Small cEllS coordinAtion for Multi-tenancy and Edge services

Grant Agreement No.671596

Topic: H2020-2014-ICT-14
Advanced 5G Network Infrastructure for the Future Internet
Research and Innovation Action

Deliverable D5.2

VIM and CESCM Implementation

Document Number:	H2020-5GPPP-GA No.671596/*WP5/D5.2/31.09.2017*
Contractual Date of Delivery:	*31.09.2017*
Editor:	ATOS
Work-package:	*WP5*
Distribution / Type:	Public (PU) / Report (R)
Version:	1.0
Total Number of Pages:	56
File:	*SESAME_Deliverable 5.2_v1.0_Final*

Abstract

This deliverable summarizes the work carried out in WP5 "*Infrastructure Virtualisation and Management*" and, *more specifically*, compiles the work performed in the Task 5.2 ("*VIM and CESCM implementation*"). This task is an important component in the SESAME infrastructure, as it acts as the platform intelligence of the system. CESC Manager (CESCM) is the component with an overall knowledge of the virtual and physical resources, responsible for the proper deployment, monitoring, configuration and orchestration of the Light DC cloud environment as well as of the various radio access functionalities, over a single/multiple CESC cluster(s).

The result of this document, the developed software is being integrated in the common platform developed in SESAME project.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	24.04.2017	Preliminary TOC	Atos
0.1.1	04.05.2017	Description of sections, and expected contribution	Atos
0.2	16.06.2017	Contribution to sections	Atos
0.3	18.08.2017	Updated version	Atos
0.4	29.08.2017	Contribution to section 3	VOSYS
0.5	30.08.2017	Contribution to section 3 and 4	I2cat, IPA
0.6	04.09.2017	Contribution to section 3	NCSRD
0.7	12.09.2017	Minor corrections to Section 4.1	NCSRD
0.9	19.09.2017	Contribution to section 3.2and 4.5	OTE, ZHAW
0.91	25.09.2017	Pre-final version	Atos
0.92	27.09.2017	Full review of the deliverable	SMNET
1.0	30.09.2017	Final version submitted by coordinator after full editorial and conceptual review	OTE

Contributors

First Name	Last Name	Partner	Email
Elisa	Jimeno	ATOS	*Elisa.jimeno@atos.net*
Javier	Garcia	ATOS	*Javier.garcial@atos.net*
Nikolay	Nikolaev	VOSYS	*n.nikolaev@virtualopensystems.com*
Pouria	Sayyad Khodashenas	i2cat	*pouria.khodashenas@i2cat.net*
Alan	Whitehead	IPA	alan.whitehead@ipaccess.com
Akis	Kourtis	NCSRD	akis.kourtis@iit.demokritos.gr
Ioannis	Giannoulakis	NCSRD	giannoul@iit.demokritos.gr
Emmanouil	Kafetzakis	ORION	*mkafetz@orioninnovations.gr*
Miren	Cava	EHU	*Miren.cava@gmail.com*
Bego	Blanco	EHU	*begona.blanco@ehu.eus*
Irena	Trajkovska	ZHAW	*traj@zhaw.ch*
Athanassios	Dardamanis	SMNET	*adardamanis@smart.net.gr*
Maria	Belesioti	OTE	*mbelesioti@oteresearch.gr*
Ioannis	Chochliouros	OTE	*ichochliouros@oteresearch.gr*

Glossary

Acronym	Explanation
5G	Fifth Generation of Mobile Communications
ANR	Automatic Neighbor Relation
AP	Application Protocol
API	Application Programming Interface
ARM	Advanced RISC Machine
BF	Broadband Forum
BSS	Business Support System
CAPEX	Capital Expenditure
CCITT	International Telegraph and Telephone Consultative Committee
CESC	Cloud-enabled Small Cell
CESCM	Cloud Enabled Small Cell Manager
CPU	Central Processing Unit
C-RAN	Cloud-enabled RAN
DB	Database
DC	Data Centre
DL	Downlink
E2E	End-to-End
EMS	Element Management System
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
EU	European Union
FCAPS	Fault, Configuration, Accounting, Performance and Security
FM	Fault Management
FP	Framework Programme
GA	Grant Agreement
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
GW	Gateway
H2020	Horizon 2020
HD	High Definition
HDD	Hard Disk Drive
HeNB	Home eNodeB
HTTP	Hyper-text Transmission Protocol
HW	Hardware
ICT	Information and Communication Technology
ID, id	Identifier
IP	Internet Protocol
IT	Information Technology
IVM	Infrastructure Virtualisation and Management
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
Light DC	Light Data Centre
LAN	Local Area Networks
MANO	Management and Orchestration
MEC	Mobile Edge Computing
MIB	Management Information Base
MME	Mobility Management Entity
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure

NFVO	NFV Orchestrator
NMS	Network Management System
NOS	Network Orchestration System
NS	Network Service
NSD	NS Descriptor
NSNM	Network Service Notification Manager
NUMA	Non-Uniform Memory Access
ODL	OpenDayLight
OPEX	Operational Expenditure
OPNFV	Open Platform for NFV
OS	Operating System
OSS	Operations Support System
OVS	Open virtual Switch
PLMN	Public Land Mobile Network
PM	Performance Management
PNF	Physical Network Function
PPP	Public-Private Partnership
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RDB	Reference Design Board
RISC	Reduced Instruction Set Computer
RRC	Radio resource Control
SC	Small Cell
SCaaS	Small Cells-as-a-Service
SCNO	Small Cell Network Operator
SCP	Session Control protocol
SDK	Software Development Kit
SDN	Software-defined Networking
SFC	Service Function Chaining
SFTP	SSH File Transfer Protocol
SLA	Service Level Agreement
SLO	Service Level Objective
SMTP	Simple Mail Transfer Protocol
SO	Service Orchestrator
SoC	System on Chip
SON	Self-Organised Network
SOTA	State-of-the-Art
SSD	Solid State Drive
SSH	Secure Shell
SW	Software
TR	Technical Report
UI	User Interface
UL	Uplink
UTRA	Universal Terrestrial Radio Access
UTRAN	Universal Terrestrial Radio Access Network
VIM	Virtual Infrastructure Manager
VLAN	Virtual Local Area Networks
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFM	VNF Manager
VSCNO	Virtual Small Cell Network Operator
vTU	virtual Transcoding Unit

WG	Working Group
WP	Work Package
WS	Wed Service
XML	Extensive Markup Language

Table of Contents

List of Figures

List of Tables

1. Introduction

This deliverable is the final report of the work carried out in WP5 ("*Infrastructure Virtualisation and Management*") and, more specifically, with the work performed in the Task 5.2 ("*Implementation of VIM and CESCM*"). The task focuses on the implementation and integration of a monitoring framework; this is able to extract, process and communicate monitoring information from both physical and virtual nodes as well as VNFs at IVM level.

The task poses the challenge of offering Small Cells-as-a-Service (SCaaS) by creating distributed clusters of Light DC. These groups are not static but require a dynamic management because these virtual nodes can be switched ON/OFF, depending -*for instance*- on the subscriber's action or availability.

The management of CESCs implies the study and incorporate statistical models at the management layer in order to consider the dynamics of the scenario. This is a great difference as compared to traditional cloud resources in a data centre, where the hardware (HW) is perfectly controlled and located in homogeneous server farms. This leads to the consolidation of the CESCM entity. This entity -incorporating the orchestrator- manages the use, performance and delivery of services and the relationship between the CESC provider and the network operator. The CESCM must cope with the functionalities of the Virtual Infrastructure Manager (VIM).

The operation of CESC resources and their provisioning as autonomous virtual networks as slices are based on state-of-the-art (SOTA) network virtualisation frameworks (i.e. OpenStack[1] or the recently announced OPNFV[2] or some functionalities claimed by OpenDayLight[3]). There are several VIM/CESCM implementation possibilities: (i) a centralised approach in which the CESCM is a new network element that manages the cluster, it allows the operator a greater control of the network functions that run over the cluster, yet scalability issues could arise, *and*; (ii) a distributed approach in which the CESCM features are carried out by the CESCs themselves as VNFs.

1.1. Deliverable outline

The presented document is structure as follow:

- Section 1 introduces the project and the scope of the document.
- Section 2 overviews the introduction of the MANO architecture, and how it has been integrated in the SESAME platform.
- Section 3 describes VIM concept, and implementation of the modification made within SESAME context.
- Section 4 presents the CESC Manager and the different components that are included in the internal architecture.
- Finally, Section 5 concludes the deliverable.

[1] For more details see: *https://www.openstack.org/*
[2] For more details see: *https://www.opnfv.org/*
[3] For more details see: *https://www.opendaylight.org/*

2. MANO

NFV MANO [1] (Network Functions Virtualization Management and Orchestration), also called as MANO, is the framework lead by the Working Group (WG) of the ETSI (European Telecommunications Standards Institute) that defines the architecture for managing and orchestrating virtualized network functions (VNFs), to facilitate the deployment and connection of the virtualized resources.

The resources in the network infrastructure composed of computing, networking, storage and virtual machines (VMs) need to be managed to create and associate as well as to allow a flexible usage and optimisation of the network components. MANO has addressed this essential need by defining the layers and APIs (application program interfaces) in the existing systems.

This architectural framework consists of three main functional blocks, that is: VIM (Virtual Infrastructure Manager), NFVO (Network Function Virtualization Orchestrator) and VNFM (Virtual Network Function Manager).

- NFV Orchestrator - It is responsible for: on-boarding of new network services (NSs) and virtual network function (VNF) packages; NS lifecycle management; global resource management; validation and authorization of network functions virtualization infrastructure (NFVI) resource requests.
- VNF Manager - This oversees lifecycle management of VNF instances; it is also responsible for the coordination and adaptation roles for configuration and event reporting, between NFVI and EMS/NMS.
- VIM - This controls and manages the infrastructure, compute, storage, and network resources.

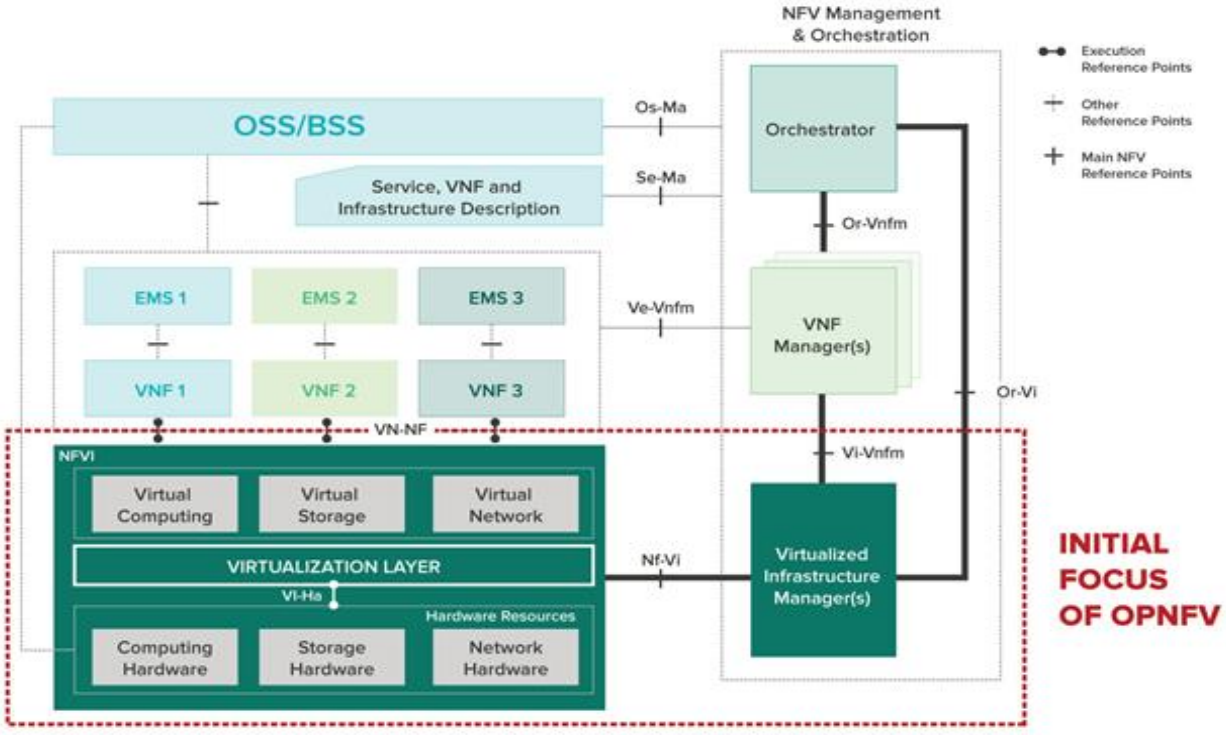

Figure 1: MANO architecture

The SESAME project has adopted the architecture defined by ETSI to facilitate the deployment and connection of the services developed, of the decoupled resources from physical devices and virtualised resources. The following picture depicts the architecture composed of the functional blocks part of the manager infrastructure component. The tree main components have been defined as: Cloud Enable Small Cell Manager (CESCM); Virtual Infrastructure Manager (VIM),

and; Network Function Virtualisation Orchestrator (NFVO), commonly known simply as orchestrator. The architecture also defines the communication and links between them.

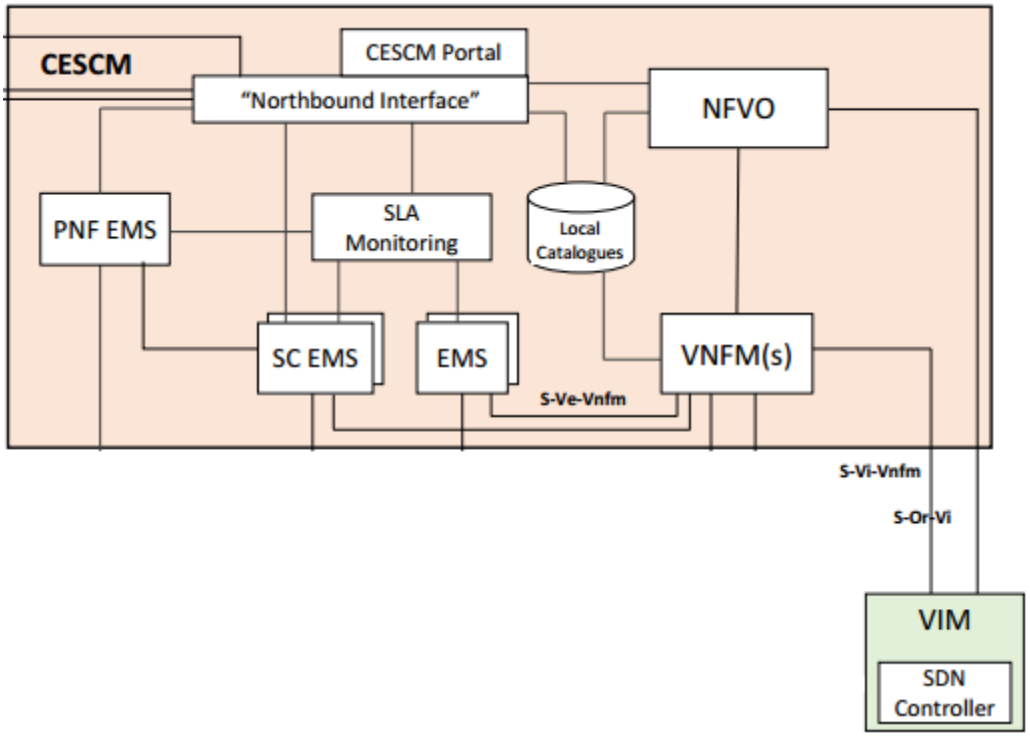

Figure 2: Sesame CESCM Architecture

- The CESCM Portal is the graphical user interface (GUI) used by SCNO and VSCNOs to interact with the platform capabilities. Such interaction is enabled by the Northbound Interface[4] which supports the data exchange and tenant configuration parameters with the Orchestrator.
- The NFV Orchestrator is the entity in charge of receiving network service requests from the VSCNOs and mapping them to specific VNFs instances and service chaining configurations in the Light DC. It receives the network service requests from the northbound interface. Every SESAME NS request is defined in a Network Service Descriptor (NSD), which contains references to the VNFD that describes the network service. The NFVO translates the references into requests able to be understood by the VIM. It also undertakes coordination with the VNFM for the creation, termination, monitoring and scaling of end-to-end (E2E) service chain.
- The VNF Manager (VNFM) is the entity in charge of the lifecycle management of the VNFs, from deployment to termination, keeping track of their status to adjust their configuration, if needed.
- The EMS is the entity in charge of the key functionalities as fault, configuration, accounting, performance and security (FCAPS). It manages the traffic between the different network elements, coordinating configuration of multiple devices. The EMS associated to radio functions also includes autonomous "self-x" functionalities to reconfigure the mobile network.

[4] For further information see, *inter-alia*: *https://en.wikipedia.org/wiki/Northbound_interface*

- The SLA component enhances service reliability providing monitoring mechanisms to evaluate the performance of network services in the radio and cloud environments. It communicates with the NFVO, notifying faults in the system for it to perform the appropriate actions that assure the QoS guarantees of each service in a multi-tenant environment.

The Virtual Infrastructure Manager (VIM), as described in ETSI, is responsible for managing the virtualized infrastructure that includes the catalogue of the allocated resources, forwarding graphs and chaining rules among VNFs and repository of resources, to provide optimized features. In the SESAME platform, VIM is the software (SW) entity that monitors and manages the NFVI (i.e., Light DC) and performs the lifecycle management of the virtual units that will host the VNFs. This centralized administration of virtual resources across multiple localized infrastructures so that instances can be administrated in a coordinated way, provides the flexibility and scalability needed to optimize and maximize the use of such resources. The VIM is enhanced with SDN component for the networking aspect. The controller takes into account the physically distributed SESAME NFVI and the stringent requirements in RAN performance metrics; SESAME uses SDN for propagating the VNF chaining requests to the NFVI in order to properly manage the networking resources within the Light DC. The description and features of the different components will be described in more detail in the following sections.

3. Virtual Infrastructure Manager

The Virtual Infrastructure Manager is the software responsible for controlling and it manages the NFV infrastructure resources, compute, storage and network. It ensures that physical and virtual resources "work smoothly" and that the challenge faced is the synchronization of allocation based on requirements while using resources from different location.

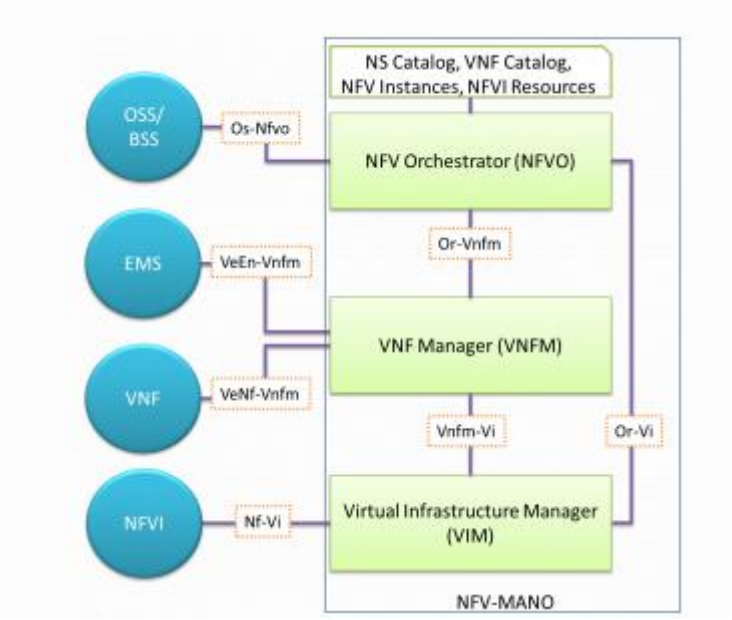

Figure 3: VIM

There are many benefits that the VIM provides; the main feature is scalability that allows creating a tailored infrastructure, provisioned with the needed resources. CAPEX and OPEX are reduced through server and network configuration, thus making the infrastructure "more flexible". It also makes more secure infrastructure as all traffic is filtered through the physical infrastructure. Moreover, load balancing computer networking distributes the workload in the infrastructure to handle inbound traffic in order to enhance speed. Finally, backup copy can be saved for a faster recovery situation.

To make sure that cloud operations run well both at the frontend and at the backend, a virtual infrastructure manager needs to fulfill some basic requirements/features. Some of these are presented below:

- Supporting the "On Demand" concept of the cloud. The virtual infrastructure manager has most importantly to ensure that the cornerstone of cloud computing -"on demand" and also "self-servicing"- is provided.
- Hardware Virtualization: Virtualization is one of the most basic and important requirements, so that to serve multiple consumers. Hardware resources -such as CPU power and memory- will be virtualized and have to be used so that to be conformant to specific requirements.
- Storage virtualization: The VIM has to "abstract" logical storage from physical storage, by creating virtual storage disks thus ensuring multiple storage requirements.
- Resource allocation on the fly. Dynamic resource allocation features in Data Centers have a "key role" in energy management, as the consumption of energy is growing; reducing the number of machines, by consolidating them, will provide significant cost benefits.
- Minimization of downtime, high Availability feature which ensures that there is a minimum downtime by using failover mechanism that restarts health servers when there is a failure detected.

- Data backups: The dependency of data backups on cloud by proactively backing up the datacenters.

The VIM component has received enormous focus to manage the virtual infrastructure of MANO; in particular, many open source initiatives have been involved in the description and the definition.

3.1. VIM adaptation for ARM platform

The VIM chosen for the SESAME project's implementation is the popular open source project OpenStack. During the course of the project, and especially during the architecture design and prototyping of the Light DC controller, the relevant components of OpenStack were adapted to be integrated within the software stack environment available on the platform.

The default firmware build to run on top of the Light DC prototyping platform (NXP LS2085A-RDB[5]) is defined in a Yocto-*based* environment. Yocto[6] is a popular flavour of another Linux distribution build framework called OpenEmbedded[7]. It is specified in creating read-only images where the software packages are installed at the time of the creation of the firmware and cannot be easily upgraded or modified at runtime.

Moreover, the NXP LS2085A SoC is a complex system, comprised of eight ARMv8[8] CPU cores and a highly sophisticated network acceleration engine called DPPA2. The integration process need to take into account those details and modify, adapt and create new components to manage the SoC hardware. These specifics of the selected Light DC platform, imposes that there are 2 levels of integration changes: (a) Nova[9] (virtual compute management) and Neutron[10] (virtual network management) changes; (b) integration with the Yocto build firmware environment for the LS2085A-RDB.

The current OpenStack version which was available as a part of the NXP Yocto SDK at the time of active development within the SESAME project, had problems properly identifying the NUMA architecture of the ARMv8 SoC. This information is used by the Nova agent in order to identify the ARM compute node as a valid host, which has the needed resources to bring up the VM. This same information is used to prepare the libvirt[11] VM description XML file. For the purposes of completing this integration task VOSYS did patch the Nova compute agent running on the Light DC prototype platform, so that the proper NUMA architecture description is generated.

Neutron is the component that manages the virtual network within OpenStack. VOSYS integrated its network dataplane called VOSYSwitch, with the advanced networking engine DPAA2[12] on the LS2085A. This was done by leveraging the OpenDataPlane compatible drivers from NXP. For the purposes of this integration VOSYS implemented a specific Neutron agent that is driving VOSYSwitch on the ARM compute node. It supports two major network isolation paradigms: VLAN and GRE overlays[13].

---

5 For more details see: *https://www.nxp.com/part/LS2085ARDB-PA*

6 For more details see, *inter-alia*: *https://wiki.yoctoproject.org/wiki/Main_Page*

7 More details can also be found at: *https://www.openembedded.org/wiki/Main_Page*

8 ARM is the industry's leading supplier of microprocessor technology, offering the widest range of microprocessor cores to address the performance, power and cost requirements for almost all application markets. Combining a vibrant ecosystem with over 1,000 partners delivering silicon, development tools and software, and with more than 90bn processors shipped, our technology is at the heart of a computing and connectivity revolution that is transforming the way people live and businesses operate. For more details also see: *https://www.arm.com/products/processors*.
The ARMv8 architecture introduces 64-bit support to the ARM architecture with a focus on power-efficient implementation while maintaining compatibility with existing 32-bit software. More related information can be found at: *http://www.arm.com/products/processors/armv8-architecture.php*

9 See: *https://wiki.openstack.org/wiki/Nova*
Also see the context at: *https://www.pluralsight.com/courses/openstack-nova-neutron*

10 See: *https://wiki.openstack.org/wiki/Neutron*

11 For more details also see: *https://libvirt.org/*

12 Purely for informative purpose also see the context provided at: *http://www.businesswire.com/news/home/20150622005367/en/Freescale-networking-personal-new-ARM-based-QorIQ-multicore*

13 For further information see: *https://ask.openstack.org/en/question/51388/whats-the-difference-between-flat-gre-and-vlan-neutron-network-types/*

The overall architecture of the VIM running on the LS2085A-RDB Light DC prototype is shown on the following figure:

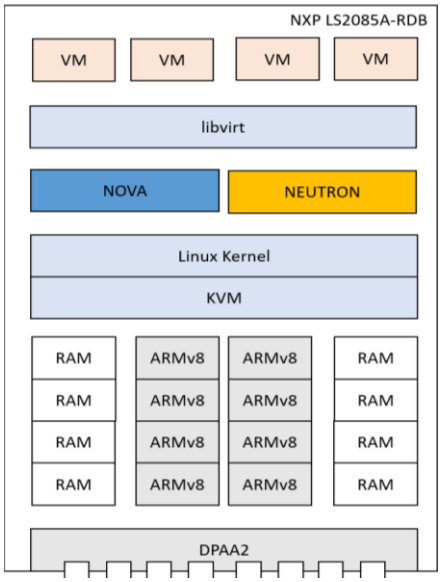

Figure 4: VIM ARM platform

VOSYS has published a guide on their website, where the whole Yocto build and installation process is thoroughly described. The process of building and creating a firmware image for that platform is illustrated on the following figure:

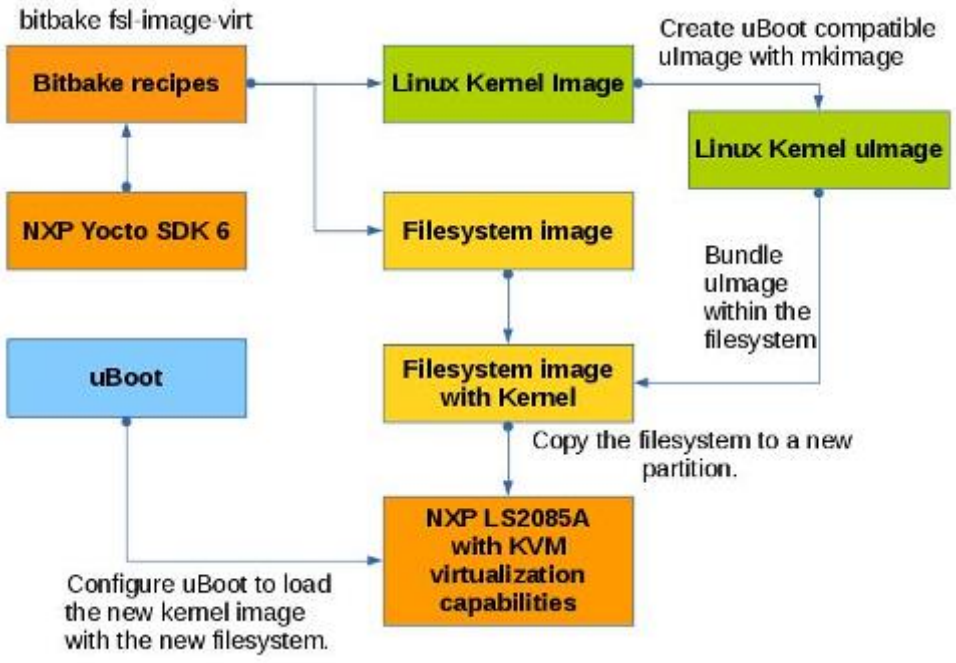

Figure 5: Building process

More details regarding the architecture and implementation of the VIM in the Light DC solution are available in the following resources:

- SESAME deliverable D4.1 ("*Light DC architecture design*"); chapter *4.4.4 ("OpenStack integration"*).

- SESAME deliverable D4.4 ("*Light DC prototype*"); chapter 4.3 ("*OpenStack Agents for ARMv8"*).

- VOSYS website guide "*How to setup NXP LS2085A-RDB as KVM virtualization host with VOSYSwitch vSwitch support*"; also reported in [2].

3.2. VIM configuration

OTE, together with ITL and VOSYS, incorporated the ARM adaptation to the OpenStack platform. Firstly OpenStack networking (Neutron) was configured so as to use the OpenVswitch (OVS) network driver. Then VOSYS incorporated the VOSYS switch and the Virt-manager (KVM) to use huge pages, while ITL was first proof testing this in their testbed with just KVM without OpenStack.

Both OpenStack Nova and Glance[14] were configured so as to be able to "see" NXP boards as Compute nodes and to support ARM images, *respectively*. In order for NXP boards to work as Compute nodes, these have been configured as well.

Initially used NXP with OS yoctoX1 and then VOSYS provided an update to yoctoX2, which OTE also incorporated to the NXP board provided by NCSRD, which worked better and with more stability. ITL also provided the vTU VNF for both x86[15] and ARM to test in the OTE OpenStack testbed and, in the meantime, OTE kept patching OpenStack as it was breaking while we incorporated the various components and changes. Finally, a lot of testing between the various stages, until Glance and Nova was able to properly manage the NXP boards as Compute nodes took place.

In the following picture, OTE's testbed topology is depicted.

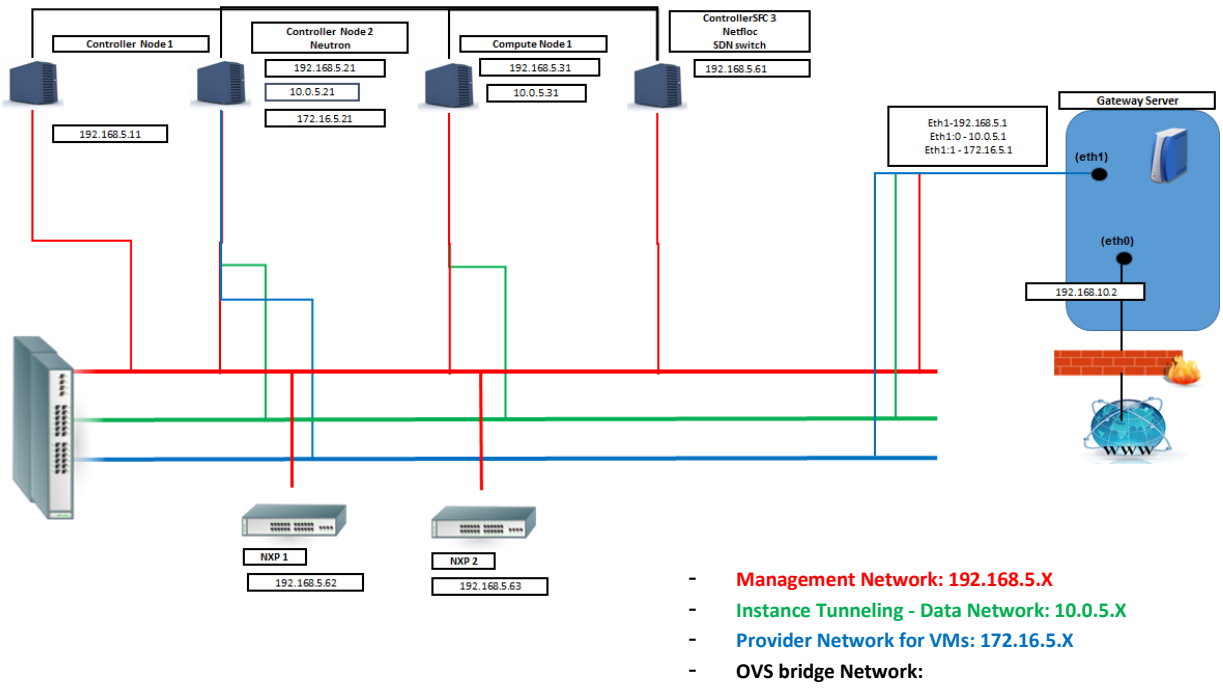

Figure 6:OTE´s testbed topology

The existing hardware of OTE's can be found in the table below.

[14] See: *https://wiki.openstack.org/wiki/Glance*

[15] x86 is a family of backward compatible instruction set architectures based on the Intel 8086 CPU and its Intel 8088 variant. More relevant information can be found, *inter-alia*, at: *https://en.wikipedia.org/wiki/X86.*

Table 1: OTE´s testbed hardware

	System	CPUs	HD	RAM (GB)	eth onboard	eth installed	OS
Gateway	T310 PowerEdge [16]	4	500GB 2,5" + 2TB 3,5''	12	2	1	Ubuntu Server 14.04 LTS 64-bit[17]
Controller 1	Optiplex 9020[18]	4	2x 500 GB	8	1	1	Ubuntu Server 14.04 LTS 64-bit
Controller 2 (Neutron)	T310 PowerEdge	4	1TB	8	2	1	Ubuntu Server 14.04 LTS 64-bit
SFC	Optiplex 9020	4	2TB	8	1	1	Ubuntu Server 14.04 LTS 64-bit
Compute 1	Optiplex 9020	4	2TB	24	1	1	Ubuntu Server 14.04 LTS 64-bit
Compute 2	NXP	4	250GB	4	1+4	0	yocto 64-bit
Compute 3	NXP	4	250GB	4	1+4	0	yocto 64-bit

[16] Also see: http://www.dell.com/en-us/work/shop/dell-poweredge-servers/poweredge-t310-tower-server/spd/poweredge-t310
[17] Also see: http://releases.ubuntu.com/14.04/
[18] Also see: http://www.dell.com/en-us/work/shop/cty/pdp/spd/optiplex-9020-desktop

3.3. VIM support for containers

The SESAME Light DC environment is formed by little IT resources available per micro-server. Depending on the number of CESCs mounted on an area (e.g. a concert hall), the Light DC capacity changes a lot. With the emergence of new applications (e.g. immersive videos and augmented reality gaming), situations where the limited resources of the Light DC should server more users and/or heavy computing services are much expected. This will "put" a spot light on employing lightweight virtualization technologies to improve IT resource utilization.

Containers have emerged as an answer to this demand [3]. In simple words, containers are a way to package software (in our case VNF) in a format that can "run" isolated on a shared operating system. Unlike virtual machines, containers do not bundle a full operating system – only required dependencies - binaries, libraries, configuration files, scripts, virtualenvs, jars, gems, tarballs, etc. This feature makes them efficient, lightweight, self-contained systems which run always the same, regardless of where it is deployed. As an example of container technology we can name Dockers[19] which are able to run on any x64 Linux kernel supporting cgroups and aufs. Anyhow, it is worth to note that containers are not meant to replace VMs; they are "complementary" in the sense that they are better for specific use cases.

Containers have their own lifecycle management systems, so called as container orchestrator. In ETSI MANO terminologies the container orchestrator can be seen in the VIM level, where interaction with the virtualized infrastructure happens. Kubernetes [4] is an open source example of container orchestrators. It helps automating deployment, scaling, and management of containerized applications.

OpenStack community, as the mostly deployed open source VIM, followed the container developments for a while and finally decided to add support in OpenStack for container technology.

A cross-functional team has been formed in 2014 [5] for this purpose with a clear vision for advancing container technology in existing OpenStack services, and new services and tools that evolve OpenStack to track with new technological advances in this area. The final target is to build up a solution where the users of OpenStack can create and manage containers on OpenStack with an experience consistent with what they expect from using the Nova service to get virtual machines.

[19] For further information see, *for example*: *https://wiki.openstack.org/wiki/Docker*

OpenStack has lunched two main projects to introduce Docker supports:

1 - Magnum [6]: It is an OpenStack API service to make container orchestration engines such as Docker Swarm[20], Kubernetes and Apache Mesos[21] available for OpenStack. To properly use Magnum[22], an OS image which contains Docker and Kubernetes (container orchestration engine) should be previously provided. It means that VNF developer who wants to lunch his/her application with Docker should also include a container orchestration engine along with the actual VNF software on the VNF/OS image. Magnum helps to run that image in either virtual machines or bare metal in a cluster configuration, i.e. the lifecycle management of a set of Dockers inside a VM is managed jointly: cluster creation, update, delete. Figure 7 presents the magnum architecture:

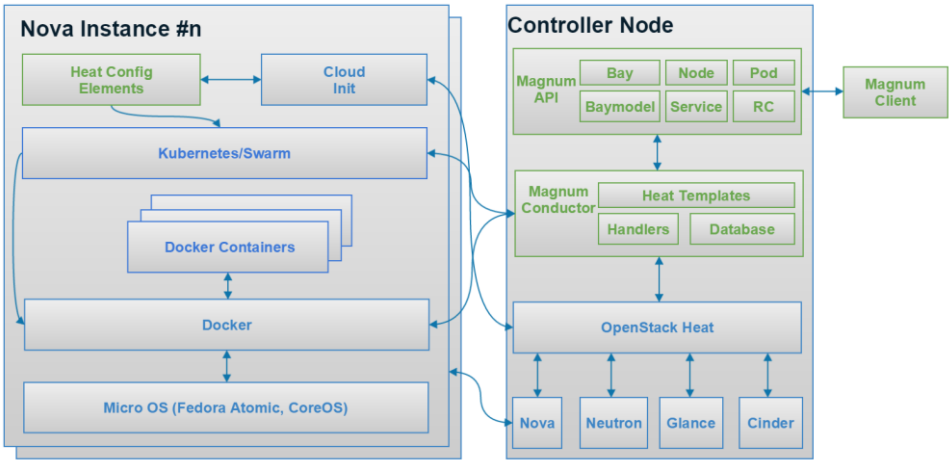

Figure 7: Magnum architecture

As shown in Figure 7, Magnum uses Nova instances (or VMs) to compose the Docker clusters. Moreover, it provides a purposed built API to manage application containers orchestration engines, which have a different lifecycle and operations than Nova (virtual machine) instances. In this sense, the IT resources dedicated to one VM might be used for several applications deployed in the form of Dockers inside the VM. However, as mentioned above, lifecycle management (i.e. creation, update and delete) is at the cluster level. This rough granularity of lifecycle control might not be sufficient for all use cases. Besides, another important challenge with Magnum is to support multi-tenancy. To do so, with Magnum, resources such as clusters can only be viewed and accessed by users of the tenant that created them. In another words, clusters are not shared, meaning that containers will not run on the same kernel as neighboring tenants. Of course, such isolation improves the security in a sense that containers belonging to the same tenant will be tightly packed within the same Pods and Clusters, but runs separate kernels (in separate Nova instances/VMs) between different tenants. However, it might not be necessarily the most IT resource efficient approach, since at least one VM is needed per tenant bases.

2 - Docker [7]: The project targets to develop the Docker driver, a hypervisor driver, for OpenStack Nova Compute. According to the documentations, it is expected that the

[20] Also see: https://docs.docker.com/engine/swarm/
[21] Also see: http://mesos.apache.org/
[22] Also see: https://wiki.openstack.org/wiki/Magnum

Docker driver be available on the Kilo release[23]. The project takes the advantage of containers and filesystem technologies in a high-level way which are not generic enough to be managed by libvirt [8]. That is the Doker virt driver is out of libvirt. The project provides items such as: (i) Process-level API: to collect the standard outputs and inputs of the process running in each container for logging or direct interaction; it allows blocking on a container until it exits, setting its environment, and other process-*oriented* primitives which do not "fit well" in libvirt's abstraction; (ii) Advanced change control at the filesystem level: every change made on the filesystem is managed through a set of layers which can be snapshotted, rolled back, diff-ed, etc.; (iii) Image portability: the state of any Docker container can be optionally committed as an image and shared through a central image registry. Docker images are designed to be portable across infrastructures, so they are a great building block for hybrid cloud scenarios; (iv) Build facility: Docker can automate the assembly of a container from an application's source code.

Figure 8 shows how the Nova hypervisor works. The Nova driver embeds a tiny HTTP client is able to communicate with the Docker internal Rest API through a Unix socket. It uses the HTTP API to control containers and fetch relevant information. The driver will fetch images from the OpenStack Image Service (Glance) and load them into the Docker filesystem. Images may be placed in Glance by exporting them from Docker using the 'docker save' command. The proposed solution by the Docker project provides a finer granularity of control compared to Magnum. However, the work is on progress and needs to be constantly followed up.

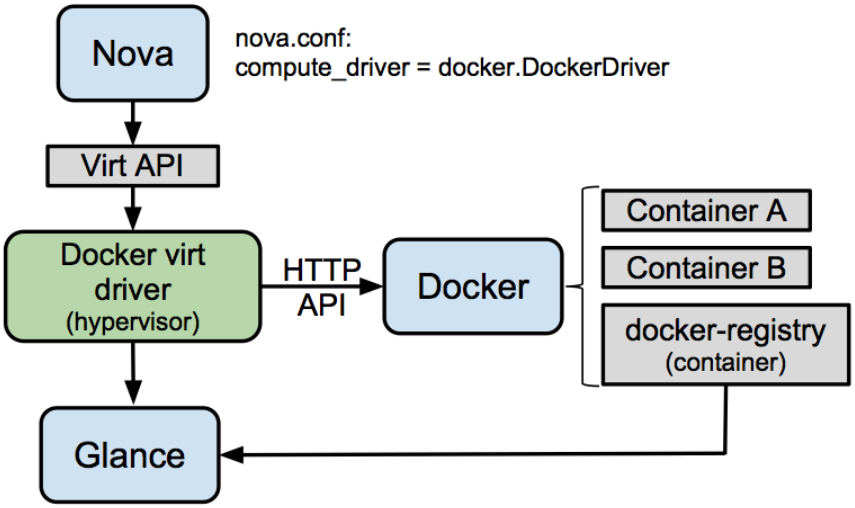

Figure 8: Docker virt driver

[23] See: *https://wiki.openstack.org/wiki/Kilo_Release_Schedule*

3.4. VNF Placement Algorithm extension

One of the "key challenges" that management process has to cope with, is the placement of each component which forms a NS. The placement process refers to the mechanism that allocates the softwarized components of a NS onto the networked datacentres that conform the CESC Cluster. In other words, it is about "where to instantiate the VNF chain that composes a NS". Figure 9 shows an example of the placement process of a NS in the virtualized resources of a SESAME Light DC. At a logical level, the NFVO configures the ordered sequence of VNFs upon a new NS request. Then, NFVO maps the VNF chain to specific VNF instances and service chaining configurations to the Light DC. Finally, the VIM, which manages the virtual units that will host the VNFs, propagates mapping requests to the underlying physical resources that compose NFVI.

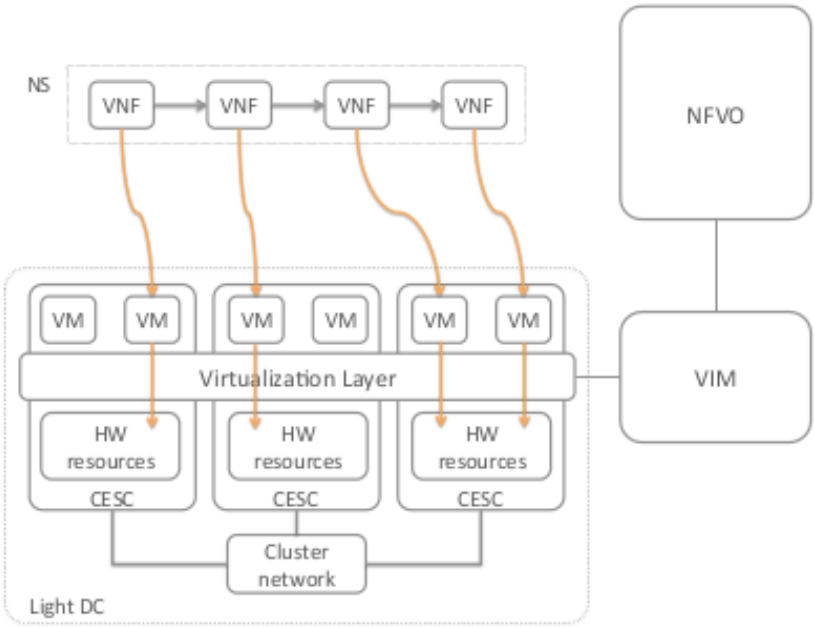

Figure 9: Placement process

Figure 10 shows the scheme of the SESAME placement algorithm. As stated previously, NSD contains the logical VNF chain that composes a requested NS plus the service constraints imposed by the KPIs of the corresponding SLA. In our work, the SLA KPIs include the following network parameters: accepted latency for the NS, aggregated user bit rate and a robust protection parameter. With this input, the placement algorithm checks the available virtualized resources in the NFVI catalogue and the instantiation requirements for each VNF in the VNF catalogue. Finally, the Infrastructure Manager functional element -*Virtualized Infrastructure Manager (VIM) in ETSI-MANO terminology*- receives the outcome of the placement decision and maps each VNF to the assigned resources. The success of the approach depends on the efficiency of the VNF placement algorithm to allocate the virtual functions in the available resources given a set of service constraints.

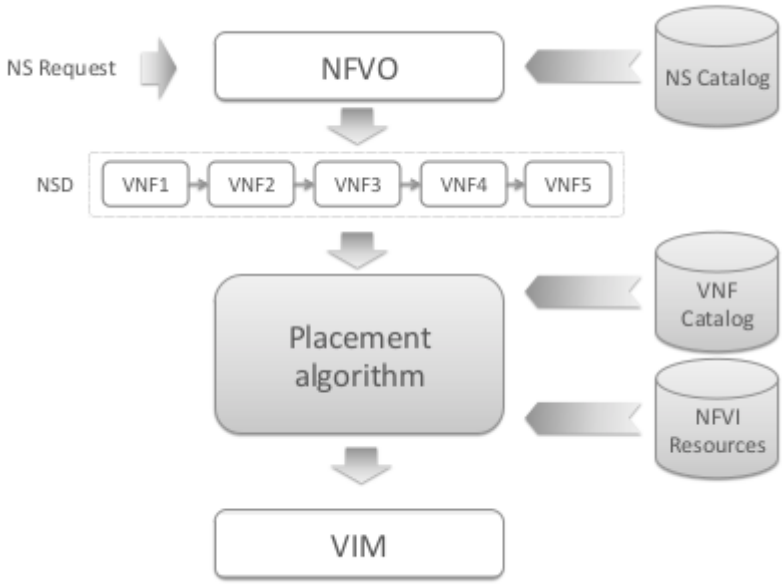

Figure 10: Scheme placement algorithm

In this context, our work is focused upon the design of a placement algorithm and this work is discussed in the deliverable D5.3, where we explain the SESAME placement problem modelling as a VNF placement optimization problem that minimizes power consumption subject to service delay constraints and the results of the placement algorithm.

4. CESC Manager

Cloud Enable Small Cell Manager (CESCM) is the main management component in the architecture, covering the orchestration, management and configuration of NSs. The CESCM has a high-level knowledge of the virtual and physical resources available on the CRAN environment, including the radio access functionalities.

The challenge SESAME aims to "overcome" (i.e. providing services that involve both radio and virtualization aspects at the network's edge), implies more complex management functionalities for such services. For that purpose, the platform is designed in such a way that the radio access management task (e.g. transmission power control, packet scheduling, handover and cell reselection thresholds, etc.) and NFV management responsibilities (e.g. VNF/service instantiation, lifecycle management, policy management etc.) can be handled in an orchestrated and more centralized way.

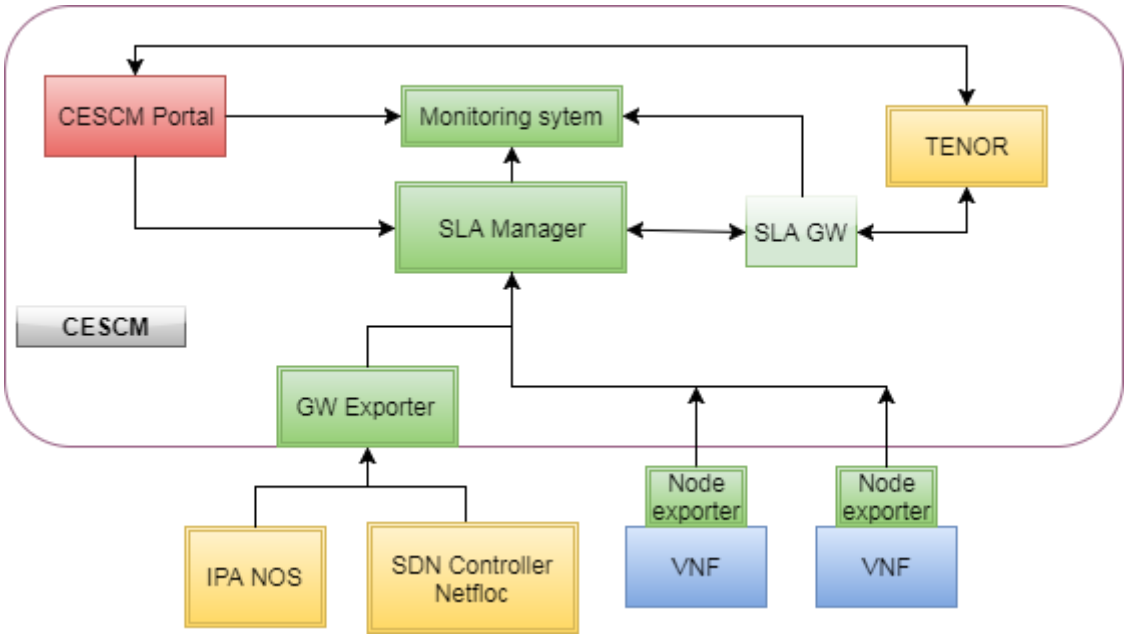

Figure 11: SLA architecture

The CESCM is composed of several pieces; together it manages the use, performance and delivery of services and the relationship between the CESC provider and the network operator. CESCM entity is the "key component" for the creation of complex virtual network scenarios, based on end-to-end virtualization management, evaluating the service assurance and SLA management on multi-tenancy scenarios.

4.1. CESCM portal

The CESCM portal is the frontend manager of the SESAME services and underlying infrastructure. SESAME, being a project oriented to the Mobile Edge Technology, combines a variety of technologies, such as NFV and network management. The portal is based upon the functionality of the CESC core components, namely the NFV orchestrator, the monitoring and the EMS of physical resources IPA's NOS.

An overview of the portal's Catalog Page is provided below:

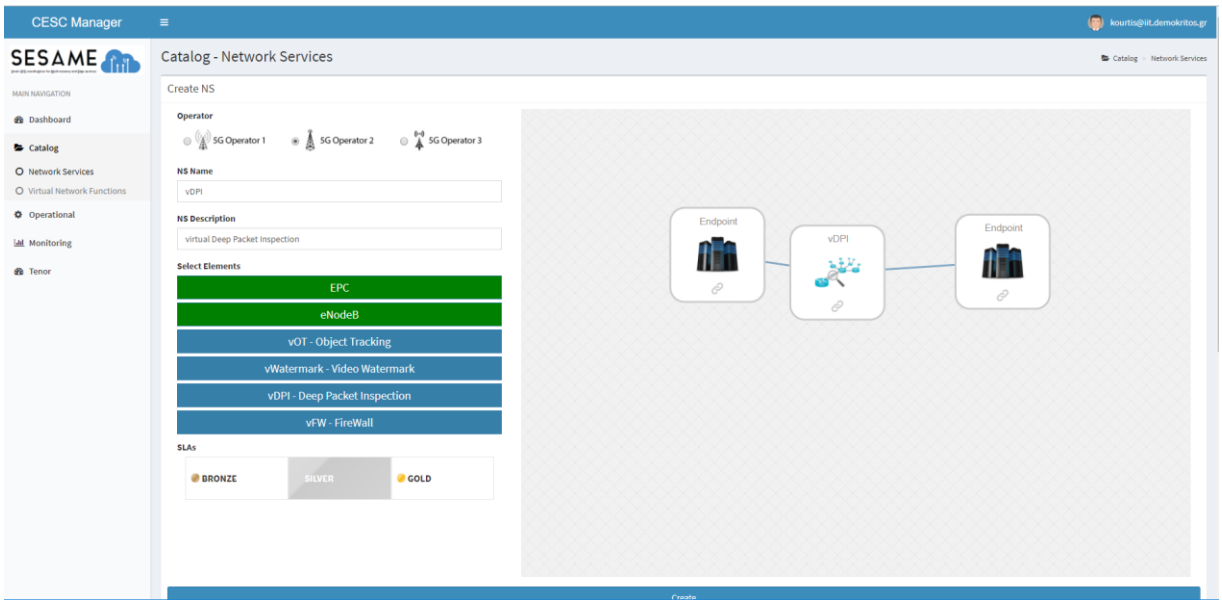

Figure 12: Portal Catalog

As it can be seen, the support for multi-operator functionality has been implemented. The CESC admin, before composing a network service, can choose among a group of operators, who want to deploy their network service in the SESAME testbed.

The admin user can choose from a visualized list a predefined VNF, a set of physical resources, in our example an EPC and an eNodeB[24], and preview the topology of the composed service. Additionally, SLA support has been added in order to choose among a 3-tier service, which is enforced afterwards to ensure the normal operation of the provided service. This visualized topology is then translated to an NFV MANO-compliant format, along with its SLA requirements.

The next steps proceed to the composition of the virtualized network service from the NFV MANO perspective and update the Monitoring service for the newly composed network service. The Monitoring service needs to be updated in order to expect alerts and SLA violations.

- Visualization of monitoring status.
- Alert notification.

The next figure shows the portal Operational Page below:

[24] For more details see, *inter-alia*: *https://en.wikipedia.org/wiki/EnodeB*

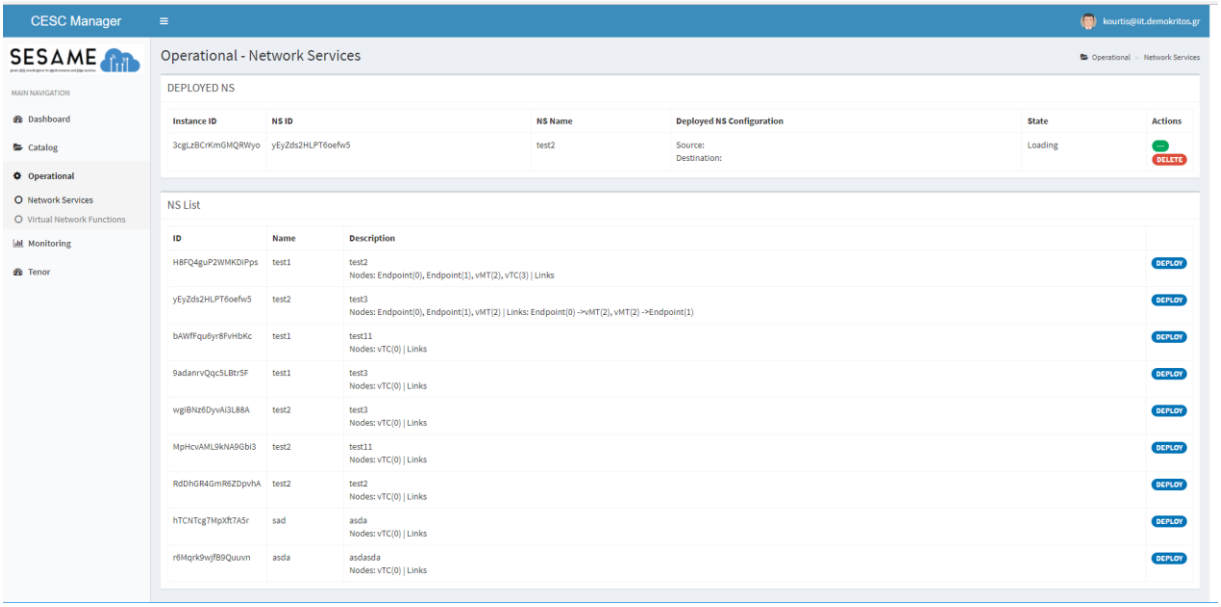

Figure 13: Operational page

In the operational page the already composed virtualized network service, can be deployed through the NFV Orchestrator to the Light DC testbed, and afterwards initiated. In this step the Monitoring service starts the SLA compliance supervising.

In case an SLA is violated in an initiated service, the Alert is pushed to the CESCM portal from the Monitoring service, which is then printed on the corresponding Monitoring page.

4.2. Northbound interface

Northbound interface is the component that conceptualizes the lower level details used by the SESAME components. The Northbound interface maps the Os-Nfvo between NFVO and VNFM, to offer a REST-*based* interface to openmano, in order to provide a user link to the services offered.

Reference points are the connection points between the components in MANO; in top of that there are the interfaces that are exposes by the different building blocks.

During the project development, it has been identified a standard that covers a number of use cases for application and service management at the Os-Ma-Nfvo (Northbound) interface. The specification it is currently developed by ETSI [9] called as *Network Functions Virtualization (NFV) Release3; Management and Orchestration; Os-Ma-Nfvo reference point - Application and Service Management Interface and Information Model Specification*, that will be release at the end of 2017.

4.3. SLA evaluation

Service Level Agreements (SLAs) represent the contractual relationship between a service consumer and a service provider in order to provide a mechanism to increase trust in providers, by encoding dependability commitments and ensuring the level of Quality of Service is maintained to an acceptable level.

In the context of SESAME a performance monitoring allows to identify when the infrastructure is reaching a level where the resources used by the tenants as services are reaching a level where performance of the system can be affected.

The SLA component follows the WS-Agreement specification[25]. This means that the documents representing templates and agreements are valid according to the schema defined in the specification.

The main functionalities are:

- Generation of WS-Agreement templates and agreements.
- Management of SLA related entities: templates, agreements, providers, violations, penalties
- Assessment of SLOs[26] and corresponding penalties when an SLO is violated.
- Notification of detected violations and incurred penalties to interested parties.

4.3.1. SLA manager

The SLA manager internal architecture is as follow:

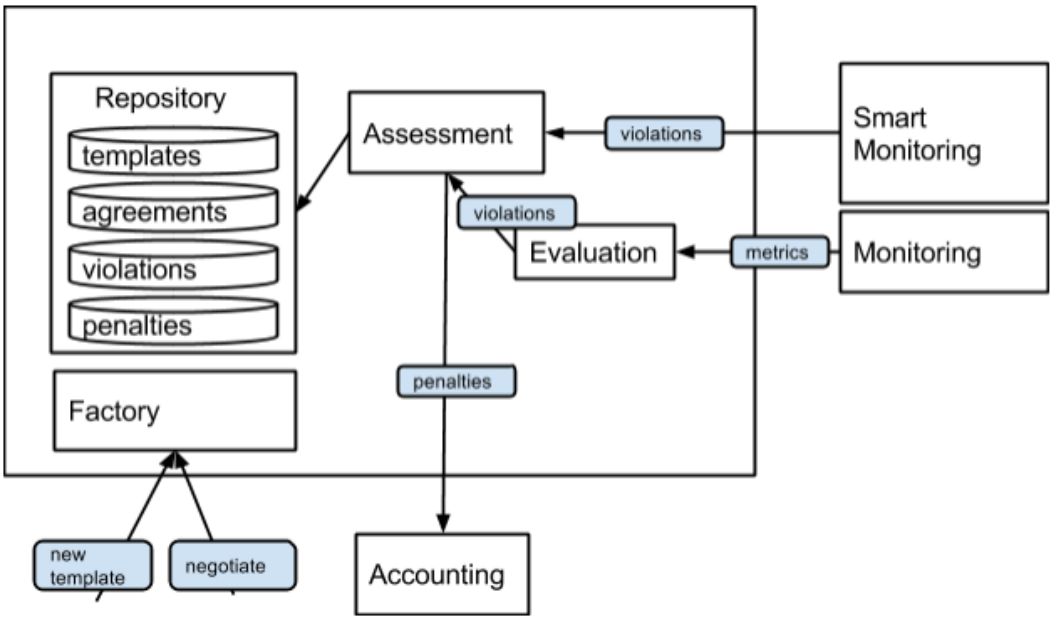

Figure 14: SLA Framework architecture

[25] For further information see, *for example*: http://wsag4j.sourceforge.net/site/wsag/overview.html
[26] Also see, inter-alia: https://en.wikipedia.org/wiki/Service_level_objective

The internal sub-modules have the following functions:

- Factory, that parses the templates and the agreements produced after the negotiation phase before storing them.
- Evaluation, that decides whether the values received from the monitoring constitute a SLA violation or not.
- Assessment, that calculates the penalties based on the produced violations.
- Repository, where all the templates, agreements and the information related to the violations and penalties generated from the SLA assessment are stored.

The first input of SLA management modules, SLA templates and SLA agreements after the negotiation phase. The templates as well as the agreements parsed are stored in the internal database. After that, the second kind of input comes from the monitoring system. The SLA management module can work with the different kind of monitoring systems available:

- Simple monitoring systems that must be polled in order to retrieve the metrics or that are able to push the metrics into the SLA core, once they are available.
- Smart monitoring systems that are able to evaluate the constraints, and raise the appropriate violations.

4.3.1.1. SLA management module

There are several actions to be performed in order to activate the SLA service. The steps are described as follow:

- Templates: Show the available SLA templates to be offered to the user.
- Agreements: Show the available SLAs that are associated to existing services.
- Enforcements: Activate the evaluation once the agreement has been signed.
- Notification: Show and send notifications of violations of an SLA.

A more detailed explanation of the steps is presented below.

4.3.1.1.1. Templates

QoS constraints and monitoring rules lead to the specification of the SLA template reported below to create the actual SLA agreement.

Templates represent a definition of the SLA terms offered by the service provider for the user to accept.

There are three available actions:

- Request available SLA templates.
- Create a new SLA template.

4.3.1.1.2. Agreements

Given a template and the customer, an agreement can be established between a customer and the service provider (the Application Provider in this case). The agreement is similar to the template, but with extra content information added in the Context section. Several agreements can be created based on a single template.

For the context of SESAME several agreements has been generated adapted to the type of service provided by the infrastructure in order to simplify the contractual process between the user and the service provider.

Agreements represent a signed SLA contract between the user and the service provider. Every agreement must be associated to an existing template in order to be registered.

There are three available actions:

- Request available agreements.
- Create a new agreement.
- Delete an existing agreement.

4.3.1.1.3. Enforcements

Enforcements determine the evaluation status of the SLA. They can be "enabled" or "disabled", meaning the SLA is currently being evaluated or not respectively.

TeNOR[27] will activate the monitoring system for the selected service once is deployed and running.

Three actions can be performed:

- Obtain active enforcements and its status.
- Enable SLA enforcement (i.e. start SLA evaluation).
- Disable SLA enforcement (i.e. stop SLA evaluation).

4.3.1.1.4. Violations

A violation occurs when a term defined in an SLA has not "met" its guaranteed value. Once the metrics are evaluated and a violation is generated as a result of the assessment, the system creates the notification with the report for a broken SLA. Then its forwarded to the endpoint defined in the module, in our case is forwarded to the orchestrator to perform the relevant action. The violation has the following format:

```
{
        'uuid':'14725e43-3e05-4c0c-b9e7-e6367ecf1882',
        'kpiName':'uplink',
        'contractUuid':'agreement-sesameradio',
        'datetime':'Sep 20 2017 2:27:30 PM',
        'serviceName':'radio-monitoring',
        'actualValue':'82.0',
        'serviceScope':''
}
```

The violations are stored in the internal database, so it can be consulted for future historical evaluation. The actions to be taken when a violation occur will be defined in the following Deliverable D6.4 (*"Orchestrator Prototype"*), where the SLA high-level workflow is described. The consequence of violation is out of scope of the SLA module.

[27] More details about TeNOR can be found at: *https://github.com/T-NOVA/TeNOR/wiki*

4.3.2. Monitoring

SLA evaluation relies upon the use of multiple solutions to retrieve the necessary metrics, to contextualized monitoring and reporting, to assess the contract terms associated with Service Level Objectives (SLOs) that have been specified in the SLA. To this end, the system is designed to be adapted to different monitoring systems; it requires additional integration with telemetry systems so that to retrieve data of the virtualized infrastructure.

There are several tools available to monitor hardware and service status, Prometheus[28] has been selected as the smart monitoring tool for SLA framework. The monitoring management receives monitoring data and manages the monitoring sub-system.

4.3.2.1. Prometheus

For the context of SESAME, Prometheus has been selected as third-party monitoring tool to assess the contract terms. Prometheus [10] is a white box monitoring and alerting system that is designed for large and scalable environments. Moreover, is an open-source service monitoring system, based on time series database that implements a highly dimensional data model, where time series are identified by a metric name and a set of key-value pairs.

It provides a flexible query language, allowing slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables and alerts, while it is integrated with visualization tools (Grafana[29] and PromDash[30]).

The data is collected in a "pull" based model, developed as exporters; allowing bridging the internal state of the application.

Additionally, it also supports ephemeral and batch job using "push" method by means of an intermediate service (PushGateway) which Prometheus can scrape. This method is only recommended for very specific cases, distinguish by the fact that do not run continuously. This approach became as single point of collection, therefore it has the potential to be a single point of failure and a potential bottleneck.

Prometheus servers scrape (pull) metrics from instrumented jobs. If a service is unable to be instrumented, the server can scrape metrics from an intermediary push gateway. There is no distributed storage. Prometheus servers store all metrics locally. They can run rules over this data and generate new time series, or trigger alerts. Servers also provide an API to query the data. Grafana utilizes this functionality and can be used to build dashboards.

Finally, Prometheus servers know which targets to scrape from due to service discovery or static configuration. Service discovery is more common and also recommended, as it allows user to dynamically discover targets. The end-points are configured in the scrape configuration file to report the statistics to the server, in order to be evaluated. Targets can be individually patterned based on the specific requirements of the specific job.

[28] See: *https://prometheus.io*
[29] See; *https://grafana.com/*
[30] See: *https://github.com/prometheus-junkyard/promdash*

The following image depicts the internal architecture of the Prometheus system.

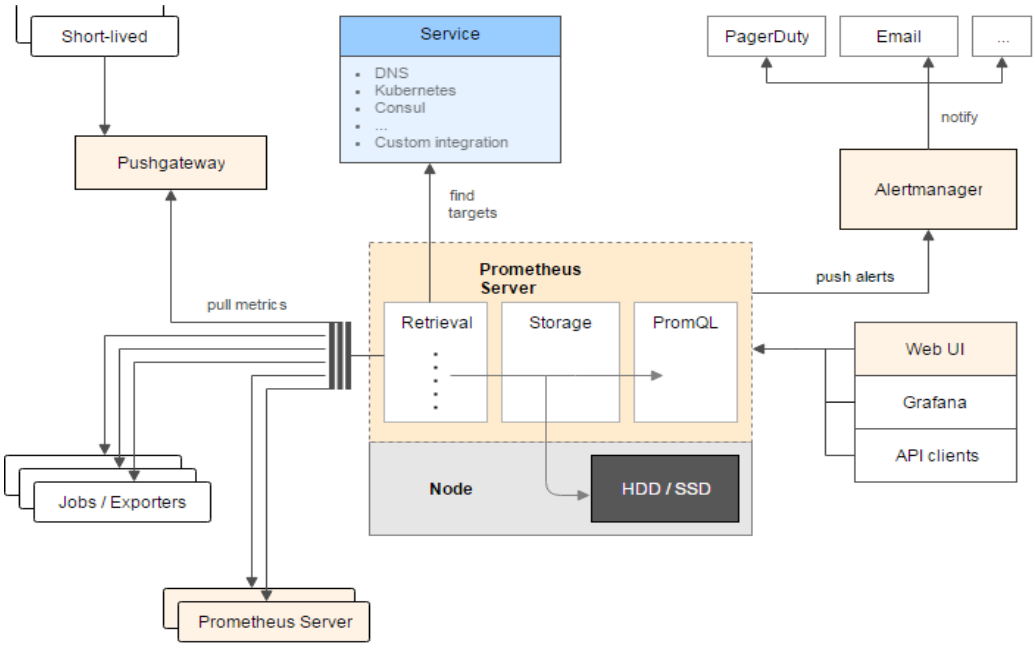

Figure 15: Prometheus system

Depending on the type of values that will generate the time series, metrics can be defined by some of the following metrics type:

- A *counter* is a metric which is a numerical value that is only incremented, never decremented. Examples include the total amount of requests served, how many exceptions that occur, etc.
- A *gauge* is an instantaneous metric value that is created via incrementing, decrementing or accumulation. An example could be memory usage, CPU usage, amount of threads or perhaps a temperature.
- A *histogram* is a metric that samples observations. These observations are counted and placed into configurable buckets. Upon being scraped, a histogram provides multiple time series, including one for each bucket, one for the sum of all values, and one for the count of the events that have been observed. A typical use case for a histogram is the measuring of response times.
- A *summary* is similar to a histogram, but it also calculates configurable quantiles. Depending on user's requirements, the user either uses a histogram or a summary.

Due to all this characteristics, Prometheus has been chosen as the external monitoring system to perform the monitoring part. It provides wide documentation and possibilities to build new exporters and its data base provides us with a powerful query language and time response that can be easily used by the orchestrator and portal.

4.3.2.2. Alerting

Alert Manager component is the responsible to handle the alerts sent by the client applications such as the cloud infrastructure, the radio deployment or the Prometheus server. It evaluates the rules defined in the system, and de-duplicating, correlating, and routing their notifications to their appropriate receiver (e.g. email, orchestrator...). These alerts are used to trigger actions in the NFVO after the alert is processed.

Alert Manager is highly configurable and supports many notification methods. The routes defined describe the level of control and management workflow to follow in the infrastructure.

Multiple alerts can be aggregated into a single notification (i.e., for an entire DC). There is one root route on which each incoming alert enters, then are forwarded to the child routes defined as the correct receiver, depending on the labelling alert. Although independent alerts can be configured, it is possible to suppress alerts and define inhibition rules; this will prevent from getting same level of information from the system that is falling in several points.

After an alert is generated and sent to the Alert Manager, it can be routed by using routes; then it can define child routes to route alerts to the correct receiver. These routes can be configured by using a YAML[31] configuration file:

```
routes:
  - match:
    continue: true
    receiver: 'tenor-notification'
    match_re:
      job: sesame-service
- name: 'tenor-notification'
 webhook_configs:
 # Whether or not to notify about resolved alerts.
 #send_resolved:  true

 # The endpoint to send HTTP POST requests to.
 - url: http://172.16.5.194:4555/notification
```

There are multiple types of receivers to which Alert Manager can push notifications, SMTP[32], HipChat[33], PagerDuty[34], PushOver[35], Slack[36] and OpsGenie[37]. Additionally, we included a web hook to send HTTP POST requests to a certain endpoint with the alert as JSON.

For our system we have selected the web hook endpoint to notify NFVO and an email notification, that can be forward to the infrastructure administrator, or the service user.

4.3.2.2.1. Web Hook notification

In order to notify orchestrator web hook receiver offers the possibility to configure a generic endpoint, our Orchestrator (TeNOR) will receive the alerts raised in the Alert Manager by the soft monitored threshold. The message with the soft alarm is sent to the endpoint defined in the system, a JSON payload with the metadada in the request body.

[31] Please see: *http://yaml.org/*
[32] For more details see, *for example*: *https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol*
[33] See: *https://zapier.com/zapbook/hipchat/*
[34] See: *https://www.pagerduty.com/*
[35] See: *https://pushover.net/*
[36] See: *https://slack.com/intl/es*
[37] See: *https://www.opsgenie.com/*

```
{
  "status": "firing",
  "groupLabels": {
    "service": "demo-service",
    "alertname": "instance_downn"
  },
  "groupKey": {},
  "commonAnnotations": {
    "summary": "Monitor service non-operational"
  },
  "alerts": [
    {
      "status": "firing",
      "labels": {
        "alertname": "instance_downn",
        "service": "demo-service",
        "instance": "localhost:8088",
        "job": "POP",
        "env": "prod",
        "service_id": "hb323kl789688g",
        "severity": "critical"
      },
      "endsAt": "0001-01-01T00:00:00Z",
      "generatorURL": "http://prometheus:9090/graph?g0.expr=up+%3D%3D+0&g0.tab=0",
      "startsAt": "2017-05-22T18:13:42.822+02:00",
      "annotations": {
        "description": "localhost:8088 service is down.",
        "summary": "Monitor service non-operational"
      }
    }
  ],
  "version": "4",
  "receiver": "tenor-notification",
  "externalURL": "http://prometheus:9093",
  "commonLabels": {
    "alertname": "instance_downn",
    "service": "demo-service",
    "job": "POP",
    "env": "prod",
    "service_id": "hb323kl789688g",
    "severity": "critical"
  }}
```


4.3.2.2.2. Email notification

Alert Manager can be integrate with email, enabling SMTP (Simple Mail Transfer Protocol) an Internet standard for email transmission that establishes communication between the client and our email service. Prometheus is configured with the SMTP keys in order to authenticate with the server that will be connected for sending the email.

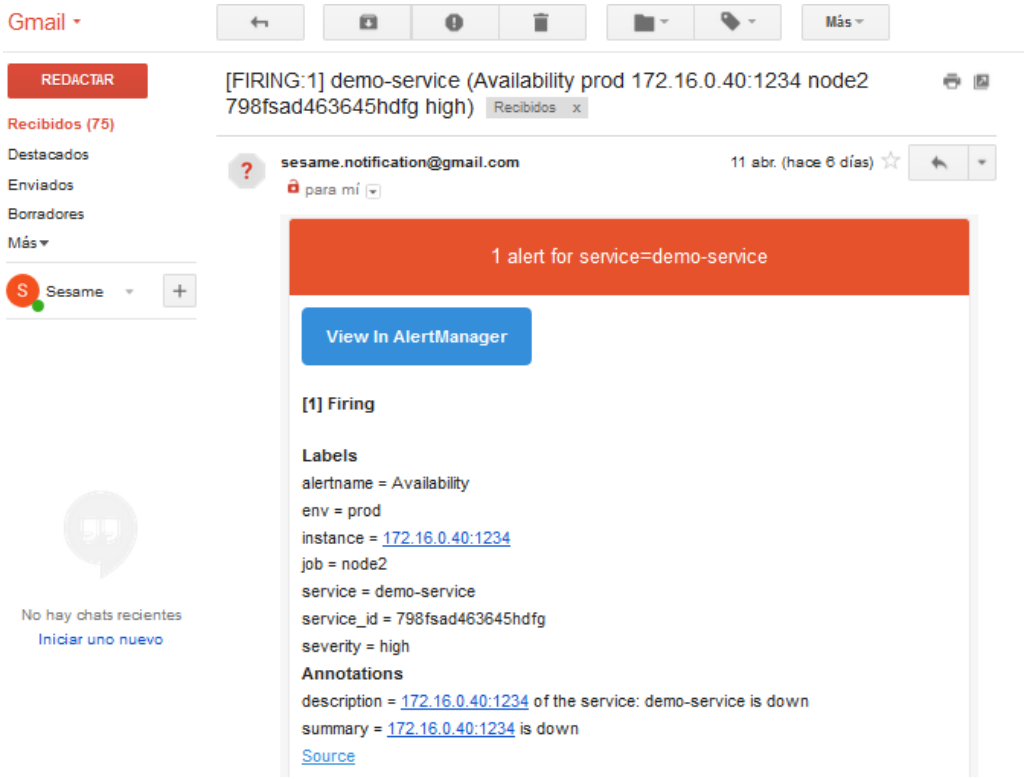

Figure 16: Email notification

Alerts can be associated with labels, and those labels can be grouped by a common characteristic (i.e.: Alertname, datacenter, service) allowing to directly notifying network operators and EU for the specific services that have been contracted; this feature implements a simple customization of email notification.

The violations are stored in the internal database, so it can be consulted for future historical evaluation. The actions to be taken when a violation occurs will be defined in the following deliverable D6.4 (*"Orchestrator Prototype"*), where the SLA high-level workflow is described. The consequence of violation is out of scope of the SLA module.

4.3.3. Dashboard visualization

Grafana is the leading graph and dashboard builder for visualizing time series infrastructure and application metrics and it allows to be added as a visualization layer to monitor Prometheus target. Although Prometheus also provides the tool to visualize the metrics, Grafana offers a much more powerful picture of the instance and it also allows to create specific dashboard to combine multiple metrics in a single view.

Grafana is configured to access Prometheus data source, by avoiding storage support, as it can be directly a query from the server. It has been also selected as the preferred dashboard due to the fact that the system already do the data gathering, data storage and alarm management.

Once Grafana is installed and configured to display the data available in Prometheus, the visualization dashboard can be access through the following IP soured: *http://localhost:3000* in a web browser.

It has been defined several Dashboard to monitor the infrastructure. *From one side*, the services are monitored as a combination of the individual resources that compose the service; this case is with greater importance when service function chain is deployed to form a service.

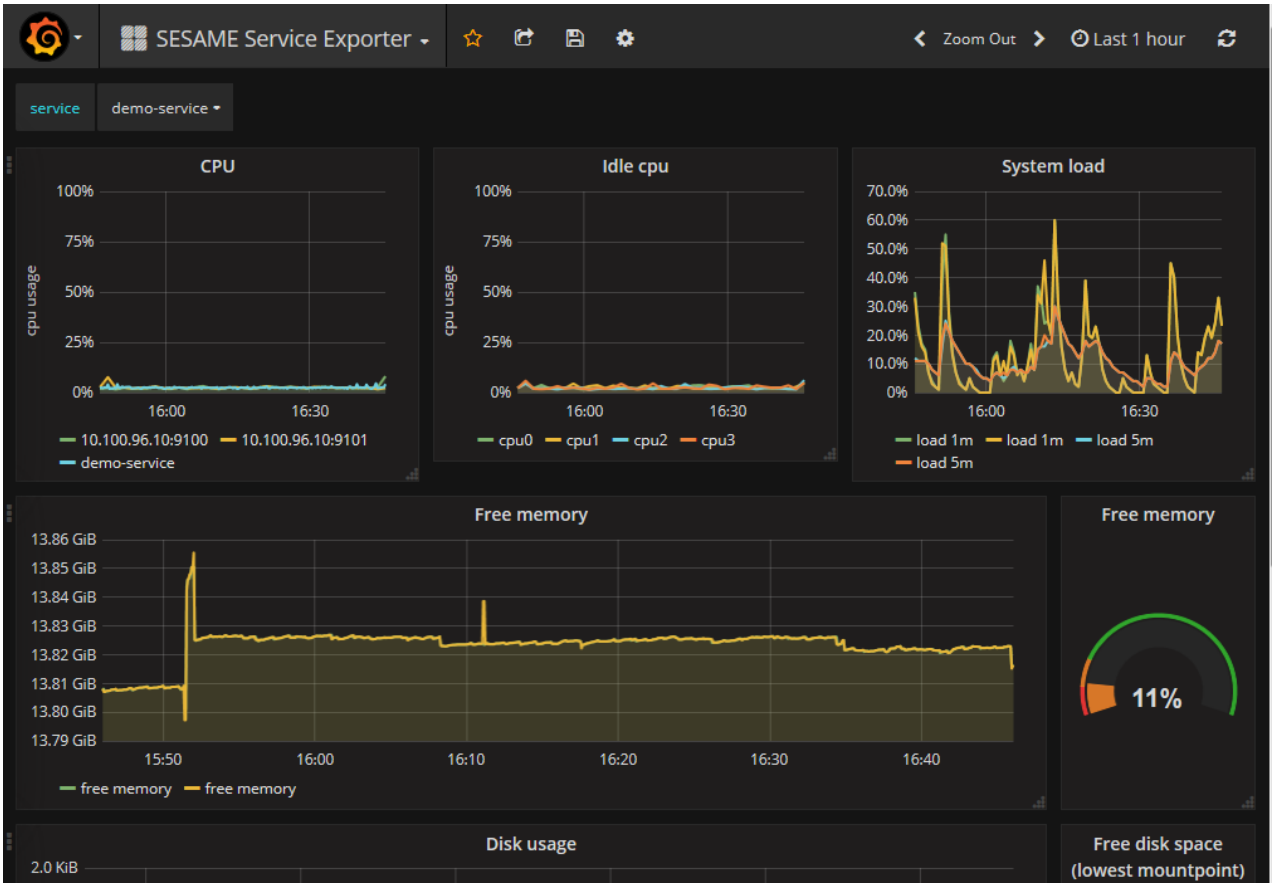

Figure 17: SESAME service monitoring

On the other hand, it is possible also to monitor individual instances that compose the service, this allow having a more detailed information of the Data Center, visualizing individual servers in the network.

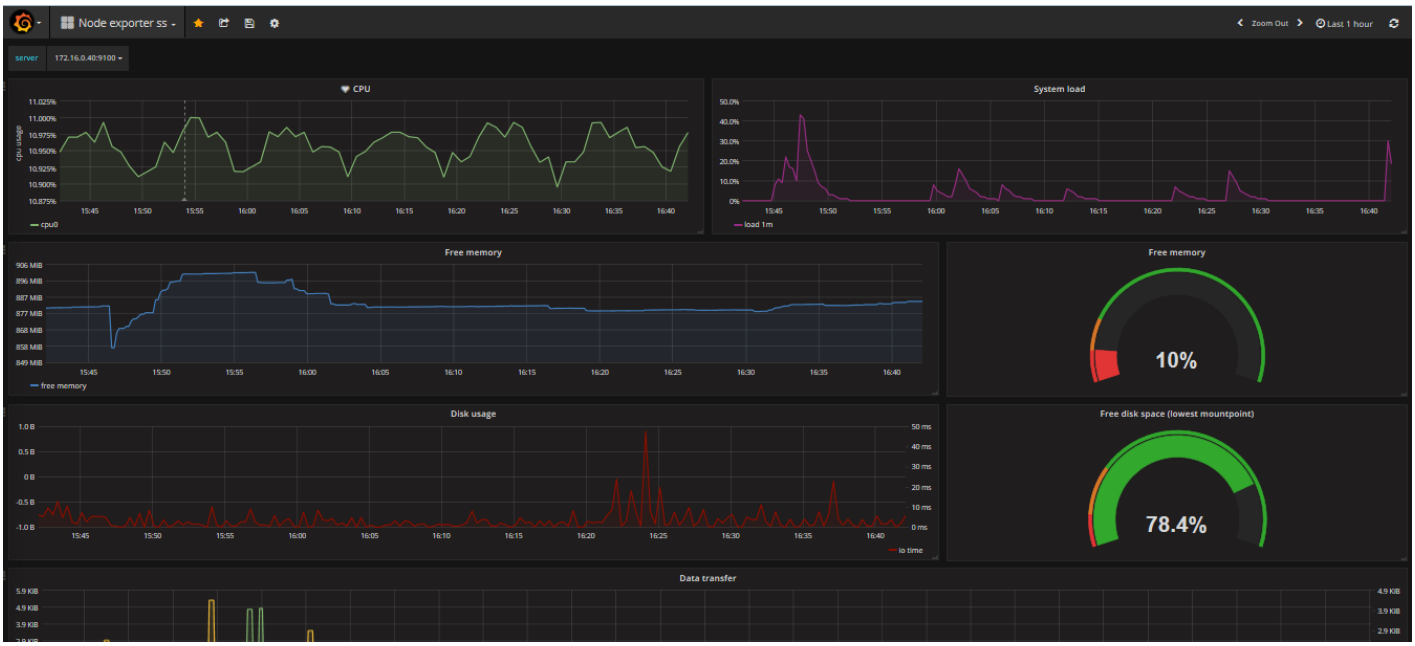

Figure 18: SESAME individual instance monitoring

4.4 NFV Orchestrator

To "close" the monitoring tool, NFVO is an essential CESCM component. It plays two main important and complimentary roles:

1- Upon instantiation of new network services, it activates the Prometheus client embedded in VMs and then communicates the required info to Prometheus (VMs IPs, etc.).
2- Once the service is up and running, it listens to Prometheus for soft and hard alerts and, depending upon the situation severity, it applies an appropriate reaction mechanism to "heal" the case. Such a reaction action might be scaling up/down resources, migrating VNF/NS, halting VNFs/NSs or any other healing mechanism foreseen in the system. The reaction mechanism is selected with the help of a logical process which takes into account parameters such as the tenant SLA requirements, resource availability, etc.

The implementation details required developments to close the monitoring loop on the NFVO side are presented in D6.4 where SESAME NFVO prototype is detailed [11].

4.5 EMS-IPA NOS

The SESAME EMS is based on the ip.access Network Orchestration System (NOS), which provides the functions of both the PNF EMS and SC EMS. The NOS is a client-server solution that comprises a single logical server that connects to each managed element and one -or more- client instances that provide a graphical user interface to users.

The functionality of an EMS is normally divided into five distinct areas as indicated by the FCAPS acronym: fault management, configuration management, accounting, performance and security management.

4.5.1 Fault Management

The fault management aspects of the SESAME EMS are based on CCITT X.733 [12]. Alarms are raised on managed objects to signify the onset of a fault condition and are cleared when the fault is resolved. The fields of an alarm include:

- Event (Alarm) Type. One of Communications, Quality of Service, Processing Error, Equipment, Environmental or Security Violation.
- Probable Cause – A unique code specifying the nature of the alarm (for example "temperatureUnacceptable"). The available values are defined in CCITT X.721 [12].
- Specific Problem – A unique code that further defines the nature of the problem. The available codes are defined in the Management Information Base (MIB) supplied with the EMS.
- Perceived Severity (Warning, Minor, Major, Critical, Indeterminate and Cleared).
- Source Managed Object Class – The class of managed object on which the alarm is raised.
- Source Managed Object Instance – The specific managed object instance on which the alarm is raised.
- Timestamp – The data and time at which the alarm was raised or last updated.
- Additional Text – Additional textual information further clarifying the cause of the specific alarm event. For example, for an over temperature alarm this might include the temperature reading at the time the alarm was raised.

Additional information included in the MIB and displayed by the EMS GUI when an alarm event is selected includes:

- Behaviour text describing the alarm.
- A list of Possible Faults – conditions that might lead to the alarm being raised.
- Suggested Repair Actions.

4.5.2 Configuration Management

The configuration management aspects of the SESAME EMS are based on CCITT X.730 [13], X.731 [14] and X.732 [15]. Managed objects are organised into a hierarchical tree structure as defined by the Management Information Base (MIB). Each managed object has a set of parameters (its attributes) that define its current state and its configuration. Optional aspects of service are provisioned by creating new managed objects and their configuration is updated by changing the value of an object's attributes. For SESAME, the MIB hierarchy contains two specific sub-trees that are of particular relevance:

- The Cloud Enabled Small Cells (CECSs) sub-tree comprises a top-level "CESCs" collection object beneath which child CESC objects may be created. Each such child object represents the configuration management aspects of a single CESC. Each CESC object has the following child objects that further define its configuration:
    - Exactly one PNF Connection object. This object represents the TR-069 [16] management connection between the EMS and the PNF. In turn, this connection

object has exactly one child PNF object that represents the configuration of the PNF.

- Exactly one SC-Common VNF Connection object. This object represents the management connection between the EMS and the SC-Common VNF hosted by the CESC. In turn, the SC-Common VNF Connection object has a set of child objects that represent the configuration of the SC-Common VNF.
- From zero to five SC VNF Connection objects. An instance of this object class is created when a virtual cell is provisioned (see below). Its child objects represent the configuration of an SC VNF instance hosted by the CESC.

The CESCs sub-tree is illustrated by Figure 19 and 20. These show, *respectively*, the logical, class hierarchy, and an instance of this hierarchy as viewed via the EMS GUI.

In Figure 21 below, the classes shown in yellow reside on the EMS. Instance of the orange VNF class reside on the CESC and the pale blue LTE class represents the PNF provided by the ip.access E40 Access Point (AP).

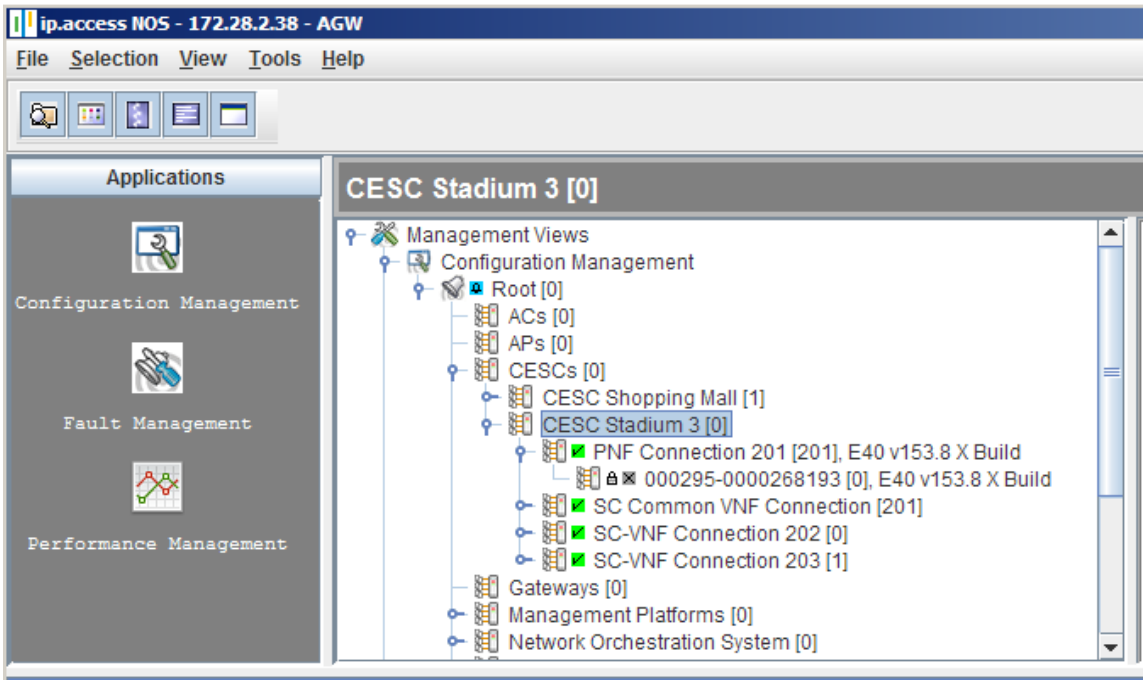

Figure 19: CESC Object Hierarchy – EMS GUI View

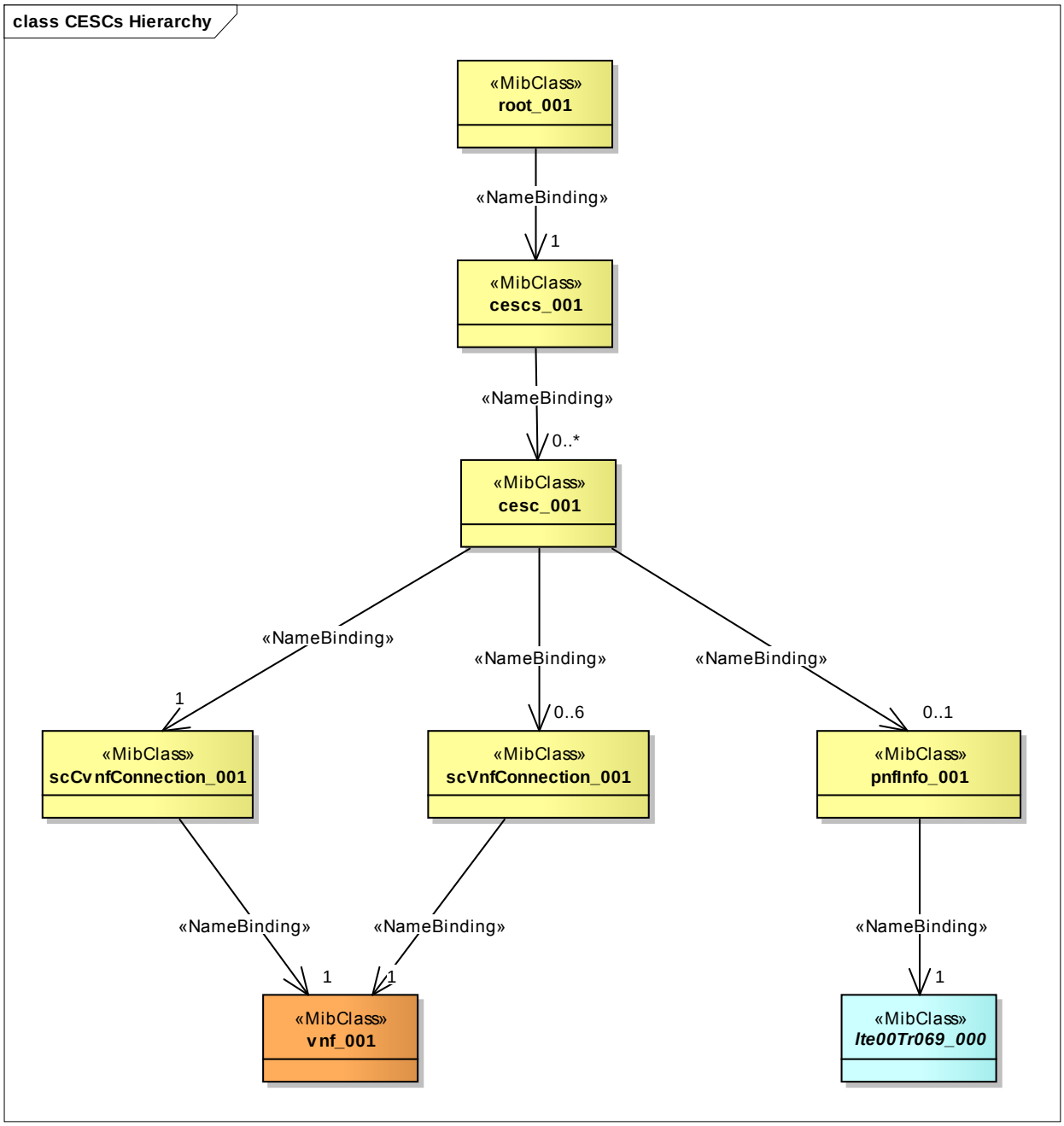

Figure 20: CESC Object Class Hierarchy

- The Virtual Small Cell Network Operators (VSCNOs) sub-tree comprises a top-level "VSCNOs" collection object beneath which child VSCNO objects are created. Each VSCNO object comprises a distinct sub-tree of managed objects that represent the network slice of an individual virtual network operator. For each virtual small cell network operator The child objects in this sub-tree include:
    - Exactly one VSCNO object. The attributes of this object define the PLMN ID of the VSCNO and the credentials for accessing northbound interfaces,
    - Exactly one MME Pools collection object. The object has one or more child MME Pool objects and, beneath each MME Pool object there are one or more child MME objects where each MME object defines the address of an MME used when provisioning a virtual cell.

- Exactly one Service Level Agreements collection object. This object has a set of child Provisioned SLA and Monitored SLA objects where each Provisioned SLA object defines a "network slice" that is applied to a virtual cell at provisioning time and each Monitored SLA object defines a set of KPIs that are monitored across a set of virtual cells as defined by the attributes of the SLA.
- Exactly one Virtual Cells collection object. As part of the virtual provisioning process, child Virtual Cell objects are created beneath this object.

Figure 21 below illustrates this class hierarchy and Figure 22 shows a possible instance of this hierarchy as displayed by the EMS GUI.

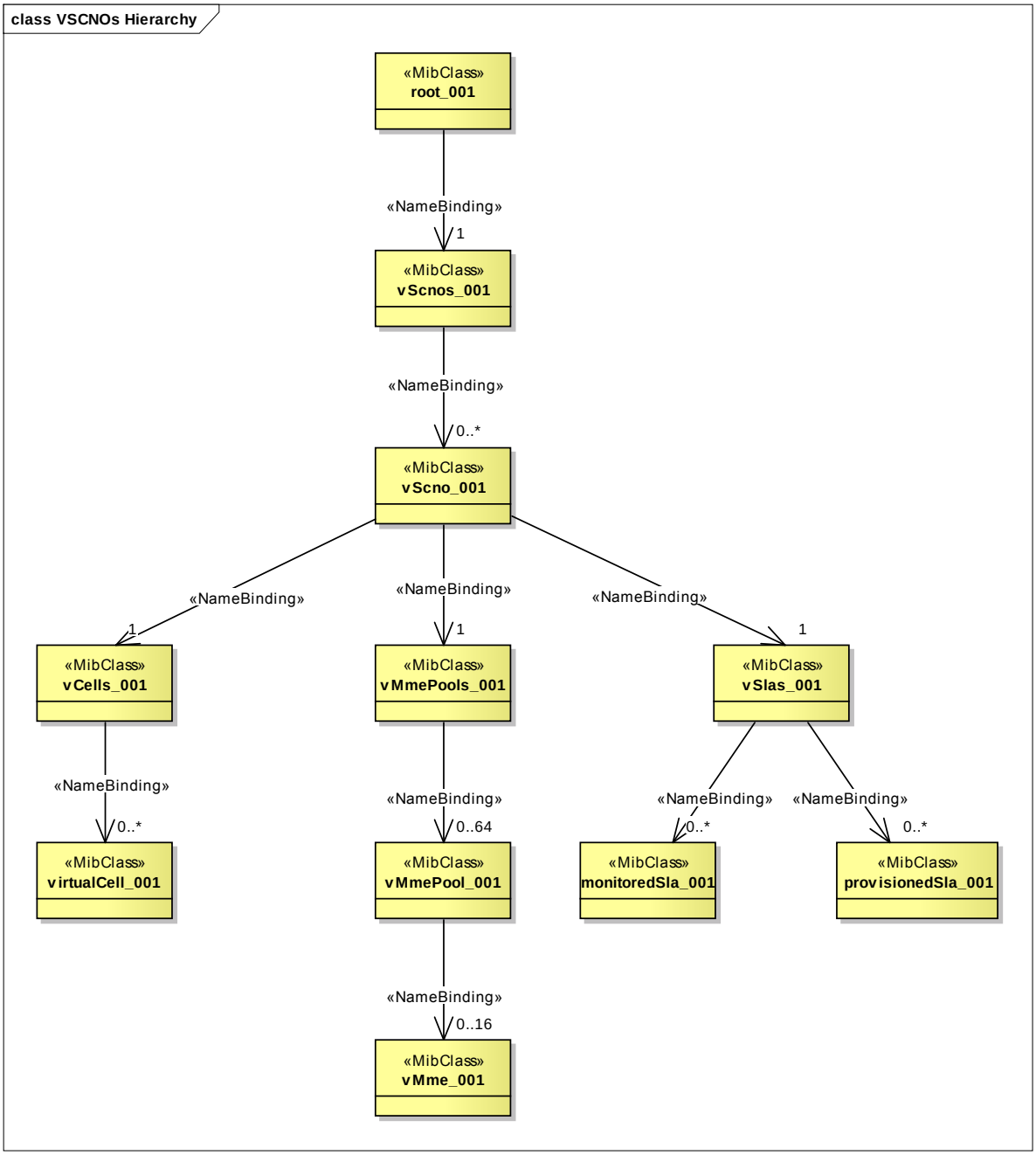

Figure 21: VSCNO Object Class Hierarchy

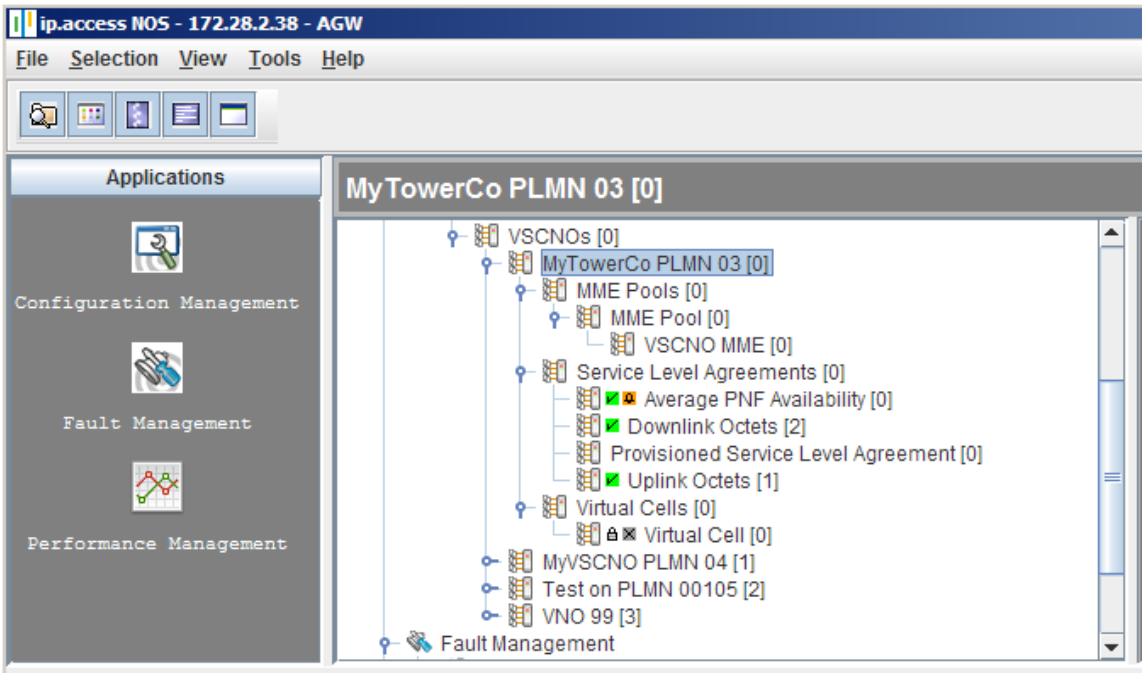

Figure 22: VSCNO Object Hierarchy – EMS GUI View

4.5.3 Accounting

The SESAME EMS does not provide any accounting functionality.

4.5.4 Performance Management

The SC PNF provided by the ip.access E40 generates Performance Management (PM) reports that conform to the XML file format specified in 3GPP 32.435 [17]. In general, the counters in this file comply with the definitions in 3GPP 36.314 [18] and 32.425 [19]. In addition, for SESAME, the PNF also implements the following counters that report performance on a per-PLMN / per-Virtual Cell basis:

- GTP-U Packets Received per PLMN without Sequence Number,
- GTP-U Packets Received per PLMN with Sequence Number,
- GTP-U Packets Transmitted per PLMN,
- GTP-U Octets Received per PLMN,
- GTP-U Octets Transmitted per PLMN,
- RRC Connection Establishment Success per PLMN,
- Maximum RRC Connection Set-up Time per PLMN,
- Mean RRC Connection Set-up Time per PLMN,
- Maximum RRC Connections per PLMN,
- Mean RRC Connections per PLMN,
- RRC Connection Re-Establishment Attempts per PLMN,
- Failed RRC Connection Re-Establishments per PLMN per Re-Establishment Cause,
- RRC Connection Re-Establishment Success per PLMN.

On receipt of PM reports from the PNF, the EMS processes the file with two objectives:

- Counter values are extracted, stored in the EMS database and used to calculate the KPI values associated with monitored SLAs. The values stored in the EMS database are made available to the SLA Manager as described in section 4.5.6

- A filtered version of each XML file is made available to VSCNOs on the EMS' northbound interface. Each file contains only counters or sub-sets of counters that are specific to the individual VSCNO.

Finally, the EMS provides the capability to forward PM reports (both those applicable to the SCNO and those applicable to a VSCNO) to multiple northbound systems by means of "upload jobs". Each upload job is defined by a managed object that captures:

- The periodicity of the upload job,
- The source data set and associated filter criteria,
- The target system address, upload protocol (SCP or SFTP), URL and access credentials.

This feature permits automated transmission of performance management data to northbound systems for analysis and action.

4.5.5 Security Management

Security Management is achieved by means of Configuration Management and Fault Management. The attributes of certain managed objects (for example the connection objects described above) specify the identify of an element that may connect to the EMS. Other aspects may include shared secrets, the IP address or range of IP addresses from which a connection can be established. Where a connection is attempted from an unrecognised address or element, the EMS raises an alarm to indicate this fact.

4.5.6 SESAME Business Logic

In addition to the SESAME specific managed object hierarchy described above, the EMS also implements the following SESAME specific business logic:

- Access permissions for GUI users allows access rights to be divided such that:
  - The SCNO may access their entire network infrastructure including the network slice of each VSCNO so that, if needed, they can perform management operations on behalf of a VSCNO.
  - Each VSNO may only view and interact with their own network slice and has no access to either the SCNO's infrastructure or another VSCNO's network slice.
- The Create Virtual Cell wizard assists a VSCNO with provisioning a new virtual cell. It offers them a list of CESCs with sufficient spare capacity to support the selected Provisioned SLA. Once a host CESC is selected, it configures the SC-VNF and SC-PNF to support the new virtual cell.
- It performs the PM file processing described in section 4.5.4 above.
- It implements an SLA Monitor function that periodically scans the set of provisioned Monitored SLA object (see 4.5.2) and computes each of the enabled KPIs across the set of virtual cells defined by the scope of the SLA. If any of these KPIs cross the configured threshold then the SLA Monitor optionally raises an alarm to indicate this fact.

4.5.6.1 Radio parameters and service implication

The master copy of radio parameters values for PNFs are stored in the EMS database and synchronised with the PNF using the TR-069 protocol [20] whenever it connects. These parameter values are defined from a number of sources that include the following:

- Default values captured in the Management Information Base (MIB).

- Values held in special “template” managed objects selected during the provisioning process.
- Values entered by the user during the provisioning process.
- Values automatically created by EMS business logic when a virtual cell is hosted on the VNF.

Once a PNF has been provisioned, adjustments to its configuration may be made in order to improve its performance. Such adjustments might include:

- Altering the PNF’s radio frequency or transmit power to adjust the coverage of the cell and optimise interference levels.
- Entering specific neighbour cells into the PNF’s list of neighbours (in addition to those detected automatically by the ANR SON function) for hand-out and possibly optimise the associated handover thresholds.

These adjustments may be achieved manually by the SCNO using the EMS client GUI or they may be performed automatically by a northbound system in response to uploaded performance management reports. The EMS provide a northbound configuration management system for this purpose.

4.5.6.2 Communication with SLA Manager

Communication with the SLA Manager is achieved by means of a set of metrics expressed in JSON format [21] uploaded from the EMS to the SLA Manager by using HTTP. A script on the EMS runs periodically to extract new KPI values from the EMS database, format them as JSON and upload them to the target address. Currently, the following KPIs are supported:

- Physical Cell Availability
- Physical Cell Call Drop Rate
- Virtual Cell Uplink Octets
- Virtual Cell Downlink Octets.

A typical JSON upload might appear as follows:

```
[
{
    "help": "Availability of the PNF measured as a percentage",
    "metric_type": "gauge",
    "collect": [
        {"metric_name": "Availability", "label": {"sc": "physical", "eid":
             "000295-0000272279", "value": "95"}}
    ]
},

{
    "help": "The total number of uplink octets transmitted by the PNF",
    "metric_type": "gauge",
    "collect": [
        {"metric_name": "UplinkOctets", "label": {"sc": "virtual", "eid":
             "000295-0000272279", "value": "200"}}
    ]
},
```

```
{
    "help": "The total number of downlink octets received by the PNF",
    "metric_type": "gauge",
    "collect": [
        {"metric_name": "DownlinkOctets", "label": {"sc": "virtual", "eid":
             "000295-0000272279", "value": "100"}}
    ]
},
{
    "help": "Call drop rate of the PNF measured as a percentage",
    "metric_type": "gauge",
    "collect": [
        {"metric_name": "PERC_CallDropRate", "label": {"sc": "virtual", "eid":
             "000295-0000272279", "value": "0.666666666"}}
    ]
}
]
```

4.6 SDN Controller

The VIM part of the SESAME project consist of OpenStack infrastructure nodes (control, compute, neutron) and a node dedicated to Netfloc[38], also acting as a physical SDN switch. As it was described in Deliverables D6.2 and D6.3, Netfloc controls the entire VIM network infrastructure by creating virtual overlay connecting the nodes and the interfaces and taking care of the flow rules installed on the OpenvSwitch.

In order to be able to expose information of the network topology and the services running in the OpenStack cloud, Netfloc has a library that exposes monitoring data. This library is native from the OpenDaylight (ODL) project, and in particular binds to the OpenFlow plugin statistics[39] collector.

4.6.1 Netfloc OpenFlow statistics collector and metrics

Netfloc uses the OpenFlow plugin APIs in order to pool metrics from the infrastructure from OpenFlow via MD-SAL[40]. The statistics are colleted in a period interval of 15 seconds. The metrics such as the following are static data collected when the node connects to the controller, and act as a descriptors to the nodes:

Table 2: Metric description

Node Description
Flow Table Features
Port Description
Group Features
Meter Features

If a new element (flow, group, meter, queue) is added to the node, it sends statistics request immediately and fetches the latest statistics. Then the statistics are stored in the operational data store of Netfloc.

Some of the other metrics that are collected on a timely base are the following:

Table 3: Statistics metric

Individual Flow Statistics	Group Statistics
Aggregate Flow Statistics	Meter Configuration
Flow Table Statistics	Meter Statistics
Port Statistics	Queue Statistics
Group Description	

Following are the RESTCONF APIs to get the various statistial data. Those have been used by Netflogi, the Netfloc GUI as well as in the Netfloc exporter library for Prometheus:

[38] See: *http://icclab.github.io/netfloc/*
[39] Also see: *https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Statistics*
[40] See: *https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_App_Tutorial*

Table 4: RESTCONF APIS

Aggregate Flow Statistics & Flow Table Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}
Individual Flow Statistics from specific table
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/flow/{flow-id}
Group Features & Meter Features Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}
Group Description & Group Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/group/{group-id}
Meter Configuration & Meter Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/meter/{meter-id}
Node Connector Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/node-connector/{node-connector-id}
Queue Statistics
GET http://<controller-ip>:8080/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/node-connector/{node-connector-id}/queue/{queue-id}

4.6.2 Netfloc exporter for Prometheus and CESCM

The Netfloc exporter for Prometheus scrapes metrics from Netfloc monitoring module and defines specific metrics that are filled in with the data collected during the setup interval that is configured in the Prometheus YAML configuration file.

In order to provide graphical representation of the continuous flow and port statistics and the current network status, the exporter is engaged with the CESCM monitoring component. The way how they interact is via simple URL setup of the Netfloc Prometheus exporter node, into the CESCM configuration file.

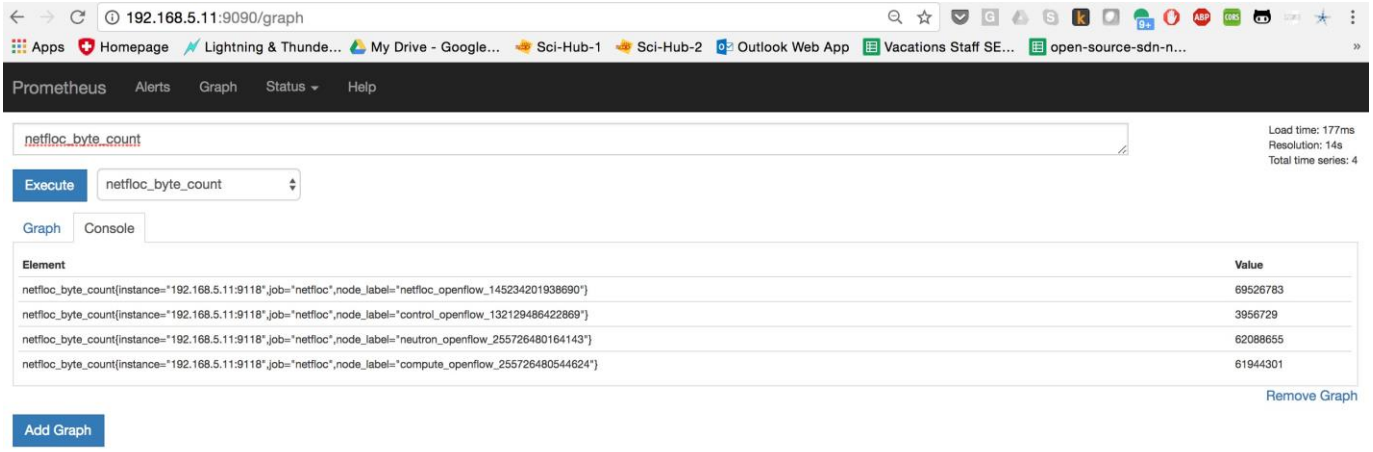

Figure 23: Metric netfloc_byte_count for each node in Prometheus

Figure 23 and 24 show the metrics that are collected (netfloc_byte_count and netfloc_packets_recieved per port), together with the representation of the netfloc_active_flows metric in Grafana, Figure 25.

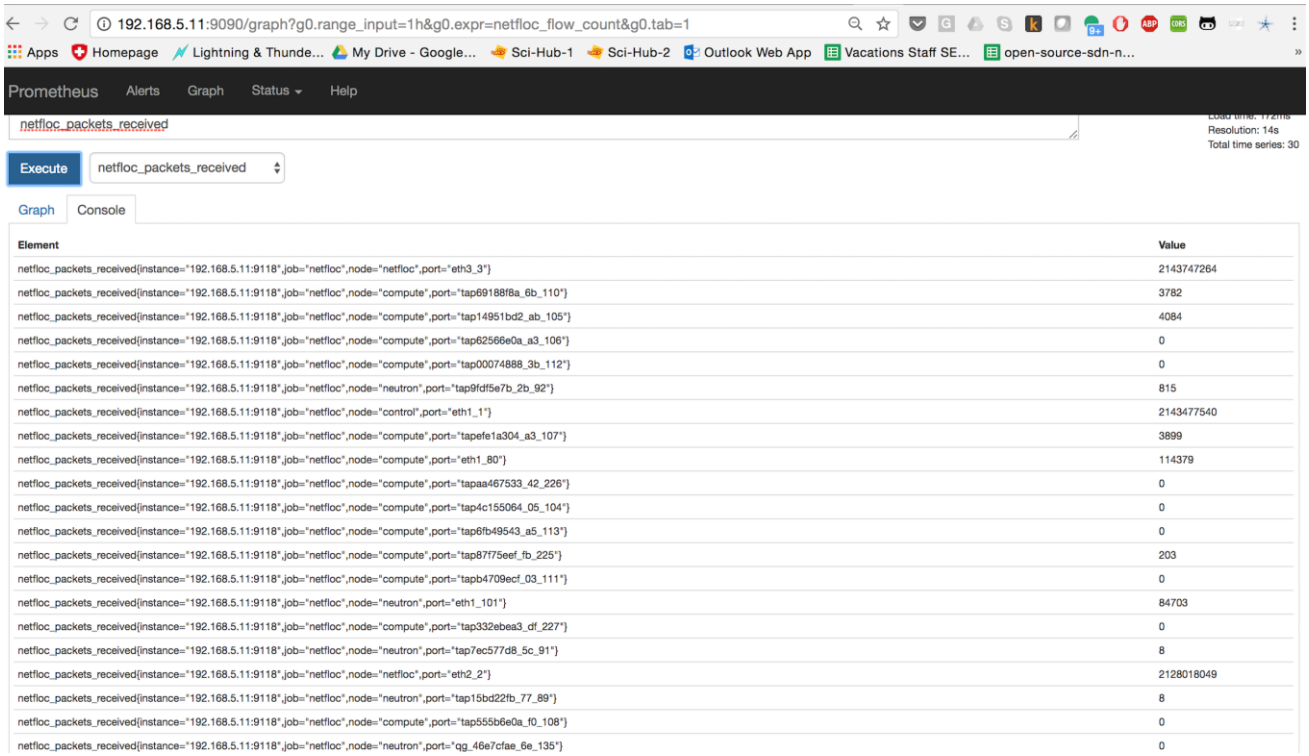

Figure 24: Metric netfloc_packets_received for each port and node in Prometheus

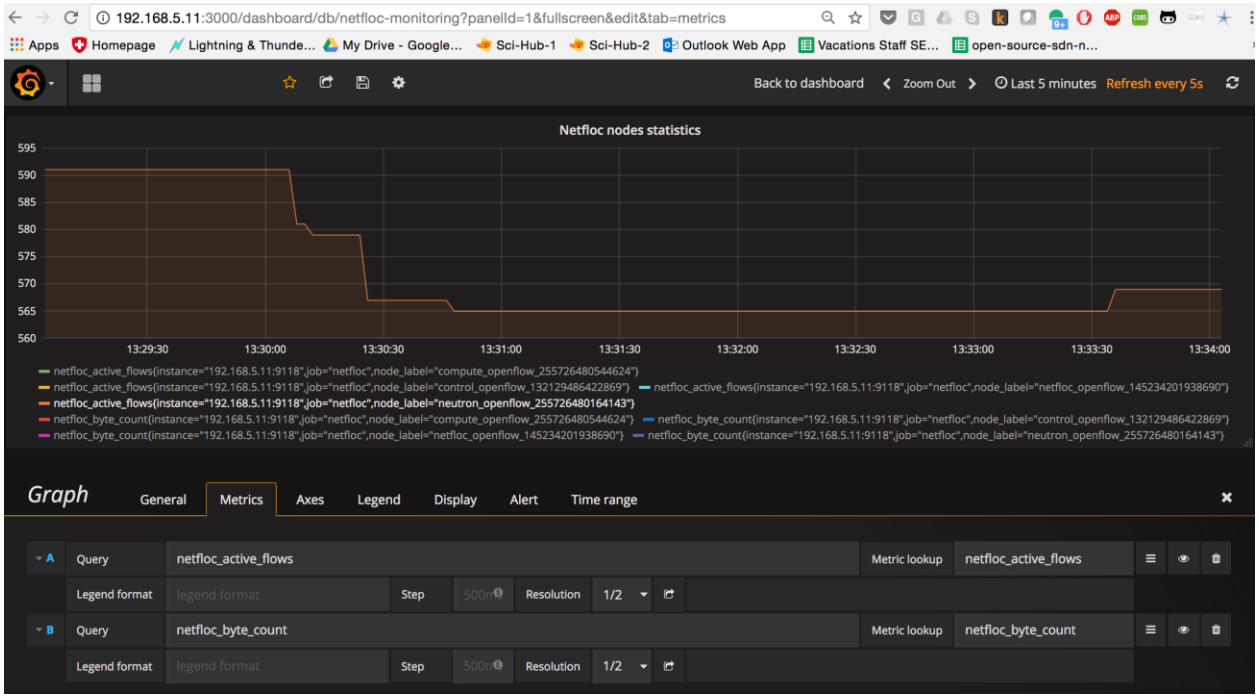

Figure 25: Metric netfloc_active_flows for each port and node in Prometheus

Netfloc also implements a custom SFC-*related* metrics, which are associated to the flows with priority=20. Those flows are the ones installed on the OpenvSwitch in order to enforce the chaining rules once TeNOR invokes the Netfloc's Create Chain API. The metrics appearing in that case are the following:

- flow-duration
- flow-packet-count
- flow-byte-count

With other words, what these metric show is the duration of the SFC flow in each of the nodes where SFC fllows are installed (e.g., where the VNFs that belong to the chain are running), as well as the number of packets and bytes crossing the port with mac address listed in the label of the metric, as shown in the example metric below:

- *netfloc_flow_duration{flow="ServiceChainEndRewrite_2_1_00_00_e8_94_f6_08_53_70",node="compute"} 680090.0*
- *netfloc_flow_duration{flow="ServiceChainRewrite_2_0_00_00_e8_94_f6_08_53_70",node="compute"} 680090.0*

Additional information on the Netfloc and the exporter can be also found in the deliverable D6.3.

5 Conclusions

The following document describes the work performed in the Task 5.2 *("VIM and CESCM implementation"*) as part of the WP5 ("*Infrastructure Virtualisation and Management*"), this is an important part of the infrastructure, as is in charge of managing the resources available for the service deployment in the SESAME platform.

Current status of NFV MANO and how this architecture is applied in SESAME project to carry on the objectives of the CESCM has been discussed. A first overview of its components is also presented.

The VIM was presented with the modifications done to cope with the ARMv8, and adaptation on the OpenStack agent to run on NUMA architecture. This report includes a summary of the main modifications for VNF placement that is also completed in the task, more detailed information is presented in Deliverable 5.3 as report of the specific task.

Furthermore, the Cloud Enabled Small Cell management components are described in the different subsection to better understand the functioning of the system. The Portal is the front-end interface from where the user selects the services provided, is the starting point of all management service. It provides also the visualization of the infrastructure thanks to the dashboard developed for the different infrastructures; the SLA status of the services acquired is also displayed. On the other hand, the SLA evaluation is working in the background of the platform to assure the QoS of the services provided in the infrastructure, the SLA framework is compliant with the WS-agreement specification; all this features are described in the document. The orchestrator (TeNOR), realizes the needed actions to close the looping flow of the service lifecycle management. Moreover, the EMS of the Small Cell presented the functionalities to provide the functioning of the PNF and VNF associated to the NOS system. Finally to assure the networking communication of the services, a Netfloc exporter has been developed to monitor the network configuration that at the same time has been integrated in the monitoring system of the infrastructure.

This document concludes the work performed in WP5 and complies with the architecture to be integrated in the final PoC of SESAME infrastructure in WP7.

6 References

[1] *https://www.ietf.org/proceedings/88/slides/slides-88-opsawg-6.pdf*

[2] *http://www.virtualopensystems.com/en/solutions/guides/yocto-qemu-kvm-vswitch-nxp-ls2085a/*

[3] *https://www.docker.com/*

[4] *https://kubernetes.io/*

[5] *https://wiki.openstack.org/wiki/ContainersTeam*

[6] https://wiki.openstack.org/wiki/Magnum

[7] *https://wiki.openstack.org/wiki/Docker*

[8] *http://libvirt.org/*

[9] *https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp*

[10] *https://prometheus.io*

[11] SESAME Project, Deliverable 6.4: "*Orchestrator Prototype*", *September 2017*.

[12] CCITT X.721, Information Technology – Open Systems Interconnection – Structure of Management Information: Definition of Management Information. Geneva, 1992.

[13] CCITT X.730, Information Technology – Open Systems Interconnection – Systems Management: Object Management Function. Geneva, 1992.

[14] CCITT X.731, Information Technology – Open Systems Interconnection – Systems Management: State Management Function. Geneva, 1992.

[15] CCITT X.732, Information Technology – Open Systems Interconnection – Systems Management: Attributes for Representing Relationships. Geneva, 1992.

[16] CCITT X.733, Information Technology – Open Systems Interconnection – Systems Management: Alarm reporting function. Geneva, 1992.

[17] TR-069: *CPE WAN Management Protocol, Issue 1 Amendment 5*. Broadband Forum, November 2013.

[18] 3GPP 32.425: *Performance Management (PM); Performance measurements; Evolved Universal Terrestrial Radio Access Network (E-UTRAN) (Release 10)*, March 2010.

[19] 3GPP 32-435: *Telecommunication management; Performance measurement; eXtensible Markup Language (XML) file format definition (Release 10)*, March 2010.

[20] 3GPP 36.314: *Evolved Universal Terrestrial Radio Access (E-UTRA); Layer 2 – Measurements (Release 10),* September 2011.

[21] JavaScript Object Notation, *www.json.org*