for public access and the city hall can expose the data to citizens.

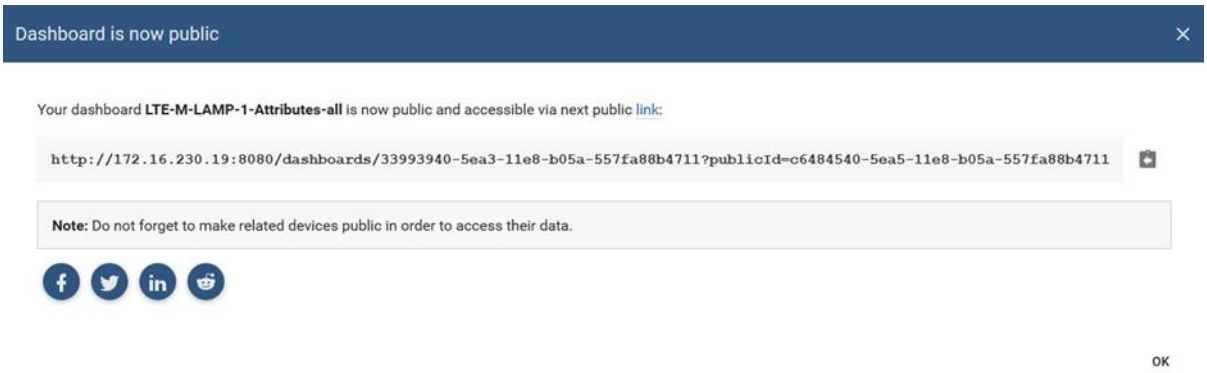

Figure 77 Dashboard generated for public access

9 Conclusions

The overall aim of this deliverable is to define, within the novel 5G E2E slicing perspective, the concepts and detailed needs of transformation to a 5G ready enterprise infrastructure. The proposed infrastructure prototype is adapted to the SliceNet system definition aspects, from vertical sector needs, to the architectural perspective, with specific elements defined in the control and management plane tasks. The proposed implementation, based on open-source tools and apps can be easily integrated into the proposed 5G slicing friendly infrastructure and can easily benefit from the further developments from other tasks, relevant achievements related to the WP5 cognition, vertical-informed QoE sensors and actuators, slice management and FCAPS functions.

One of the main outputs of the deliverable is defining the transition from the traditional enterprise approach to a new 5G implementation model, a virtualized and programmable 5G ready infrastructure, as there are enormous technical expectations from many business vertical perspectives, with huge financial growth impact.

The second important output of the deliverable is related to the system automation, the first 5G enterprise prototype model proposing to remove the limitations of the current network infrastructure, inflexible, costly and hard scalable, by deploying software 5G slice network infrastructure, including controlling and orchestration of resources availability, also by extending the 5G slicing-friendly concepts to the enterprise border.

As 5G will bring major transformation to the networks architecture, including the enterprise segment, the vertical sector services running over the novel 5G environment will face also a major transformation. The technological model proposed will maximize the potential of the new services to be deployed within the 5G context, over advanced software cognitive networks.

SliceNet is proposing an intelligent 5G slicing approach, by offering QoS and QoE capabilities to the end consumers, the verticals, as an E2E novel multi-domain implementation framework, cross-plane orchestrator, including innovative elements of E2E slicing, QoE modelling, one-stop-shop API flexible solution to facilitate the creation of the new services, design and customization capabilities offered to the vertical, as a perspective of P&P control function.

The prototype proposed by this deliverable is designed to be adapted to any open source orchestration and automation implementation and is intelligent designed to be easily integrated with any 5G SliceNet software blocks, as cognitive management, QoE Optimizer, sensors and actuators, one-stop API and P&P control.

The model is intended to be demonstrated into a 5G Smart City scenario, as it is fulfilling the specific smart lighting requirements, supported by the 5G fast time to market, efficiency, flexibility, environmental friendly, and secured mMTC IoT use cases.

References

[1] SliceNet, Deliverable 2.2 -Overall Architecture and Interfaces Definition, Jan 2018

[2] SliceNet, Deliverable 3.1 - Design and Prototyping of SliceNet Virtualised Mobile Edge, Apr. 2018

[3] SliceNet project, https://SliceNet.eu/

[4] SliceNet, Deliverable 2.1 - Vertical Sector Requirements Analysis and Use Case Definition, Oct. 2017

[5] SliceNet, Deliverable 2.3 -Control Plane System Definition, APIs and Interfaces, Apr. 2018

[6] SliceNet, Deliverable 2.4- Management Plane System Definition, APIs and Interfaces, May 2018

[7] http://www.etsi.org/deliver/etsi_ts/136200_136299/136213/14.02.00_60/ts_136213v140200p.pdf

[8] Network Function Virtualization Architectural Framework, ETSI GS NFV 002: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[9] Network Function Virtualization Management and Orchestration, ETSI GS NFV-MAN 001: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[10] https://docs.openstack.org/heat/pike/man/heat-api-cfn.html

[11] https://docs.openstack.org/heat/pike/man/heat-engine.html

[12] https://docs.openstack.org/ceilometer/latest/admin/telemetry-system-architecture.html

[13] https://docs.openstack.org/nova/pike/cli/nova-compute.html

[14] https://specs.openstack.org/openstack/cinder-specs/specs/liberty/non-eventlet-wsgi-app.html

[15] https://docs.openstack.org/ocata/install-guide-ubuntu/common/get-started-networking.html

[16] http://searchcloudcomputing.techtarget.com/definition/private-cloud

[17] http://searchcloudcomputing.techtarget.com/definition/public-cloud

[18] https://www.openstack.org/software/

[19] http://cdn.ttgtmedia.com/searchCloudComputing/downloads/OpenStack+Guide.pdf

[20] https://www.redhat.com/en/topics/virtualization/what-is-KVM

[21] Michael Solberg , Ben Silverman , Openstack for Architects, Packt Publishing (February 6, 2017), ISBN-13: 978-1784395100

[22] https://docs.openstack.org/tacker/latest/install/openstack_vim_installation.html

[23] https://docs.openstack.org/tacker/latest/contributor/vnfd_template_description.html

[24] Technical Report: "RAN slicing runtime system for flexible and dynamic service execution environment" , http://www.eurecom.fr/fr/publication/5351/detail/ran-slicing-runtime-system-for-flexible-and-dynamic-service-execution-environment?popup=1

[25] http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[26] http://www.etsi.org/deliver/etsi_ts/128500_128599/128526/14.00.00_60/ts_128526v1 40000p.pdf

[27] http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/010/02.03.01_60/gs_NFV-IFA010v020301p.pdf

[28] http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[29] OSM Whitepaper, Release THREE, Feb. 2018. [Online]. Available: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.pdf

[30] JUJU, https://jujucharms.com

[31] RIFTWARE, https://www.riftio.com/tag/rift-ware/

[32] TACKER , https://wiki.openstack.org/wiki/Tacker

[33] ONAP, http://onap.org/

[34] ONAP2, Whitepaper, [Online]. Available: https://www.onap.org/wp-content/uploads/sites/20/2017/12/ONAP_CaseSolution_Architecture_120817_FNL.pdf

[35] ECOMP+OPEN-O , https://www.onap.org/category/announcement

[36] OPNFV, https://www.opnfv.org

[37] SONATA, www.sonata-nfv.eu

[38] 5GEx Project, http://www.5gex.eu/

[39] OPENBATON Whitepaper, [Online]. Available: https://openbaton.github.io/documentation/

[40] K. Katsalis, N. Nikaein, and A. Huang, "JOX: an event-driven orchestrator for 5G network slicing", in Proc. IEEE/IFIP Network Operations and Management Symposium, 2018.

[41] SliceNet work package -WP2: SliceNet System Definition

[42] SliceNet work package -WP4: 5G Multi-Domain Slice Control Plane

[43] SliceNet work package -WP5:Cognitive, Service-Level QoE Management

[44] SliceNet work package -WP7: Cross-Plane Orchestration and Use Cases Prototyping

[45] SliceNet work package -WP8: System Integration and Demonstration

[46] Michael Solberg, Ben Silverman, Openstack for architects, 978-1784395100

[47] https://docs.openstack.org/ocata/install-guide-ubuntu/neutron-compute-install.html#neutron-compute-compute

[48] https://docs.openstack.org/ocata/install-guide-ubuntu/neutron-controller-install-option2.html

[49] https://docs.openstack.org/ocata/install-guide-ubuntu/neutron-controller-install.html#neutron-controller-metadata-agent

[50] https://wiki.openstack.org/wiki/Neutron-Linux-Bridge-Plugin

[51] http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/section_adv_cfg_dhcp_agent.html

[52] https://docs.openstack.org/security-guide/networking/architecture.html

[53] https://docs.openstack.org/nova/pike/cli/nova-compute.html

[54] https://docs.openstack.org/nova/pike/cli/nova-manage.html

[55] http://blog.flux7.com/blogs/openstack/tutorial-what-is-cinder-and-how-to-install-and-use-it

[56] https://docs.openstack.org/nova/pike/cli/nova-api.html

[57] https://docs.openstack.org/nova/latest/cli/nova-conductor.html

[58] https://docs.openstack.org/nova/pike/cli/nova-consoleauth.html

[59] https://docs.openstack.org/nova/pike/cli/nova-novncproxy.html

[60] https://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html

[61] https://docs.openstack.org/cinder/pike/scheduler-filters.html

[62] https://docs.openstack.org/keystone/latest/getting-started/architecture.html

[63] https://launchpad.net/glance

[64] https://www.slideshare.net/mirantis/openstack-architecture-43160012

[65] Alok Shrivastwa, Suntil Sarat, Kevin Jackson, Cody Bunch, Egle Sigle, Tony Campbell, Openstack: building a cloud environment", Publisher Packt Publishing, September 19, 2016, ASIN: B01M0IREB3

Annex A TOSCA template descriptors

NSD [6]	VNFD [6]	VDU [6]	Vnfc [7]	VNFFGD [6]	NFP [8]
tosca_definitions_version: tosca_simple_profile_for_ nfv_1_0_0 imports: - VNFD1 - VNFD2 topology_template: node_templates: VNF1: type: tosca.nodes.nfv.VNF1 requirements: - virtualLink1: VL1 - virtualLink2: VL2 VNF2: type: tosca.nodes.nfv.VNF2 VL1: type: tosca.nodes.nfv.VL properties: network_name: net0 vendor: tacker VL2: type: tosca.nodes.nfv.VL properties: network_name: net_mgmt vendor: tacker	tosca_definitions_version: tosca_simple_profile_for_ nfv_1_0_0 description: VNF TOSCA template with input parameters metadata: template_name: sample- tosca-vnfd topology_template: node_templates: VDU1: type: tosca.nodes.nfv.VDU.Tack er properties: image: cirros-0.3.5- x86_64-disk flavor: m1.tiny availability_zone: nova mgmt_driver: noop config: | param0: key1 param1: key2 CP1: type: tosca.nodes.nfv.CP.Tacker	topology_template: node_templates: VDU1: type: tosca.nodes.nfv.VDU.Tack er properties: image: cirros-0.3.5- x86_64-disk availability_zone: nova capabilities: nfv_compute: properties: disk_size: 10 GB mem_size: 2048 MB num_cpus: 2	topology_template: node_templates: firewall_vnfc: type: tosca.nodes.nfv.VNFC.Tac ker requirements: - host: VDU1 interfaces: Standard: create: install_vnfc.sh	policy: type: ACL criteria: - name: block_tcp classifier: network_src_port_id: 640dfd77-c92b-45a3- b8fc-22712de480e1 destination_port_range: 80-1024 ip_proto: 6 ip_dst_prefix: 192.168.1.2/24 - name: block_udp classifier: network_src_port_id: 640dfd77-c92b-45a3- b8fc-22712de480eda destination_port_range: 80-1024 ip_proto: 17 ip_dst_prefix: 192.168.2.2/24	tacker nfp-list tacker nfp-show <nfp id> tacker chain-list tacker chain-show <chain id> tacker classifier-list tacker classifier-show <classifier id>

	properties: management: True anti_spoofing_protection: false requirements: - virtualLink: node: VL1 - virtualBinding: node: VDU1 CP2: type: tosca.nodes.nfv.CP.Tacker properties: anti_spoofing_protection: false requirements: - virtualLink: node: VL2 - virtualBinding: node: VDU1 CP3: type: tosca.nodes.nfv.CP.Tacker properties: anti_spoofing_protection: false requirements: - virtualLink: node: VL3 - virtualBinding: node: VDU1				

	VL1: type: tosca.nodes.nfv.VL properties: network_name: net_mgmt vendor: Tacker VL2: type: tosca.nodes.nfv.VL properties: network_name: net0 vendor: Tacker VL3: type: tosca.nodes.nfv.VL properties: network_name: net1 vendor: Tacker				

Annex B The interface configuration

Compute node role	Interface configuration
Controller node	# This file describes the network interfaces available on your system # and how to activate them. For more information, see interfaces(5). source /etc/network/interfaces.d/* auto lo iface lo inet loopback auto eno1 iface eno1 inet static address 192.168.204.15 netmask 255.255.255.192 network 192.168.204.0 broadcast 192.168.204.63 gateway 192.168.204.1 auto eno3 iface eno3 inet static address 10.10.12.1 netmask 255.255.255.192 network 10.10.12.0 broadcast 10.10.12.255 auto eno4 iface eno4 inet static address 10.10.13.1 netmask 255.255.255.192 network 10.10.13.0 broadcast 10.10.13.255
Management & Orchestration node	# This file describes the network interfaces available on your system # and how to activate them. For more information, see interfaces(5). source /etc/network/interfaces.d/* auto lo iface lo inet loopback auto eno1 iface eno1 inet static address 192.168.204.16 netmask 255.255.255.192 network 192.168.204.0 broadcast 192.168.204.63 gateway 192.168.204.1 dns-nameservers 192.168.204.16 auto eno2 iface eno2 inet static address 10.10.23.2 netmask 255.255.255.0 network 10.10.23.0 broadcast 10.10.23.255 auto eno3 iface eno3 inet static address 10.10.12.2 netmask 255.255.255.0 network 10.10.12.0 broadcast 10.10.12.255
Compute node	# The loopback network interface auto lo iface lo inet loopback auto eno1

	iface eno1 inet static address 192.168.204.17 netmask 255.255.255.192 network 192.168.204.0 broadcast 192.168.204.63 gateway 192.168.204.1 dns-nameservers 8.8.8.8 auto eno2 iface eno2 inet static address 10.10.23.3 netmask 255.255.255.192 network 10.10.23.0 broadcast 10.10.23.255 auto eno3 iface eno3 inet static address 172.18.60.37 netmask 255.255.252.0 network 172.18.60.0 broadcast 10.10.12.255 auto eno4 iface eno4 inet static auto eno4 iface eno4 inet static address 10.10.13.3 netmask 255.255.255.192 network 10.10.13.0 broadcast 10.10.13.255

Annex C Annex C Configure name resolution

Compute node role	Interface configuration
Controller node	127.0.0.1 localhost 192.168.204.15 h2020-server1 192.168.204.16 h2020-server2 192.168.204.17 h2020-server3 10.10.12.1 controller 10.10.12.2 mano 10.10.13.1 controller 10.10.13.3 compute # The following lines are desirable for IPv6 capable hosts ::1 localhost ip6-localhost ip6-loopback ff02::1 ip6-allnodes ff02::2 ip6-allrouters
Management & Orchestration node	127.0.0.1 localhost 192.168.204.16 h2020-server2 192.168.204.15 h2020-server1 192.168.204.17 h2020-server3 10.10.23.2 mano 10.10.23.3 compute 10.10.12.2 mano 10.10.12.1 controller # The following lines are desirable for IPv6 capable hosts ::1 localhost ip6-localhost ip6-loopback ff02::1 ip6-allnodes ff02::2 ip6-allrouters
Compute node	127.0.0.1 localhost 192.168.204.15 h2020-server1 192.168.204.16 h2020-server2 192.168.204.17 h2020-server3 192.168.204.18 h2020-server4 10.10.23.3 compute 10.10.23.2 mano 172.18.60.37 vm_gw 10.10.13.3 compute 10.10.13.1 controller # The following lines are desirable for IPv6 capable hosts ::1 localhost ip6-localhost ip6-loopback ff02::1 ip6-allnodes ff02::2 ip6-allrouters

Annex D Prototyped SDN and VIM integration

Server node	Software module	Configurations

Compute Node	neutron-linuxbridge-agent	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/neutron
		chown neutron:neutron /var/$i/neutron
		done
		for i in log ; do
		mkdir -p /var/$i/neutron
		chown neutron:adm /var/$i/neutron
		done
		end script
		
		script
		[-x "/usr/bin/neutron-linuxbridge-agent"] || exit 0
		DAEMON_ARGS="--config-file=/etc/neutron/plugins/ml2/linuxbridge_agent.ini"
		CONFIG_FILE="/etc/neutron/neutron.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-linuxbridge-agent.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/neutron \

		--chuid neutron:neutron --make-pidfile --pidfile /var/run/neutron/neutron-linuxbridge-agent.pid \ --exec /usr/bin/neutron-linuxbridge-agent -- ${DAEMON_ARGS} end script

	neutron-linuxbridge-cleanup	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/neutron
		chown neutron:neutron /var/$i/neutron
		done
		for i in log ; do
		mkdir -p /var/$i/neutron
		chown neutron:adm /var/$i/neutron
		done
		end script
		
		script
		[-x "/usr/bin/neutron-linuxbridge-agent"] || exit 0
		DAEMON_ARGS="--config-file=/etc/neutron/plugins/ml2/linuxbridge_agent.ini"
		CONFIG_FILE="/etc/neutron/neutron.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-linuxbridge-agent.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/neutron \
		--chuid neutron:neutron --make-pidfile --pidfile /var/run/neutron/neutron-linuxbridge-agent.pid \

		--exec /usr/bin/neutron-linuxbridge-agent -- ${DAEMON_ARGS} end script
Controller Node	neutron-dhcp-agent	start on runlevel [2345] stop on runlevel [!2345] respawn chdir /var/run pre-start script mkdir -p /var/run/neutron chown neutron:root /var/run/neutron # Check to see if openvswitch plugin in use by checking # status of cleanup upstart configuration if status neutron-ovs-cleanup; then start wait-for-state WAIT_FOR=neutron-ovs-cleanup WAIT_STATE=running WAITER=neutron-dhcp-agent fi end script exec start-stop-daemon --start --chuid neutron --exec /usr/bin/neutron-dhcp-agent -- --config-file=/etc/neutron/neutron.conf --config-file=/etc/neutron/dhcp_agent.ini --log-file=/var/log/neutron/dhcp-agent.log
	neutron-l3-agent	start on runlevel [2345] stop on runlevel [!2345] respawn chdir /var/run pre-start script mkdir -p /var/run/neutron chown neutron:root /var/run/neutron # Check to see if openvswitch plugin in use by checking # status of cleanup upstart configuration if status neutron-ovs-cleanup; then start wait-for-state WAIT_FOR=neutron-ovs-cleanup WAIT_STATE=running WAITER=neutron-l3-agent fi end script exec start-stop-daemon --start --chuid neutron --exec /usr/bin/neutron-l3-agent -- \

		--config-file=/etc/neutron/neutron.conf --config-file=/etc/neutron/l3_agent.ini \ --config-file=/etc/neutron/fwaas_driver.ini --log-file=/var/log/neutron/l3-agent.log

	neutron-linuxbridge-agent	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/neutron
		chown neutron:neutron /var/$i/neutron
		done
		for i in log ; do
		mkdir -p /var/$i/neutron
		chown neutron:adm /var/$i/neutron
		done
		end script
		
		script
		[-x "/usr/bin/neutron-linuxbridge-agent"] || exit 0
		DAEMON_ARGS="--config-file=/etc/neutron/plugins/ml2/linuxbridge_agent.ini"
		CONFIG_FILE="/etc/neutron/neutron.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-linuxbridge-agent.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/neutron \
		--chuid neutron:neutron --make-pidfile --pidfile /var/run/neutron/neutron-linuxbridge-agent.pid \
		--exec /usr/bin/neutron-linuxbridge-agent -- ${DAEMON_ARGS}

		end script

	neutron-linuxbridge-cleanup	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/neutron
		chown neutron:neutron /var/$i/neutron
		done
		for i in log ; do
		mkdir -p /var/$i/neutron
		chown neutron:adm /var/$i/neutron
		done
		end script
		
		script
		[-x "/usr/bin/neutron-linuxbridge-agent"] || exit 0
		DAEMON_ARGS="--config-file=/etc/neutron/plugins/ml2/linuxbridge_agent.ini"
		CONFIG_FILE="/etc/neutron/neutron.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-linuxbridge-agent.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/neutron \
		--chuid neutron:neutron --make-pidfile --pidfile /var/run/neutron/neutron-linuxbridge-agent.pid \
		--exec /usr/bin/neutron-linuxbridge-agent -- ${DAEMON_ARGS}
		end script

	neutron-metadata-agent	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/neutron
		chown neutron:neutron /var/$i/neutron
		done
		for i in log ; do
		mkdir -p /var/$i/neutron
		chown neutron:adm /var/$i/neutron
		done
		end script
		
		script
		[-x "/usr/bin/neutron-metadata-agent"] || exit 0
		DAEMON_ARGS="--config-file=/etc/neutron/metadata_agent.ini"
		CONFIG_FILE="/etc/neutron/neutron.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-metadata-agent.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/neutron \
		--chuid neutron:neutron --make-pidfile --pidfile /var/run/neutron/neutron-metadata-agent.pid \
		--exec /usr/bin/neutron-metadata-agent -- ${DAEMON_ARGS}
		end script

	neutron-server	start on runlevel [2345]

		stop on runlevel [!2345]
		
		respawn
		
		chdir /var/run
		
		pre-start script
		  mkdir -p /var/run/neutron
		  chown neutron:root /var/run/neutron
		end script
		
		script
		  [-x "/usr/bin/neutron-server"] || exit 0
		  [-r /etc/default/openstack] && . /etc/default/openstack
		  [-r /etc/default/neutron-server] && . /etc/default/neutron-server
		  [-r "$NEUTRON_PLUGIN_CONFIG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$NEUTRON_PLUGIN_CONFIG"
		  ["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		  ["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/neutron/neutron-server.log"
		  exec start-stop-daemon --start --chuid neutron --exec /usr/bin/neutron-server -- \
		  --config-file=/etc/neutron/neutron.conf ${DAEMON_ARGS}
		end script

Server node	Software module	Configurations

Compute Node	nova-compute	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		env MAX_STATUS_CHECK_RETRIES=20
		
		pre-start script
		  mkdir -p /var/run/nova
		  chown nova:root /var/run/nova/
		
		  mkdir -p /var/lock/nova
		  chown nova:root /var/lock/nova/

```
  # Only try to modprobe if not running within a container
  if [ ! -f /run/container_type ]; then
    modprobe nbd
  fi

  # If libvirt-bin is installed, always wait for it to start first
  if status libvirt-bin; then
    start wait-for-state WAIT_FOR=libvirt-bin WAIT_STATE=running WAITER=nova-compute
  fi

  # If installed, wait for neutron-ovs-cleanup to complete prior to starting
  # nova-compute.
  if status neutron-ovs-cleanup; then
    # See LP #1471022 for explanation of why we do like this
    retries=$MAX_STATUS_CHECK_RETRIES
    delay=1
    while true; do
      # Already running?
      s=`status neutron-ovs-cleanup`
      echo $s
`echo $s| grep -qE "\sstart/running"` && break
      if retries=`expr $retries - 1`; then
        # Give it a push
        echo "Attempting to start neutron-ovs-cleanup"
        start neutron-ovs-cleanup || :
        # Wait a bit to avoid hammering ovs-cleanup (which itself may be waiting
        # on dependencies)
        echo "Recheck neutron-ovs-cleanup status in ${delay}s"
        sleep $delay
        if _=`expr $retries % 2`; then
           delay=`expr $delay + 2`
        fi
      else
        echo "Max retries ($MAX_STATUS_CHECK_RETRIES) reached - no longer waiting for neutron-ovs-cleanup to
start"
        break
      fi
    done
  fi
end script
```

		exec start-stop-daemon --start --chuid nova --exec /usr/bin/nova-compute -- --config-file=/etc/nova/nova.conf --config-file=/etc/nova/nova-compute.conf
	nova-manage	import sys from nova.cmd.manage import main if __name__ == "__main__": sys.exit(main())

	cinder-volume	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		  for i in lock run lib ; do
		  mkdir -p /var/$i/cinder
		  chown cinder:cinder /var/$i/cinder
		  done
		  for i in log ; do
		  mkdir -p /var/$i/cinder
		  chown cinder:adm /var/$i/cinder
		  done
		end script
		
		script
		  [-x "/usr/bin/cinder-volume"] || exit 0
		  DAEMON_ARGS=""
		  CONFIG_FILE="/etc/cinder/cinder.conf"
		  USE_SYSLOG=""
		  USE_LOGFILE=""
		  NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		  [-r /etc/default/openstack] && . /etc/default/openstack
		  [-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		  ["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"

		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/cinder/cinder-volume.log" [-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE" exec start-stop-daemon --start --chdir /var/lib/cinder \ --chuid cinder:cinder --make-pidfile --pidfile /var/run/cinder/cinder-volume.pid \ --exec /usr/bin/cinder-volume -- ${DAEMON_ARGS} end script

Controller Node	nova-api	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/nova
		chown nova:nova /var/$i/nova
		done
		for i in log ; do
		mkdir -p /var/$i/nova
		chown nova:adm /var/$i/nova
		done
		end script
		
		script
		[-x "/usr/bin/nova-api"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/nova/nova.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/nova/nova-api.log"

		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE" exec start-stop-daemon --start --chdir /var/lib/nova \ --chuid nova:nova --make-pidfile --pidfile /var/run/nova/nova-api.pid \ --exec /usr/bin/nova-api -- ${DAEMON_ARGS} end script

	nova-conductor	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/nova
		chown nova:nova /var/$i/nova
		done
		for i in log ; do
		mkdir -p /var/$i/nova
		chown nova:adm /var/$i/nova
		done
		end script
		
		script
		[-x "/usr/bin/nova-conductor"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/nova/nova.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/nova/nova-conductor.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-

		file=$CONFIG_FILE" exec start-stop-daemon --start --chdir /var/lib/nova \ --chuid nova:nova --make-pidfile --pidfile /var/run/nova/nova-conductor.pid \ --exec /usr/bin/nova-conductor -- ${DAEMON_ARGS} end script

	nova-consoleauth	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/nova
		chown nova:nova /var/$i/nova
		done
		for i in log ; do
		mkdir -p /var/$i/nova
		chown nova:adm /var/$i/nova
		done
		end script
		
		script
		[-x "/usr/bin/nova-consoleauth"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/nova/nova.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/nova/nova-consoleauth.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"

		exec start-stop-daemon --start --chdir /var/lib/nova \ --chuid nova:nova --make-pidfile --pidfile /var/run/nova/nova-consoleauth.pid \ --exec /usr/bin/nova-consoleauth -- ${DAEMON_ARGS} end script

	nova-novncproxy	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/nova
		chown nova:nova /var/$i/nova
		done
		for i in log ; do
		mkdir -p /var/$i/nova
		chown nova:adm /var/$i/nova
		done
		end script
		
		script
		[-x "/usr/bin/nova-novncproxy"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/nova/nova.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/nova/nova-novncproxy.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"

		exec start-stop-daemon --start --chdir /var/lib/nova \ --chuid nova:nova --make-pidfile --pidfile /var/run/nova/nova-novncproxy.pid \ --exec /usr/bin/nova-novncproxy -- ${DAEMON_ARGS} end script

	nova-scheduler	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/nova
		chown nova:nova /var/$i/nova
		done
		for i in log ; do
		mkdir -p /var/$i/nova
		chown nova:adm /var/$i/nova
		done
		end script
		
		script
		[-x "/usr/bin/nova-scheduler"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/nova/nova.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/nova/nova-scheduler.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/nova \

		--chuid nova:nova --make-pidfile --pidfile /var/run/nova/nova-scheduler.pid \ --exec /usr/bin/nova-scheduler -- ${DAEMON_ARGS} end script

	cinder-scheduler	start on runlevel [2345]
		stop on runlevel [!2345]
		
		chdir /var/run
		
		respawn
		respawn limit 20 5
		limit nofile 65535 65535
		
		pre-start script
		for i in lock run lib ; do
		mkdir -p /var/$i/cinder
		chown cinder:cinder /var/$i/cinder
		done
		for i in log ; do
		mkdir -p /var/$i/cinder
		chown cinder:adm /var/$i/cinder
		done
		end script
		
		script
		[-x "/usr/bin/cinder-scheduler"] || exit 0
		DAEMON_ARGS=""
		CONFIG_FILE="/etc/cinder/cinder.conf"
		USE_SYSLOG=""
		USE_LOGFILE=""
		NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
		[-r /etc/default/openstack] && . /etc/default/openstack
		[-r /etc/default/$UPSTART_JOB] && . /etc/default/$UPSTART_JOB
		["x$USE_SYSLOG" = "xyes"] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
		["x$USE_LOGFILE" != "xno"] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/cinder/cinder-scheduler.log"
		[-z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG"] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"
		
		exec start-stop-daemon --start --chdir /var/lib/cinder \
		--chuid cinder:cinder --make-pidfile --pidfile /var/run/cinder/cinder-scheduler.pid \

		--exec /usr/bin/cinder-scheduler -- ${DAEMON_ARGS} end script
	keystone	isten 5000 Listen 35357 <VirtualHost *:5000> WSGIScriptAlias / /usr/bin/keystone-wsgi-public WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone group=keystone display-name=%{GROUP} WSGIProcessGroup keystone-public WSGIApplicationGroup %{GLOBAL} WSGIPassAuthorization On LimitRequestBody 114688 <IfVersion >= 2.4> ErrorLogFormat "%{cu}t %M" </IfVersion> ErrorLog /var/log/apache2/keystone.log CustomLog /var/log/apache2/keystone_access.log combined <Directory /usr/bin> <IfVersion >= 2.4> Require all granted </IfVersion> <IfVersion < 2.4> Order allow,deny Allow from all </IfVersion> </Directory> </VirtualHost> <VirtualHost *:35357> WSGIScriptAlias / /usr/bin/keystone-wsgi-admin WSGIDaemonProcess keystone-admin processes=5 threads=1 user=keystone group=keystone display-name=%{GROUP} WSGIProcessGroup keystone-admin WSGIApplicationGroup %{GLOBAL} WSGIPassAuthorization On LimitRequestBody 114688

		<IfVersion >= 2.4> ErrorLogFormat "%{cu}t %M" </IfVersion> ErrorLog /var/log/apache2/keystone.log CustomLog /var/log/apache2/keystone_access.log combined <Directory /usr/bin> <IfVersion >= 2.4> Require all granted </IfVersion> <IfVersion < 2.4> Order allow,deny Allow from all </IfVersion> </Directory> </VirtualHost> Alias /identity /usr/bin/keystone-wsgi-public <Location /identity> SetHandler wsgi-script Options +ExecCGI WSGIProcessGroup keystone-public WSGIApplicationGroup %{GLOBAL} WSGIPassAuthorization On </Location> Alias /identity_admin /usr/bin/keystone-wsgi-admin <Location /identity_admin> SetHandler wsgi-script Options +ExecCGI WSGIProcessGroup keystone-admin WSGIApplicationGroup %{GLOBAL} WSGIPassAuthorization On </Location>
	glance-api	start on runlevel [2345] stop on runlevel [!2345]

```
chdir /var/run

respawn
respawn limit 20 5
limit nofile 65535 65535

pre-start script
    for i in lock run lib ; do
        mkdir -p /var/$i/glance
        chown glance:glance /var/$i/glance
    done
    for i in log ; do
        mkdir -p /var/$i/glance
        chown glance:adm /var/$i/glance
    done
end script

script
    [ -x "/usr/bin/glance-api" ] || exit 0
    DAEMON_ARGS=""
    CONFIG_FILE="/etc/glance/glance-api.conf"
    USE_SYSLOG=""
    USE_LOGFILE=""
    NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG=""
    [ -r /etc/default/openstack ] && . /etc/default/openstack
    [ -r /etc/default/$UPSTART_JOB ] && . /etc/default/$UPSTART_JOB
    [ "x$USE_SYSLOG" = "xyes" ] && DAEMON_ARGS="$DAEMON_ARGS --use-syslog"
    [ "x$USE_LOGFILE" != "xno" ] && DAEMON_ARGS="$DAEMON_ARGS --log-file=/var/log/glance/glance-api.log"
    [ -z "$NO_OPENSTACK_CONFIG_FILE_DAEMON_ARG" ] && DAEMON_ARGS="$DAEMON_ARGS --config-file=$CONFIG_FILE"

    exec start-stop-daemon --start --chdir /var/lib/glance \
        --chuid glance:glance --make-pidfile --pidfile /var/run/glance/glance-api.pid \
        --exec /usr/bin/glance-api -- ${DAEMON_ARGS}
end script
```