D6.1 Definition of the Pilots, infrastructure setup and maintenance report

Project Acronym	SONATA
Project Title	Service Programming and Orchestration for Virtualised Software Networks
Project Number	671517 (co-funded by the European Commission through Horizon 2020)
Instrument	Collaborative Innovation Action
Start Date	01/07/2015
Duration	30 months
Thematic Priority	ICT-14-2014 Advanced 5G Network Infrastructure for the Future Internet

Deliverable	D6.1 Definition of the Pilots, infrastructure setup and maintenance report
Workpackage	WP6 Infrastructure setup, validation and pilots
Due Date	30th June, 2016
Submission Date	12th Oct, 2016
Version	1.1
Status	FINAL
Editor	George Xilouris (NCSRD), Alberto Rocha (AlticeLabs)
Contributors	Sharon Mendel-Brin (NOKIA), Felipe Vicens (ATOS), Santiago Rodríguez (Optare), Stavros Kolometsos (NCSRD), Luís Conceição (UBI), Panos Trakadas (SYN)
Reviewer(s)	Michael Bredel (NEC), Pedro A. Aranda Gutirrez (TID)

Keywords:

Infrastructure flavours, pilots

Deliverable Type		
R	Document	
DEM	Demonstrator, pilot, prototype	
DEC	Websites, patent filings, videos, etc.	
OTHER		**X**
Dissemination Level		
PU	Public	**X**
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary:

This document presents the details of the first phase of the activities associated with the specification of the SONATA infrastructure flavors and the elaboration on the pilots. During this first phase, the focus has been on preparing the hosting infrastructure where the anticipated SONATA infrastructures would be deployed and initial validation and integration test would be executed. The resulted deployments will support, in full, the first year demonstrations. Particularly, the physical topology of the deployed NFVIs is given as well as the specification of the software and hardware components used.

The aim of this deliverable is not only to present the technical progress of the project in the field, but also to constitutes a rough technical guide for the installation and integration of SONATA artifacts. At this stage the document's importance is to complement the Open sourcing activity for SONATA artifacts, constituting a rough guide for their deployment. However as the project carries on, future versions of this document will be enriched with additional and refined information.

The document elaborates on the state-of-art used for the selection of the appropriate technologies or used as examples for concluding on the SONATA approach. The focus of the SOTA was on fields like: (i) virtualisation enablers, that could be exploited for the deployment of SONATA as well as for the implementation of the VNFs; (ii) Virtual Infrastructure Management (VIM) solutions that could be considered as targets for the Infrastructure Adaptation component and at last (iii) open source infrastructure deployment automation frameworks. The survey included as well a summary of implementations coming from EU-funded projects like T-NOVA and UNIFY.

The deliverable continues with the definition of the SONATA infrastructure flavors, namely: Integration Infrastructure, Qualification Infrastructure and Demonstration Infrastructure. In order to support the automatic deployment and maintenance of the aforementioned infrastructures, the assumption of the existence of a virtualisation capable underlying infrastructure (host infrastructure) was made. In addition, the selected technology for managing and running Host infrastructure was Openstack and more precisely the Brahmaputra OPNFV release (it deploys Openstack Liberty release). For each infrastructure version a detailed deployment and configuration guide is included, ranging from the Continuous Deployment / Continuous Integration (CI/CD) framework - used for the development of both SONATA artifacts but also as a model for the future SONATA developers- to the deployment of SONATA SDK and SONATA Service Platform.

Regarding the operations, a specific section is devoted to this part. The section discusses the supported operations over the infrastructure and provides insight regarding the operation and monitoring of critical components (e.g. Jenkins servers).

Finally, the document concludes with the definition of pilot selection methodology that is applied over the first year Use Cases. The methodology uses multiple criteria such as:

- Project KPIs
- Project Objectives
- SONATA functionalities coverage
- Workload characteristics variety
- Feasibility of pilot implementation

Checking the considered first year Use Cases against the criteria mentioned, yields that, at the time of writing, the most applicable Use Case to be piloted as a whole is the vCDN use case. At the same time focused pilots for particular SONATA functionalities that are not currently covered are specified.

Contents

List of Figures

List of Tables

1 Introduction

SONATA aims at delivering a DevOps framework that will enable the exploitation of the softwarisation capabilities offered by the NFV and SDN environments. These technologies provide mechanisms to automate resource intensive processes of network operators. However, on their own, they are not enough to drive transformation. Building on top of them, SONATA will enable management and control of the DevOps cycle for the development, continuous deployment and integration of added value services and their components.

In this environment the focus area of SONATA is mainly on the software development and service life-cycle management (before and during operation). However in order to support the demonstration of the high layer capabilities and results of SONATA framework, a consistent infrastructure supporting virtualisation of HW resources, i.e. processing power, memory, storage and network resources plus high level of softwarisation both in networking and IT infrastructure.

For this reason, the approach used is the introduction of incremental steps towards integration, validation and testing of SONATA components exploiting three environments (aka infrastructure versions). These infrastructures are (1.) integration infrastructure (II); (2.) qualification infrastructure (QI) and (3.) demonstration infrastructure (DI). This deliverable attempts a definition and specification of these infrastructures as well as presents the implementation of these infrastructures (II and QI for the first year of the project lifetime). Furthermore, operational results are presented, highlighting the workloads created by the preliminary implementations of SONATA components along with maintenance activities.

This document provides both technical progress on the specification, deployment, operation and maintenance of the SONATA infrastructure flavours. In addition defines the Pilot selection strategy that will be followed in the course of the project. The aforementioned activity will allow the definition and refinement of the SONATA Pilots taking into account the UCs that were used for requirements elicitation in [12] or/and also define new ones that will appropriately highlight the SONATA outstanding functionalities.

The document is structured as follows:

Chapter 1 is the introduction of the deliverable discussing the contribution and scope of the document.

Chapter 2 Section 2 surveys relevant technologies and frameworks for the deployment automation of components that will form the SONATA infrastructure versions, support the virtualisation of resources and provide sufficient management and control (through open APIs) over those physical infrastructures used for the deployment of network services.

Chapter 3 Section 3 discusses the specification and deployment of the SONATA Integration, Qualification and Demonstration infrastructures, as well as the Host infrastructure.

Chapter 4 Section 4 presents the pilot selection strategy and assorted criteria used for the selection/creation of SONATA Pilots.

Chapter 5 Section 5 concludes the document.

1.1 Document Interdependencies

This deliverable integrates the work carried out so far within the rest technical WPs, and as such, contains either implicit or explicit references to deliverables summarised in the following table.

Table 1.1: Document Interdependencies

Deliverable Name	Description	Reference
D2.1 - Requirements Elicitation	This deliverable is relevant due to the elaboration on the initial Use Cases envisaged by SONATA in order to elicit requirements. The Use Cases are used in this deliverable in order to lead to pilot specification and specific demonstrators. Furthermore, clarifications on the specification of the SONATA infrastructure flavours are fuelled by this deliverable.	[12]
D2.2 - Architecture Design	The initial architecture design laid out in this document specifies the southbound interfaces and the expected components that are anticipated in the supported NFV infrastructure in order to deploy Network Services. In addition initial understanding of the SONATA Service Platform deployment over operator infrastructure is acquired from this deliverable.	[13]
D3.1 - SDK prototype	This deliverable has relevant information that affects the specification of the Integration and Qualification infrastructure flavours.	[15]
D4.1 - Orchestrator prototype	This deliverable has relevant information w.r.t. to the definition of the Infrastructure Adaptor, component that interacts directly with the NFV infrastructure manager (VIM) or WAN infrastructure manager (WIM)	[16]
D5.1 - Continuous integration and testing	This deliverable has relevant information w.r.t. to the definition and specification of the Continuous Integration / Continuous Deployment (CI/CD) methodology followed by SONATA. The established methodology and identified integrated components are exploited by this Deliverable in order to precisely define the Integration/Qualification infrastructures. Moreover this deliverable also elaborated on initial results from WP6 on the infrastructure versions.	[14]
D5.2 - Integrated Lab-based SONATA Platform	This deliverable is synchronous to D6.1, it serves as a first integration of the SONATA platform and elaborates on the integration tests for the SONATA components. The results of this effort will be transferred to the integration and qualification infrastructure for further testing and validation. From the D6.1 scope, this deliverable provides a more technical insight on the components integration and interfacing with the underlying infrastructures.	[17]

2 State of the art survey

This section provides an overview of the state-of-the-art technologies and mechanisms relevant to the scope of this deliverable, which is the documentation of the infrastructure setup the maintenance and an early elaboration on the anticipated pilots. In summary the section discussed the virtualisation enablers relevant to the project i.e. container-based and Virtual Machine (VM)-based solutions, then elaborates on the Virtualised Infrastructure Management (VIM) solutions that are available presently and finally the document concludes referencing some solutions provided by EU funded projects.

2.1 Virtualisation Enablers

2.1.1 Virtualisation modes

Virtualisation allows to run many guest virtual machines on top of a host operating system. There are many different virtualisation technologies, which can be categorised into four main families based on the levels of abstraction [27]:

2.1.1.1 Hardware Emulation

Hardware emulation is created on a host system to emulate the hardware of interest. The main advantage of this complex virtualisation is the flexibility it gives, e.g. an unmodified operating system intended for a PC can run on an ARM processor host. Furthermore multiple emulators, each emulating a different processor, can run on the same system.
The main problem with hardware emulation is that it can be excruciatingly slow as every instruction must be simulated on the underlying hardware.
This type of "virtualisation" is out of SONATA scope and will not be supported.

2.1.1.2 Full virtualisation

Full virtualisation AKA as native virtualisation, uses a virtual machine that mediates between the guest operating systems and the native hardware. One of the main challenges in this type of virtualisation is that the underlying hardware isn't owned by an operating system but is instead shared by it through the hypervisor. This method is faster than hardware emulation, but performance is less than bare hardware because of the hypervisor mediation. The biggest advantage of full virtualisation is that an operating system can run unmodified. The only constraint is that the operating system must support the underlying hardware.

2.1.1.3 Para-Virtualisation

This virtualisation method uses a hypervisor for shared access to the underlying hardware but integrates virtualisation-aware code into the operating system itself. Using this type of virtualisation usually offers performance near that of an non-virtualised system. Also here multiple different operating systems can be supported concurrently.

2.1.1.4 Containers (Operating system-level virtualisation)

Servers are virtualised on top of the operating system itself. This method supports a single operating system and simply isolates the independent servers from one another. The level of resource isolation achieved here is high and so is the level of performance.

2.1.2 Container-based Virtualisation vs. Hypervisor-based Virtualisation

The two most common and relevant to SONATA virtualisation families are container-based virtualisation and hypervisor-based virtualisation. Comparing the two reveals the lightweight nature of containers (see Figure 2.1).

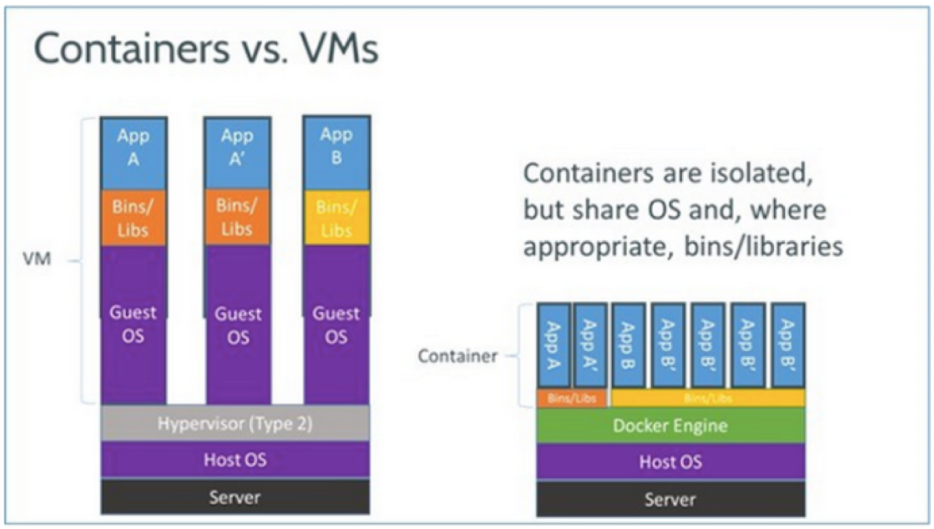

Figure 2.1: Containers vs VM

Hypervisor-based virtualisation is based on emulating the hardware while containers are based on sharing the operating system. This means that instead of running a hypervisor and a guest operating system and the applications on top of the guest operating system, the applications running in container share the kernel of the host operating system. They can also share additional user space libraries and binaries. Since in case of container based virtualisation there is no need to start an additional guest operating system and additional libraries and binaries but to use and share an already running kernel, the start-up speed of the applications is very high [35]. Containers offer high level of density, allow dynamic resource allocation and have almost native bare metal performance. On the other hand containers still have a disadvantage of not being able to run on the same hardware platform, applications that require different kernel versions or kernel modules, because all applications share exactly one kernel, that of the host operating system.

The following sections give a detailed overview of these two virtualisation approaches.

2.1.3 Container Based Virtualisation

Hypervisor-based virtualization and container technology are similar in that both ensure an environment for application isolation. This isolation is important from resource access and usage point of view. This is especially important when two applications are running on the same hardware environment and they are competing for the same type of resources. This greater isolation is on cost of greater overhead. Reducing this overhead is the biggest advantage of container technology

and allows an application to start even 100 or 1000 times faster than in case of hypervisor- based virtualization. Container commodity tools, as detailed in the following sections, not only ease container creation but introduce the notion of container image and image repository. By doing this, container commoditisation tools make not just the SW delivery but also the application delivery much faster. Continuous application delivery is also a benefit as result of the reduced time of SW delivery. This causes an increased agility based on reducing the provisioning time between development and testing. Containers are lightweight, regarding speed and size. Size is much smaller compared to virtual machines meanwhile the start-up speed is noticeably faster. [35]

2.1.3.1 LXC

LXC is a user-space interface for the Linux kernel containment features. It lets Linux users easily create and manage system or application containers through a set of API and tools [29].

LXC containers are often considered as something in the middle between a change rooted environment (c.f. chroot - an operation on Unix operating systems that changes the apparent root directory for the current running process and its children [38]) and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.

Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)

Components

LXC is currently made of a few separate components:

- The liblxc library
- Several language bindings for the API:
  - python3 (in-tree, long term support in 1.0.x and 2.0.x)
  - lua (in tree, long term support in 1.0.x and 2.0.x)
  - go
  - ruby
  - python2
- A set of standard tools to control the containers
- Distribution container templates

Licensing

Most of LXC code is released under the terms of the GNU LGPLv2.1+ license, some Android compatibility bits are released under a standard 2-clause BSD license and some binaries and templates are released under the GNU GPLv2 license. The default license for the project is the GNU LGPLv2.1+.

2.1.3.2 Docker

Docker [23] is an Open-Source (Go) framework to manage "container virtualisation". Docker developers realised that not just container creation is difficult but also it is not easy to persistently save a container content like in case of virtual machines. The basic difference between Linux containers and Docker containers is that Docker adds a command line interface to container creation, a REST API and most importantly an image repository to store modified or pre-installed container images. [35]

Features

The Docker images conceptually are very similar to VM images as they store a snapshot of the installed application but the main difference is that when an user modifies this image and wants to store the modified image, then only the modification is stored and not the entire modified image. This is important from the perspective that an image modification can be used quickly not just on the place of modification but pushing it to a central repository is much easier because of the total size of the content committed to repository is much smaller than committing the full modified image. There are public and private registries in order to upload and download container images, the public Docker registry is called Docker Hub. A Docker container holds everything that is needed for an application in order to run.

Components

Docker consists a Docker engine and one or more Docker clients, Docker clients can initiate operations like pull or run Docker images by Docker engine and the engine itself will actually create the containers on the host where the daemon is installed. The user does not directly interact with the engine but through the Docker client. The Docker client is the Docker binary and is the primary user interface to Docker. An overview of all components [37]: Sonia's corrections

- Docker Engine: the daemon managing docker images and containers (using namespaces and cgroups). It runs on the (Linux-based) Host.
- Docker client: the binary interacting with the Docker Engine.
- Docker Image: a filesystem (read-only template) used to create a Container (think "the binary")
- Docker Container: a running image providing a service (think "the process")
- Host: the computer running the Docker Engine
- Docker Registry: a private or public (Docker Hub) collection of Docker Images
- Docker Machine: provision hosts and install Docker on them
- Docker Compose: create and manage multi-container architectures
- Docker Swarm: orchestrating tool to provision and schedule containers

Licensing

The Docker project has not currently adopted a Code of Conduct though there is work being done in this regard. It is anticipated that participants and contributors follow the generic contributor covenant until a formal code of conduct is adopted.

2.1.3.3 Key differences between LXC and Docker

Dockers has been initially based on LXC, however their implementation w.r.t. separation and isolation is different. The figure below Figure 2.2 illustrates the deployment differences.

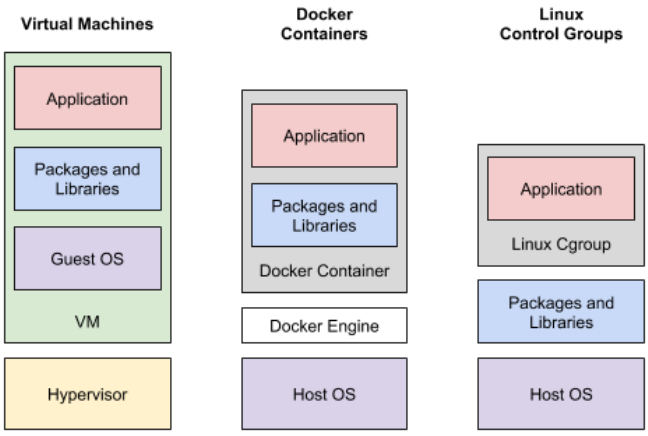

Figure 2.2: LXC vs Docker containers

Dockers compared to LXC main differences are:

- New behaviour in terms of network, storage and management.
- Development friendly model with the Dockerfile and commits.
- No particular OS know-how is needed.
- Share and distribute apps via Docker Registry.

2.1.4 Hypervisor-based Virtualisation

Hpervisor-based Virtualisation aka Virtual Machine Based Virtualisation is based on emulating the hardware and makes it possible for one hardware to run applications together that share different kernel versions or kernel modules.

2.1.4.1 QEMU

QEMU is a generic and open source machine emulator and virtualiser. QEMU supports two modes of operation. The first is the *Full System Emulation mode*. In this mode QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). By using dynamic translation, it achieves very good performance. QEMU also supports a second mode called *User Mode Emulation*. In this mode, which can only be hosted on Linux, a binary for a different architecture can be launched. In this mode QEMU achieves near native performance by executing the guest code directly on the host CPU [4].

2.1.4.2 KVM

Kernel-based Virtual Machine (KVM) is a hypervisor built into the Linux kernel. Unlike native QEMU, which uses emulation, KVM is a special operating mode of QEMU that uses CPU extensions (HVM) for virtualisation via a kernel module. [2]

Using KVM, one can run multiple virtual machines running unmodified GNU/Linux, Windows, or any other operating system. (See Guest Support Status for more information.) Each virtual machine has private virtualised hardware: a network card, disk, graphics card, etc.

KVM is a full virtualisation solution, that is unique in that it turns a Linux kernel into a hypervisor using a kernel module. This module allows other guest operating systems to then run in user-space of the host Linux kernel. The KVM module in the kernel exposes the virtualised hardware through the /dev/kvm character device. The guest operating system interfaces to the KVM module using a modified QEMU process for PC hardware emulation. The KVM module introduces a new execution mode into the kernel. The KVM introduces a guest mode, along with vanilla kernels support of kernel mode and user mode. The guest mode is used to execute all non-I/O guest code, where normal user mode supports I/O for guests. [27]

2.2 Virtualised Infrastructure Management

Aligned with the vision of ETSI NFV ISG, the management of the underlying infrastructure that is being exploited by NFV (also known as NFVI-PoP) is managed by an entity called Virtualised Infrastructure Management (VIM) [24]. The VIM is required to manage both the IT (compute and hypervisor domains) and network resources by controlling the abstractions provided by the Hypervisor and Infrastructure network domains management (in the generic case this is performed by the data-centre network manager or in case of SDN by the Controller). Additionally, the VIM is required to collect infrastructure utilisation/performance data and to make this data available to the MANO framework. The specifics of how the metrics are provisioned and processed at both the VIM and Orchestrator layers can vary and will typically be implementation specific. To accomplish these goals the VIM needs the following capabilities:

- The Network Control capability in the VIM exploits the presence of SDN features to manage the infrastructure network domain within an NFVI-PoP;
- Hardware abstraction in the Compute domain for efficient management of resources; however, the abstraction process should ensure that platform specific information relevant to the performance of VNFs is available for resource mapping decisions;
- Virtual resource management in the hypervisor domain to provide appropriate control and management of VMs;
- Strong integration between the three sub-domains above through appropriate interfaces;
- Expose well-defined northbound interfaces to provide infrastructure related data to the MANO and to receive management and control requests.

This section attempts an identification of the possible technologies that support partially or fully the aforementioned functionalities and might by considered as possible targeted VIMs for the SONATA infrastructure adaptors. Although currently most deployments of NFVI-PoPs are based on Openstack deployments, constituting the choice of VIM almost trivial, efforts in the area differentiate by integrating other technologies and implementations in order to support more features in an attempt to support refined networking functionalities and service chaining.

2.2.1 OpenStack

2.2.1.1 OpenStack and NFV

OpenStack as an open source project has "greatly simplified the path to virtualisation for many" (see [28]). The practical evidence for the former abstract statement is that ETSI and OPNFV have defined specifications and released reference platforms for NFV that select OpenStack as the Virtualisation Infrastructure Manager. Additionally, OpenStack is the dominant choice for additional management and orchestration functions. NFV on OpenStack offers an agile, scalable, and rapidly maturing platform with compelling technical and business benefits for telecommunications providers and large enterprises (see [33]). Examples for such benefits:

- Standardized interfaces between NFV elements and infrastructure
- Resource pools cover all network segments
- Network/element deployment automation, roll-out efficiency
- Pluggable architecture with documented APIs, UIs, shared services, operations, automation
- All popular open source and commercial network plug-ins and drivers
- Outstanding global community contributing to rapid pace of innovation, working on unique
- NFV requirements from users, related communities and standards developing organisations, NFV features in every release since 2013
- Proven telecom and enterprise implementations: AT&T, China Mobile, SK Telecom, Ericsson, Deutsche Telekom, Comcast, Bloomberg, and more

2.2.1.2 Relevant OpenStack Projects

The primary OpenStack projects (also considered as part of the VIM) are:

- Nova - OpenStack compute - manages virtual or bare metal servers [10]
- Magnum - uses Nova instances to run application containers [10]
- Cinder - OpenStack Block Storage - manages virtual storage, supporting practically any storage back end [6]
- Neutron - OpenStack Networking - provides virtual networking [9]

Other core OpenStack projects:

- Glance - image management service [7]
- Ceilometer - data collecting service [5]
- Keystone - identity manager [WikiBibliography#openstackKeystone]]
- Heat - the main project in the OpenStack Orchestration program, implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files

OpenStack projects related to advanced features required for NFV support, such as scalability, high availability, resiliency, multi tenancy:

- Monasca - fault-tolerant monitoring-as-a-service solution that integrates with OpenStack. It uses a REST API for high-speed metrics processing and querying and has a streaming alarm engine and notification engine [8]

OpenStack addresses MANO through the Tacker project [11], which is building a generic VNF Manager (VNFM) and a NFV Orchestrator (NFVO), and it is based on ETSI MANO Architectural Framework

2.2.1.3 OpenStack relevance for SONATA

Taking into account the capabilities mentioned above, SONATA has chosen OpenStack Controller as the VIM on which it will perform its first year demonstration. Openstack distribution will support the SONATA PoPs for Y1 and till the end of the project.

2.2.2 OPNFV

OPNFV [19] is an open source project founded and hosted by the Linux Foundation, and composed of Telecom Service Providers and vendors. It aims at establishing a carrier-grade, integrated, open source reference platform to advance the evolution of NFV and to ensure consistency, performance and inter-operability among multiple open source components. OPNFV plans to validate existing standard specifications, contribute improvements to relevant upstream open source projects, and develop necessary new functionality both within OPNFV and upstream projects. In particular, it is focused on building NFV Infrastructure (NFVI) and Virtualised Infrastructure Management (VIM) by integrating components from upstream projects such as OpenDaylight, OpenStack, CEPH Storage, KVM, Open vSwitch, and Linux. These components, along with APIs to other NFV elements, compromise the needed infrastructure required for VNF and MANO components. Concentrating on these components, OPNFV aims to enhance NFV services by increasing performance and power efficiency, improving reliability, availability and service ability, and delivering comprehensive platform instrumentation. The figure below (Figure 2.3) highlights the project collaboration and positioning in the NFV scheme.

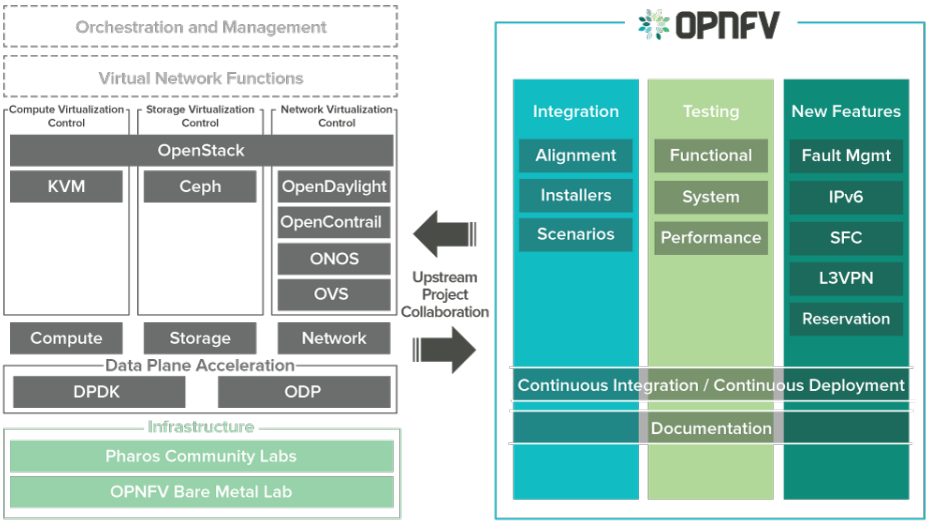

Figure 2.3: OpenNFV Project Collaboration

The newest release is named Brahmaputra and the users are able to deploy their VNFs using some MANO solution. The target software platform is integrated from a set of other open source components, of which the biggest ones are OpenStack and SDN controllers. Besides the target software platform, OPNFV provides a set of tools that helps the user deploy this target software platform on a set of servers. Brahmaputra implements a rigorous upstream integration process that has pulled the latest code from partner communities and projects that have contributed new capabilities, specifications, and community resources to the platform. It enables continuous integration, automated deployment and testing of components from upstream projects. Developers can deploy their own VNFs and test their functionality, and it allows the automatic continuous integration of specific components. OPNFV's automated tool-chain gives to ability to perform continuous automatic builds and verification. The Brahmaputra release provides carrier grade feature enhancements that include improved IPv6 support, fault detection and recovery capabilities (via work in OpenStack Neutron and Ceilometer), service function chaining capabilities, VPN instantiation and configuration, resource reservation and performance/throughput enhancements such as data plane acceleration and NFV-focused enhancements to OVS and KVM.

2.2.2.1 OPNFV Relevance to SONATA

For SONATA the Brahmaputra release of OPNFV is considered as the one running on the Data Center where the Service Platform and the SDK are running. This release will support the Y1 demos and will be updated according to the OPNFV project roadmap, when the new version is released. For the implementation of the various PoP a mix of deployments will be supported as it is presented further in the document. Some of the PoPs will use OPNFV (when the number of nodes is more than 3) and some will be deployed with vanilla Openstack distribution.

2.2.3 OpenMANO

OpenMANO [30] is an open source project that aims to provide a practical implementation of the reference architecture for NFV management and orchestration proposed by ETSI NFV ISG. The OpenMANO framework consists of three major components: openvim, openmano, and openmano-gui (see Figure 2.4) all available under Apache 2.0 license. The first component focuses on building a virtual infrastructure manager (VIM) that is optimized for VNF high and predictable performance. Although it is comparable to other VIMs, like OpenStack, it includes control over SDN with plugins (floodlight, OpenDaylight) aiming at high performance dataplane connectivity. It offers a CLI tool and a northbound API used by the orchestration component OpenMANO to allocate resources from the underlying infrastructure; this includes the creation, deletion and management of images, flavours, instances and networks. OpenVIM provides a lightweight design that does not require additional agents to be installed on the managed compute nodes [30].

The orchestration component itself can either be controlled by a web-based interface (openmano-gui) or by a command line interface (CLI) through its northbound API. OpenMANO's orchestrator is able to manage entire service chains that are called network scenarios and correspond to ETSI NFV's network services at once. These network scenarios consist of several interconnected VNFs and are specified by the service developer with easy YAML/JSON descriptors. It offers a basic life-cycle of VNF or scenarios (define/start/stop/undefine). This goes beyond what simple cloud management solutions, like OpenStack, can handle. The easy to install framework includes both, catalogues for predefined VNFs and entire network services including support to express EPA (Enhanced Platform Awareness) requirements. OpenMANO does not provide interfaces for the integration of service development tools, like feedback channels for detailed monitoring data to be

accessed by service developers. This limits the current system functionalities to orchestration and management tasks only.

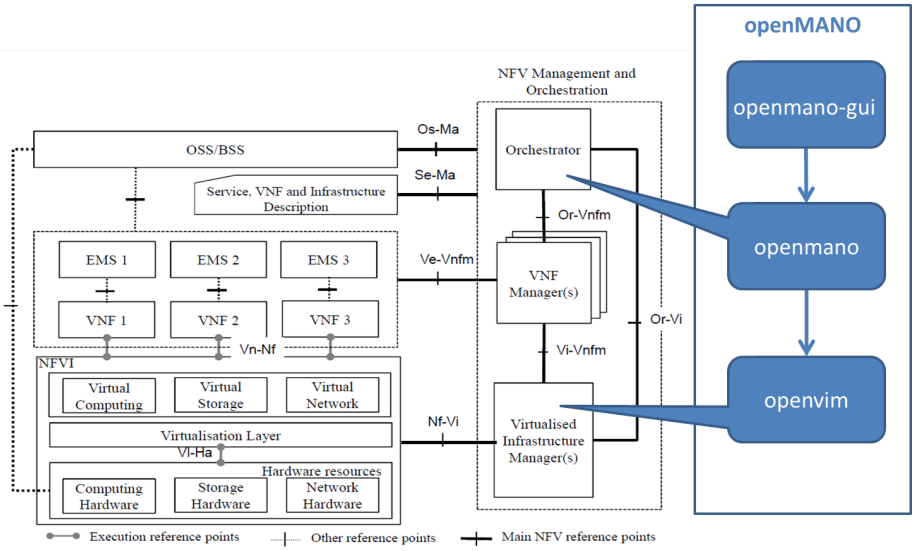

Figure 2.4: OpenMANO focus in context of ETSI MANO framework

2.2.4 Open Source MANO (OSM)

More recently, a new project namely Open Source MANO (OSM) [31] was announced. It is focused on delivering an Open Source NFV Management and Orchestration software stack for production NFV networks. The project launched the first release (Release 0) of the software in May 2016. The OSM project consists of OpenMANO project, Canonical's Juju Charms and Rift.io orchestrator, as shown in Figure 2.5. The OSM project aims to position itself as an evolution of the ETSI MANO framework towards an open source code which can be used to implement an ecosystem driven by NFV and NS development for different use cases. The OSM MANO group includes industries and research institutions [21].

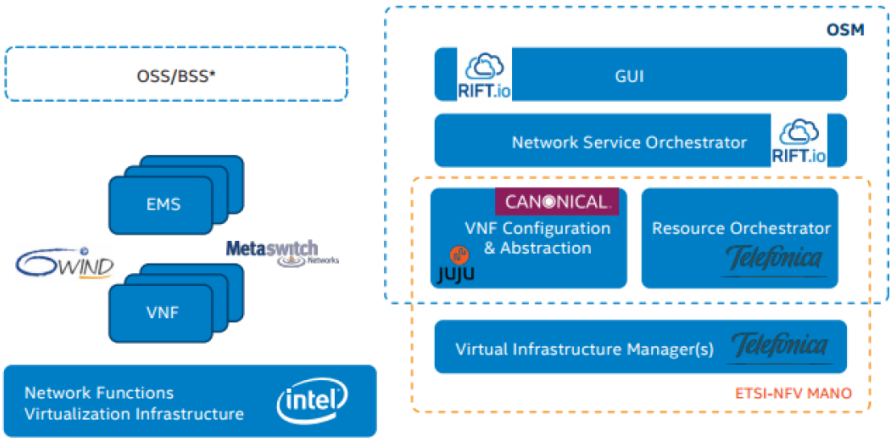

Figure 2.5: OSM architecture overview

The Virtual Infrastructure Manager(s) layer is out of the scope of the OSM project, as depicted

in Figure 2.5. However, OSM was demonstrated at MWC'16 using OpenVIM (from Open MANO project) and OpenStack Kilo. The OSM architecture for the demonstration in MWC'16 is shown in Figure 2.6. It shows the direct interfacing of OpenMANO with the OpenStack controller and OpenVIM controller separately [21]. This approach makes OSM very much VIM dependent and requires substantial effort to enable it for any other VIM. Since VIM layer is out of the scope of the OSM project, perhaps it will be beneficial to make it VIM agnostic with the help of a VIM abstraction layer.

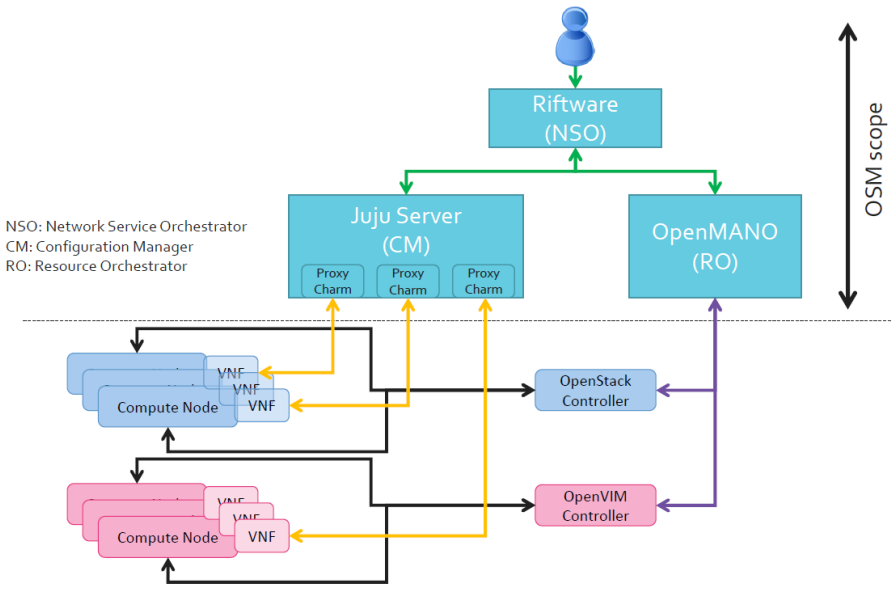

Figure 2.6: OSM architecture MWC 2016

2.2.5 VMWare vCloud

The VMware solution for NFV is the product vCloud NFV, an ETSI compliant platform that combines many VMware technologies, including vSphere for running virtual machines, VSAN software defined storage and NSX for software defined networking. On top of these core components, vCloud is designed to perform management and orchestration (MANO) operations. The vCloud Director is responsible for resource management along with vRealize Operations Manager for performance monitoring and analytics. vCloud NFV also includes VMware Integrated Openstack, a custom deployed Openstack, based on the Kilo release, as a management alternative to vCloud Director. Figure 2.7 shows the vCloud NFV platform.

2.2.5.1 NFVI Design

The NFV Infrastructure is implemented with the components ESXi, Virtual SAN (VSAN), and NSX for vSphere. These are the components on which VNFs are deployed and executed. ESXi is the hypervisor for running and managing virtual machines and VNF instances. VSAN is the storage solution, which is built into ESXi hypervisor, for storing virtual machines and VNF running instances. Finally, NSX for vSphere handles the network virtualisation, providing an overlay to support horizontal scaling of VNFs using logical switches and routers.

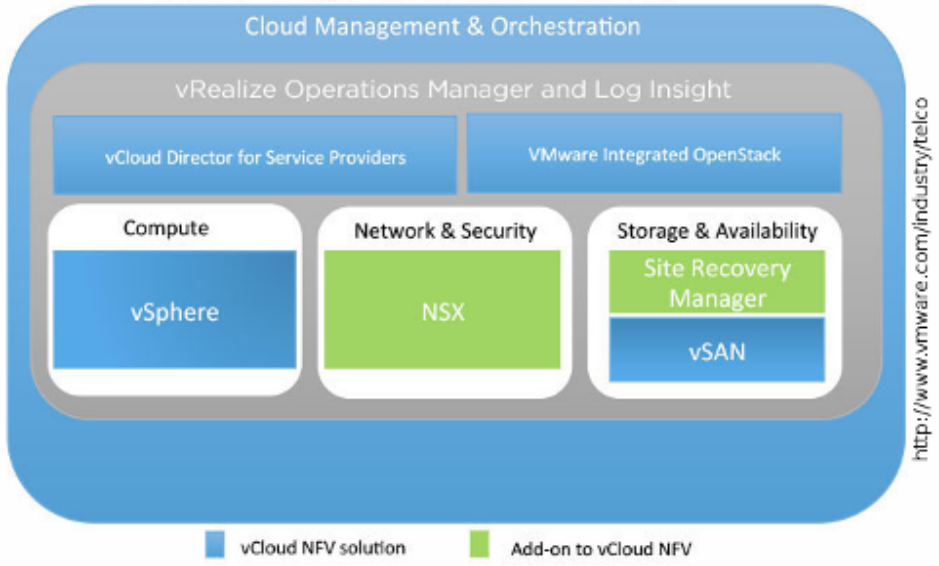

Figure 2.7: VMware vCloud Platform

2.2.5.2 VIM Design

Regarding the MANO working domain, the VIM is responsible for controlling and managing the NFVI resources and expose the northbound interface to VNF managers and the NVF Orchestrator. In vCloud NFV, the VIM is composed by the modules vCloud Director, vCenter Server and NSX for vSphere Manager. The following paragraphs will provide a brief description of the mentioned modules.

The vCenter Server is a central platform for managing and configuring the ESXi hypervisors. Typically, two vCenter Server instances are deployed per site, one to manage the cluster (VIM) and the other to manage the NFVI resources. A centralised point of administration of the VIM and the NFVI is then provided through vSphere Web Client.

The NSX for vSphere Manager provides a centralised management plane for NSX for vSphere. In other words, it provides further abstraction to the NVFI networking, enabling the inclusion of logical routers, gateways and controller clusters.

The vCloud Director provides an abstract management of multi-tenant clouds by pooling virtual infrastructure resources into virtual datacenters (vDC) (Figure 2.8). Virtual datacenters can be managed through web-based interfaces using a catalogue-based paradigm. vCloud Director relies on vSphere resources to run and store virtual machines. These resources are presented to the developer in the form of cloud resources, which is an abstraction of the underlying vSphere resources. They provide compute, memory, virtual machines and vApps. vApps are templates that specify parameters and operational details of one or more virtual machines.

2.2.5.3 NFVI Operations Management

The NFVI Operations Management functional block is responsible for the performance management of NFV infrastructures. The vRealize Operations Manager is the main module of this function block, providing management of NFVI and VIM components. The vRealize Log Insight is responsible for performance analysis with machine learning intelligent grouping.

The vRealize Operations Manager provides an integrated approach to performance, health and capacity analytics across the NFVI. This information can be exposed to OSS/BSS or integrated with

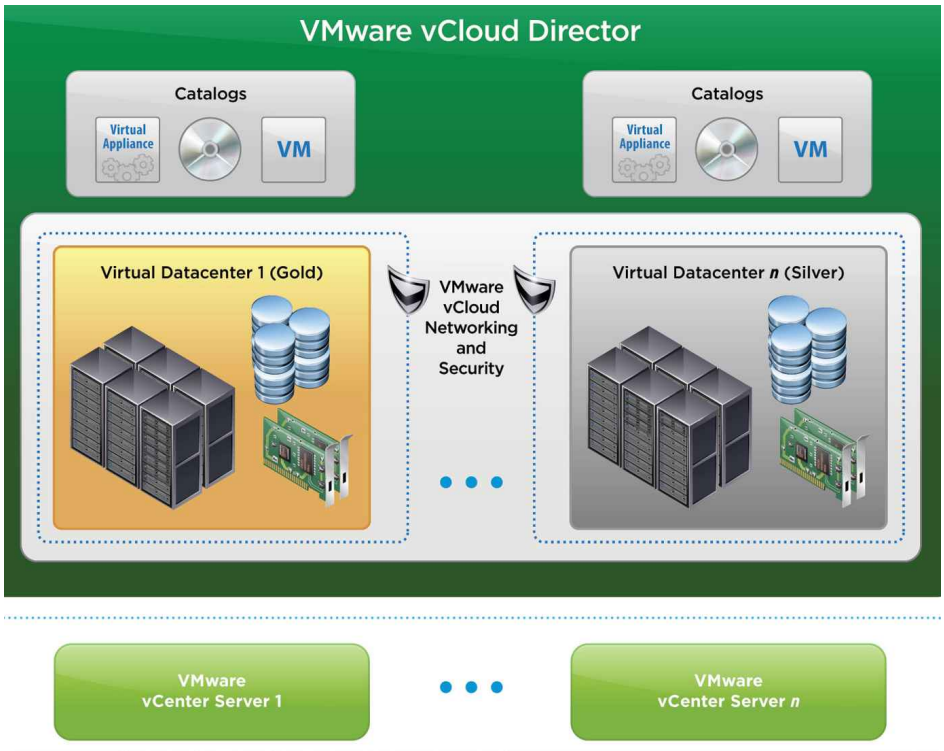

Figure 2.8: VMware vCloud Platform

MANO. Moreover, it is possible to create customised packs, bundles of relevant filtered information, and exposed them in an API to be consumed by an arbitrary external system. However, vRealize Operations Manager does not provide information about VNF instances or internal VNF KPIs. This information is handled only by the VNF Manager.

The vRealize Log Insight provides real time log analytics for hypervisors, hardware and other NVFI components. They can also be consumed by OSS/BSS or integrated with MANO. Generally, this module serves as a data source to vRealize Operations Manager.

The remaining three modules are dedicated to the protection of data. The functionalities, such as automation in recovering processes for disasters, data replication solutions and data backup, are handled by the modules Site Recovery Manager, vSphere Replication and vSphere Data Protection.

2.2.5.4 VMware Integrated Openstack

As an alternative to vCloud Director, the VMware Integrated Openstack distribution is designed to be deployed on top of an existing VMware infrastructure (Figure 2.9). The main goal of Integrated Openstack is providing developers with the open Openstack APIs and services while using the VMware infrastructure. Using the vSphere client, Openstack can be instantly and easily deployed within a cluster, therefore eliminating the complexity involved in Openstack deployment.

While open source VIMs, such as Openstack that support multiple variants of hypervisors and containers (KVM, VMware ESXi, Microsoft Hyper-V, Xen, Docker and LXC) vCloud only supports the VMware ESXi hypervisor. Similarly for storage, Openstack can integrate multiple storage solutions, such as Cinder block volumes, Ceph, GlusterFS, etc. whereas vCloud relies in VMware VMFS. In summary, vCloud NFV focuses in delivering a high availability virtualisation platform for enterprises and large service providers.

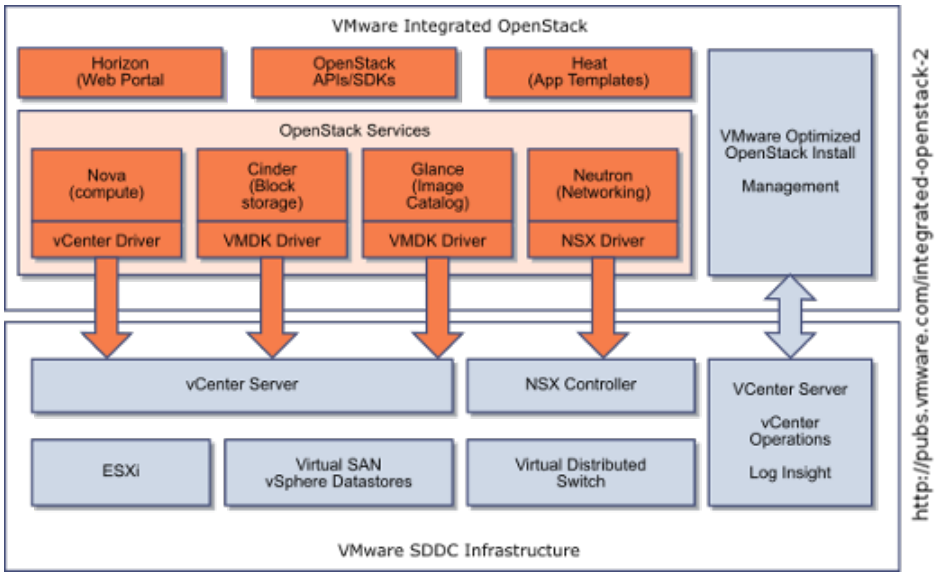

Figure 2.9: VMware Integrated Openstack

2.2.6 Project efforts

Recently a lot of EU funded projects have been focusing, in variable detail, to specify and implement infrastructures that are supporting virtualisation of IT resources and data-centre networking. Overall the activity of continuous market watch is undertaken by Task 2.4. This section attempts a summary of the most relevant efforts with a narrow scope in VIM technologies or available integrations made available by EU funded projects.

2.2.6.1 Unify

As shown in Figure 2.10, the UNIFY architecture is multi-layered with the infrastructure layer forming the base and lowest layer. The Infrastructure Layer encompasses all networking, compute and storage resources. By exploiting suitable virtualisation technologies, this layer supports the creation of virtual instances (networking, compute and storage) out of the physical resources. Primarily, four domains of physical resources are considered: Universal Node (general purpose server hardware including local UNIFY management and orchestration interfaces), SDN enabled network nodes (like OpenFlow switches), Data Centres (like controlled by OpenStack), legacy network nodes or network appliances.

Orchestration Layer (OL) comprising of two major functional components:

- *' Resource Orchestrator (RO)'* comprising of virtualisers, policy enforcement and resources orchestration between virtualisers and the underlying resources.

- *' Controller Adapter (CA)'* comprising domain-wide resource abstraction functions and virtualisation for different resource types, technologies, vendors or even administrations. The CA maintains the domain global view of resources and capabilities and presents its virtualisation to the RO.

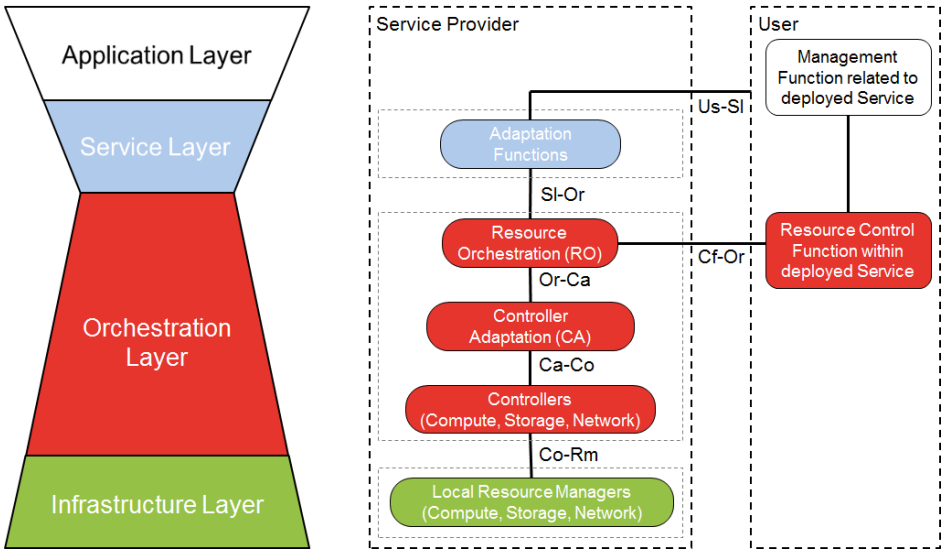

Figure 2.10: Unify Architecture Layers

Infrastructure Layer (IL) comprising of resources, local resource agents and/or controllers:

- Controllers comprising virtualisation functions corresponding to a single domain. For example: an SDN Controller for a transport network domain; a compute controller for a Data Center (DC).
- Agents and Physical Resources comprising all possible resource options (compute, storage and network) together with their corresponding local agents, e.g., OpenFlow switch agent, Open Stack Nova compute interface, etc.

The concept of infrastructure virtualisation and abstraction is crucial. Any provider of infrastructure in the UNIFY architecture can be augmented by a virtualiser entity. Virtualisers are responsible for the allocation of abstract resources and capabilities to particular consumers and for policy enforcements. This virtualised views should be vendor, technology and domain agnostic and contain the resources at compute, storage and network abstraction and capabilities.

Big Switch with Big Software (BiS-BiS) is virtualisation of a Forwarding Element (FE) connected with a Compute Node (CN), which is capable of running Network Functions (NFs) and connecting them to the FE. The concept combines both networking and compute resource abstractions (Figure 2.11).

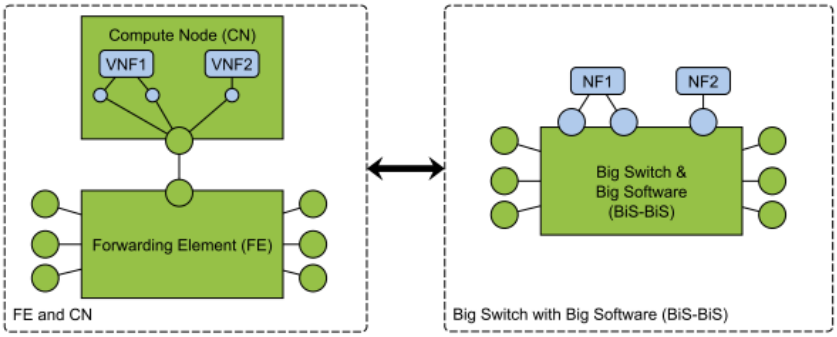

Figure 2.11: Unify Big Switch with Big Software

Service Access Point (SAP): a service provider's physical or logical port, which represents customers' point of presence, access to internal services or exchange points to other providers. SAP definitions are included into virtualisation. The description of the SAPs is part of the contract between the provider and the consumer of the virtualization.

As a result, UNIFY infrastructure providers can (recursively) abstract and virtualise the infrastructure they offer to orchestration components using BiS-BiS abstractions. The interface that is exposed by the infrastructure provider is following the YANG model which is presented in [36].

2.2.6.2 T-NOVA IVM Layer

T-NOVA's Infrastructure Virtualisation and Management (IVM) layer provides the requisite hosting and execution environment for VNFs. The IVM incorporates a number of key concepts that influence the associated requirements and architecture for the layer [18]. Firstly the IVM supports separation between control and data planes and network programmability. The T-NOVA architecture leverages SDN for designing, dimensioning and optimising control and data-plane operations separately, allowing capabilities from the underlying hardware to be exposed independently. Secondly the IVM is based around the use of clusters of Standard High Volume (SHV) computing nodes in cloud computing configurations to support instantiation of software components in the form of VMs for NFV support, offering resource isolation, optimisation and elasticity. The proposed IVM layer architecture is illustrated in the figure below (Figure 2.12).

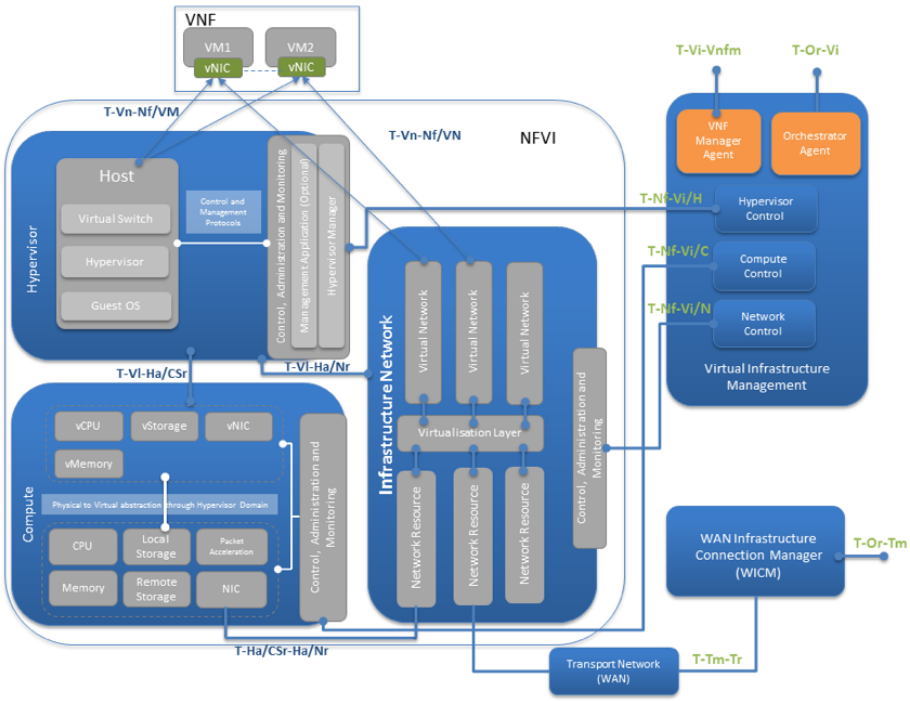

Figure 2.12: T-NOVA IVM Layer

Two are the main components included in the T-NOVA IVM layer. The first is the Virtualised Infrastructure Manager (VIM) responsible for the NFVI-PoP resources and the second is the WAN Infrastructure and Connectivity Manager (WICM) [1] for the WAN interconnecting PoPs across the network footprint. Currently the T-NOVA VIM is based on OpenStack services (e.g Nova and Neutron), however the networking and service function chaining is achieved via SDN control through an SDK for SDN developed in the frame of the project [39].

2.3 Automated NFVI deployment frameworks

It is generally understood that configuration and deployment of infrastructure and management components is cumbersome and time consuming. Various solutions are available for providing a variety of automation with respect to the complexity of the infrastructure to be deployed. In SONATA case the automatic deployment is used for the automatic deployment of the SONATA Service Platform as well as Integration and Qualification infrastructure. This section references the most relevant frameworks for automated infrastructure deployment. These frameworks are extensively used by cloud operators and IT engineers.

2.3.1 Mirantis Fuel

Fuel is an open-source tool to facilitate the deployment of multiple OpenStack environments, as well as their verification and management, using a web-based application UI. Fuel provides several key features to enable an effortless deployment, such as hardware discovery, network and disk partitioning in UI, non-HA and HA configurations, health verification and test validation of environments, real-time logs in UI, etc. Currently, Fuel only supports CentOS and Ubuntu operating systems and a explicit plan to extend its support could not be found.

The Fuel architecture comprises several independent components, described as follows:

- **UI** is the main Fuel interface, enabling the user to configure hardware, virtual machines, assign OpenStack modules to machines, configure networking, etc. Ultimately it allows the deployment and management of multiple OpenStack environments.
- **Nailgun** is the coordinator component of Fuel. It provides a REST API with which the UI and the CLI interact. Nailgun is responsible to manage disk volumes configuration, networks configuration, and all the environment required data for a successful deployment. It transmits messages through an AMQP service which will be consumed by other modules and trigger deployment and management actions.
- **Astute** represents Nailgun's workers which will consume and analyse the gathered messages to infer the required type of action. For the required action, Astute will launch a specific service, which includes Cobbler, Puppet, shell scripts, etc.
- **Cobbler** provides root services and the base installation of the OS on OpenStack nodes. Cobbler is able to control DHCP and TFTP services to enable network booting in nodes and start the OS installation.
- **Puppet** is the service that handles the deployment of OpenStack environments. Its operation is based on manifests for deployment and on-the-fly management of the OpenStack infrastructure.
- **Mcollective agents** handle small maintenance tasks, such as clearing of hard drives, probing network connectivity, etc. This service is usually triggered by Puppet or directly invoked by Astute.
- **OSTF (OpenStack Testing Framework)** was designed to perform post-deployment verifications of OpenStack. It aims at running non-destructive tests in environments in order to assess the proper operation of services.

A typical Mirantis Openstack platform topology deployed by Fuel is shown in Figure 2.13.
The typical networking topology for the deployment of Openstack using Fuel is as follows:

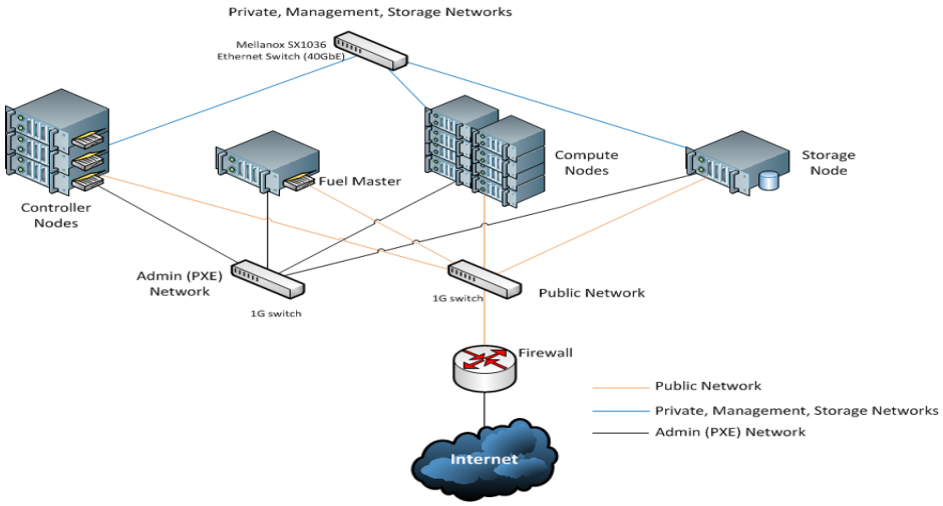

Figure 2.13: Mirantis Fuel topology

- Public network - the one that provides the interconnection of the whole infrastructure with the internet and also supplies the floating IP segments to the running VMs in the Openstack environment.
- Private, Management and Storage network(s) - these networks may be seen as a single network segment (for small installations) or segmented in three isolated network segments. This segment is used for the private openstack networking needs (communication of the compute nodes with the control and network node). Management and monitoring can be located on a different network and finally storage is required in case iSCSI or NAS storage solutions are employed.
- Admin (PXE) network - this segment is actually the admin network used for booting and controlling the deployment process from Mirantis Jumphost. PXE will be used for bootstrapping the machines and feeding the appropriate distribution packages.

2.3.2 Foreman

Foreman is an open source project that helps system administrators manage servers throughout their lifecycle, from provisioning and configuration to orchestration and monitoring. With the use of Puppet, Chef, Salt, Ansible and Foreman's smart proxy architecture, it can automate repetitive tasks, deploy applications, and manage change. Foreman provides comprehensive, interaction facilities including a web front-end, CLI and RESTful API which enables to build higher level business logic on top of a solid foundation. Foreman supports a variety of operating systems as Fedora, Ubuntu, Debian.

The Foreman architecture is illustrated in Figure 2.14. A Foreman installation will always contain a central Foreman instance that is responsible for providing the Web based GUI, node configurations, initial host configuration files, etc. The smart proxy manages remote services and is generally installed with all Foreman installations to manage TFTP, DHCP, DNS, Puppet, Puppet CA, Salt, and Chef. A Smart-Proxy is located on or near a machine that performs a specific function and helps foreman orchestrate the process of commissioning a new host. Placing the proxy on or near to the actual service will also help reduce latencies in large distributed organisations.

Foreman can provision on bare metal as well as in a variety of cloud providers including OpenStack, VMWare, Rackstack, EC2, and there is the ability to do the following:

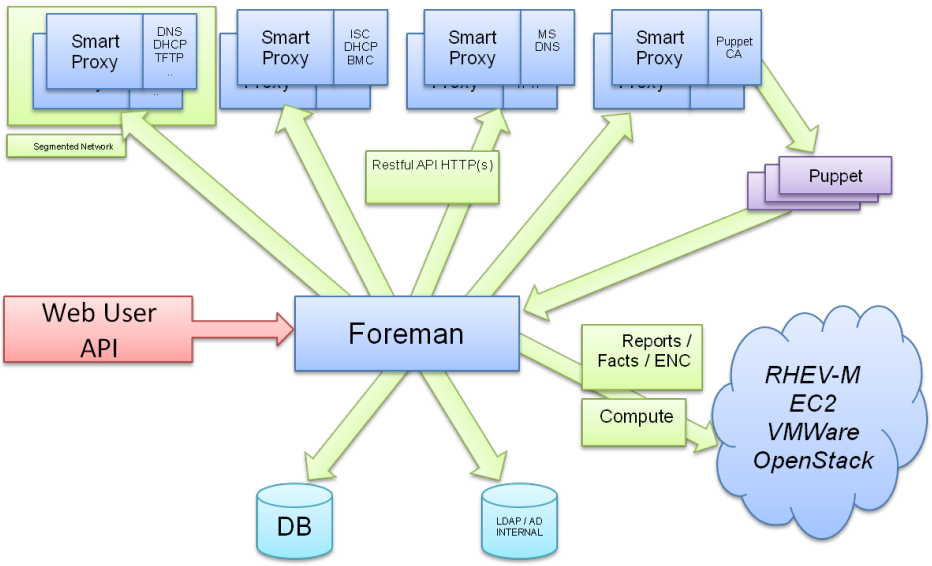

Figure 2.14: Foreman Architecture

- Discover, provision and upgrade your entire bare-metal infrastructure
- Create and manage instances across private and public clouds
- Group your hosts and manage them in bulk, regardless of location
- Review historical changes for auditing or troubleshooting
- Extend as needed via a robust plugin architecture
- Automatically build images (on each platform) per system definition to optimise deployment

2.3.3 Rackspace Private Cloud

Rackspace Private Cloud (RPC) uses a combination of Ansible and Linux Containers (LXC) to install and manage an Openstack platform.

- The Ansible provides the automation platform in order to simplify the deployment of infrastructure and applications by SSH (other methods also supported). The orchestration is based on playbooks, a YAML language. The machine where the playbooks run is referred to as the deployment host (typically, a VM), while the hosts where the playbooks install the RPC are referred to as target hosts
- The Linux Containers (LXC) provide virtualisation at the operating system level, improving the concept of "ch-root" environment, which isolates and filesystems resources to a particular group of processes, without the overhead and complexity of virtual machines. This group of processes accesses the same kernel, devices and filesystem of the underlying host

The use of kernel-level provides isolation through the use of cgroups (CPU, RAM, block IO, NET). Containers also provide advanced network configuration as:

- Bridge - providing layer 2 connectivity (like a switch) inside the host (physical, logical network interfaces and virtual)

- Namespaces - providing layer 3 connectivity (like a router) within a host (virtual interfaces, veth)

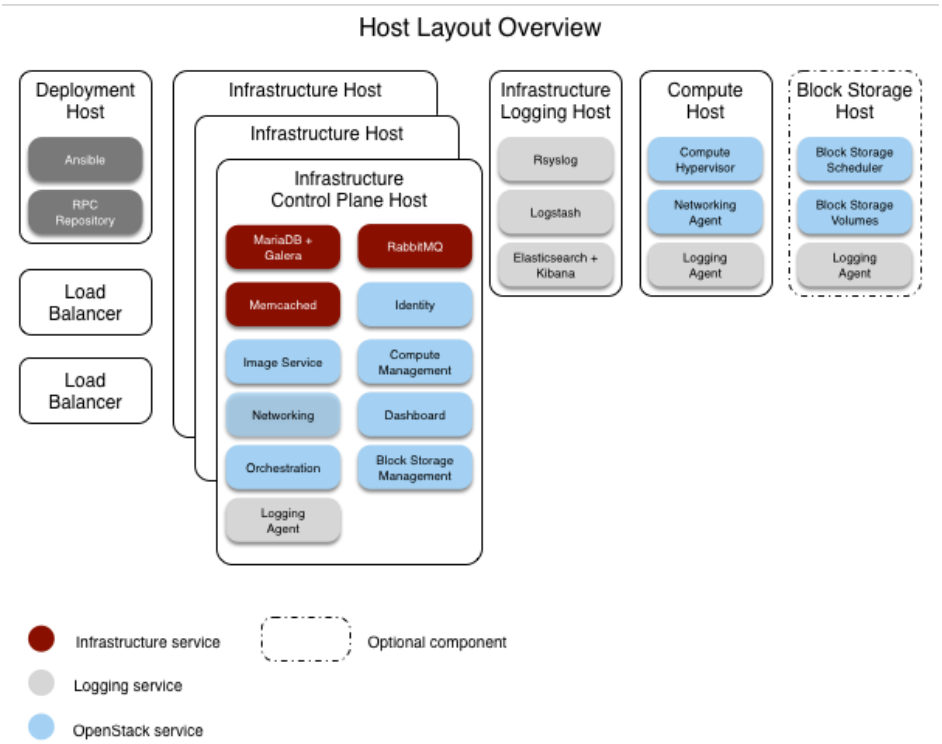

Figure 2.15: RPC environment overview

The application services automatically deployed by the RPC tool are:

- Infrastructure control, monitory and logging
  - galera (w/ MariaDB), rabbitMQ, memcached, rsyslog, logstash, elasticsearch w/ kibana
- Openstack services
  - Compute (nova-compute), Object Storage (swift), Block Storage (cinder), Networking (neutron), Dashboard (horizon), Identity (keystone), Image Service (glance), Orchestration (heat)

The RPC Reference Model consists of +5 machines as illustrated in Figure 2.15:

- 3 infrastructure nodes
- 1 logging node
- 1+ block storage nodes (to connect to a storage backend)
- 1+ block compute nodes (to run VMs and/or Containers)
- 1 (VM) deployment node

OpenStack Networking (Neutron) with the ML2 plug-in and Linux Bridge agents support the following features:

- Flat and provider VLAN networks
- Flat, VLAN, and VXLAN overlay (tenant) networks
- L3 agents for routing, NAT, and floating IP addresses

The network topology is illustrated in Figure 2.16

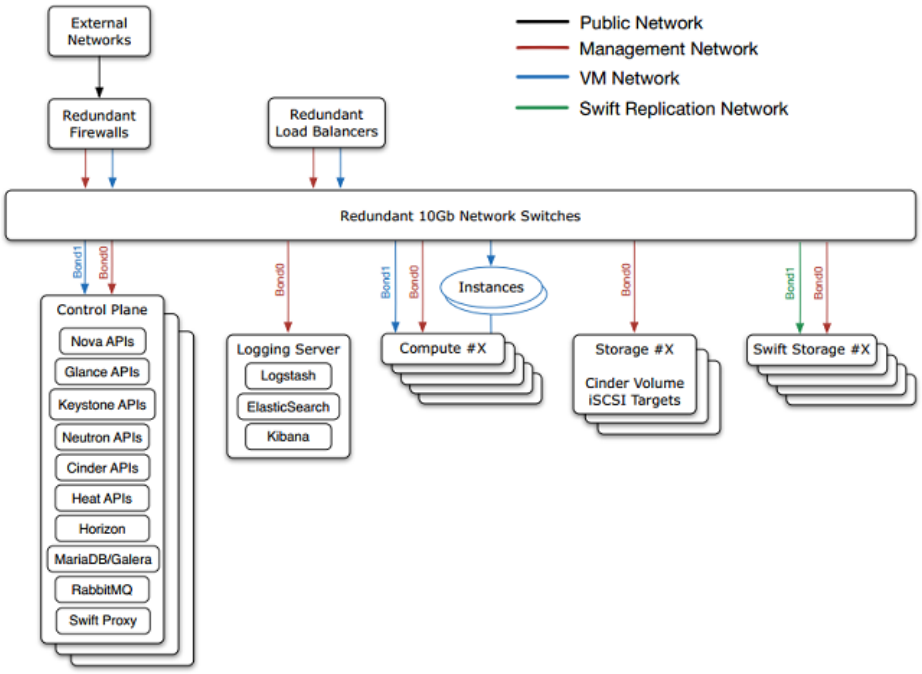

Figure 2.16: RPC Reference Architecture

The installation process follows the following steps:

1. the host deployment preparation
2. the preparation of target hosts
3. the host deployment configuration
4. the playbook running: foundation + infrastructure + openstack

The convenience of this solution is that you only have to change 2 files to deploy an Openstack platform i.e rpc_user_ config.yml and variables.yml.
NOTE: currently, RPC v.12 deploys Openstack Liberty release.

2.3.4 Canonical Juju

Juju is a modelling tool for cloud applications and services. It provides a framework to model, configure, deploy and manage applications in cloud environments. Models exist for many services in the so-called Juju store. There are two categories there:

- charms: to deploy applications
- bundles: to create services out of single applications

Juju can be controlled from a graphical user interface (GUI) and the command-line. It is available for different public and private cloud environments, including:

- Amazon Web Services
- Windows Azure
- Digital Ocean
- Google Compute Engine
- HP Public Cloud

In local mode, Juju supports Linux Containers (LXCs) and KVM. Using Juju with Vagrant is currently under development (and available as a beta).

Once a charm as been deployed, each created node is known as unit. Units are named after the application or bundle they run and include the sequence number of the created instance (i.e. 1 for the first instance, 2 for the second, etc.). Charms also define specific operations or 'actions'.

2.3.4.1 Inner structure of a Charm

Charms are defined in a text file that follows a YAML format. The file is called metadata.yaml and should be enough to define the charm. In addition, two directories are created, the mandatory /hooks for hooks to execute things and the optional /actions directory to define actions for user to invoke. More information on how to create charms is available at 1.

3 Infrastructure Flavours

SONATA Continuous Integration and Continuous Deployment (CI/CD) process specifies three stages (c.f [14]) as illustrated in Figure 3.1:

- *Development* - where the development of SONATA artefacts is performed continuously with the unitary testing and packaging
- *Integration* - where the packaged components will be deployed for integration testing
- *Qualification* - where the integrated and verified packaged components will be deployed for qualification tests

To serve these stages, SONATA provides three distinct infrastructure platforms:

- Integration Infrastructure (II)
- Qualification Infrastructure (QI)
- Demonstration Infrastructure (DI)

To close the CI/CD cycle, SONATA provides built-in plugins for multiple VIM connectivity at multiple sites.

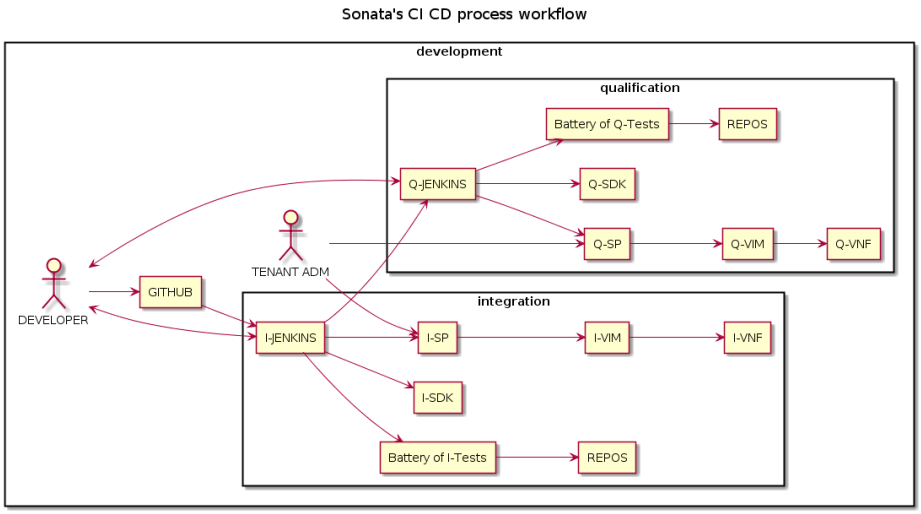

Figure 3.1: CI CD process workflow

The infrastructure resources available for Y1 are:

- **single-VIM**

  - Openstack based - For Y1, only Openstack will be supported. However for the second year a number of candidates is considered for additional wrapper implementation at the SONATA Infrastructure Adapter (e.g., OpenVIM/OSM, Apache CloudStack, OpenNebula, AWS, GCP, AZR).

- **multi-PoP**
  - the 1st NFVI-PoP at NCSRD in Athens, Greece
  - the 2nd NFVI-PoP at AlticeLabs in Aveiro, Portugal

In the multi-PoP scenario interconnection between the two sites will be established over a L3VPN (IPSEC based) overlay. Details on the physical and logical topology of the SONATA infrastructure and its assorted elements are discussed in the next sections.

The following sections describe the various infrastructure flavours and provide information on their deployment.

3.1 Hosting Infrastructure

SONATA assumes the existence of an underlying available datacenter infrastructure in order to be able to automatically deploy SONATA artefacts. Additionally it requires communication with VIM and WIM entities in order to be able to orchestrate the available resources on the managed infrastructures under their administration. This section describes the hosting infrastructure as it was laid out for the first year of the project.

The figure below (Figure 3.2) provides a physical view of the physical infrastructure hosting SONATA infrastructure flavours and PoPs. The infrastructure comprises of a main datacenter managed by Openstack and smaller installations that run on single-node all-in-one Openstack deployment for the realisation of additional PoPs. Currently the hosting infrastructure is located in NCSRD premises and a similar deployment (hosting more PoPs) is located in AlticeLabs (formerly known as PTIN) premises. This section discusses mostly the deployment used in NCSRD premises, similar description applies to Alticelabs premises too. The detailed hardware committed for the realisation of this infrastructure is described at the NCSRD Pilot section.

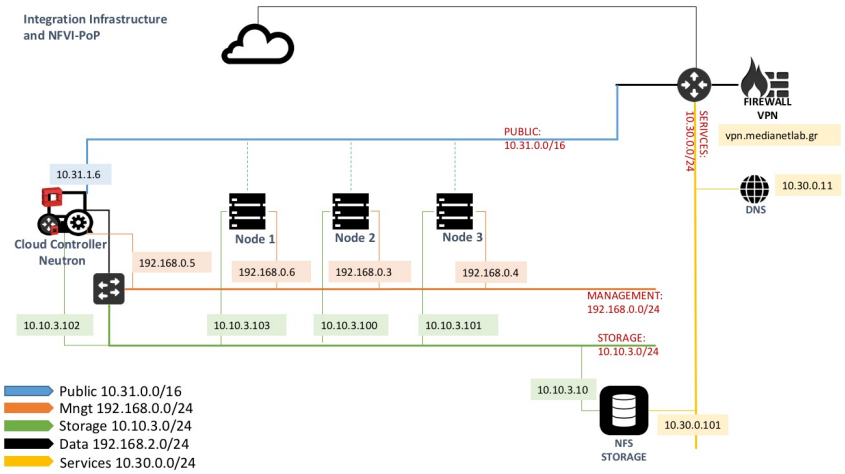

Figure 3.2: Integration Infrastructure and NFVI-PoP Physical deployment

3.1.1 Automated Infrastructure Deployment

The selected automatic deployment framework used for the implementation of the Hosting Infrastructure is based on OPNFV Brahmaputra. This release deploys the Liberty version of Openstack along with many supported Openstack projects (such as Ceilometer, Murano, CHEF, SWIFT, etc). This choice was made in order to have fast redeployment cycles and also be able to easily upgrade our infrastructure. Furthermore as the intention is to co-host the SONATA SP with the NFVI-PoP for some of the SONATA infrastructure flavours, the selection of OPNFV is fairly obvious.

In the following section a summary of the automatic deployment process is being presented. Detailed generic technical guidelines can be searched in the official OPNFV distribution site.

3.1.1.1 Network Provisioning

Before proceeding with the installation, the networking should pre-provisioned across the infrastructure elements (compute nodes, switched etc). The following physically separated networks are used,(see Figure 3.3):

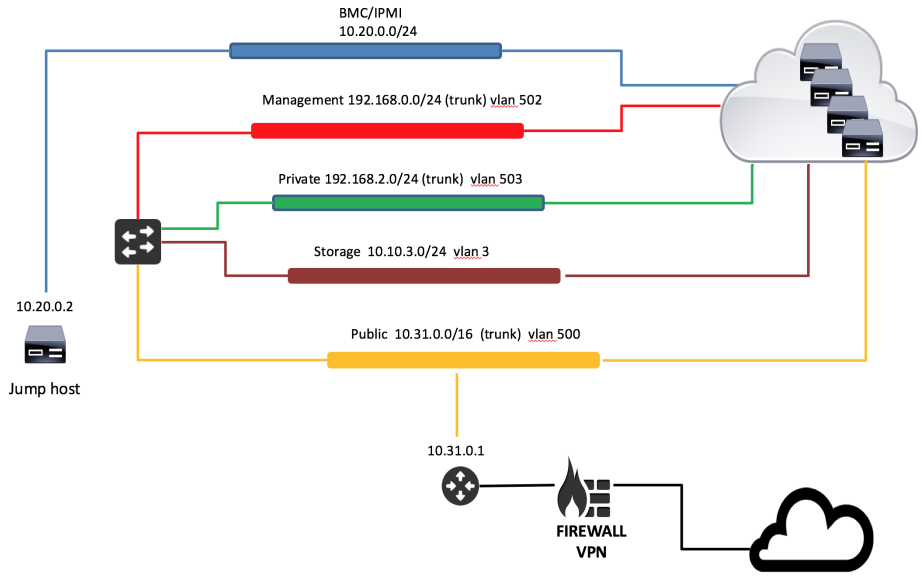

Figure 3.3: Networking topology deployment for Fuel

1. A physical network for administration and control
2. A physical network for tenant private and public networks
3. A physical network for storage
4. A physical network for Lights-Out Management (LOM)

The above networks are provisioned manually at the Top-of-the-rack (TOR) switches of the infrastructure. All the networks involved in the OPNFV infrastructure as well as the provider networks and the private tenant VLANs needs to be manually configured.

3.1.1.2 Mirantis FUEL configuration

As soon as the networks have been manually provisioned, the process of automatic deployment can start. Firstly, the ISO image of Brahmaputra Fuel is downloaded and then installed to our Fuel Master server (aka Jumphost). When this is done, PXE booting should be enabled in all the nodes we want FUEL Master to control. After that, the nodes that FUEL can control can be seen through the FUEL UI.

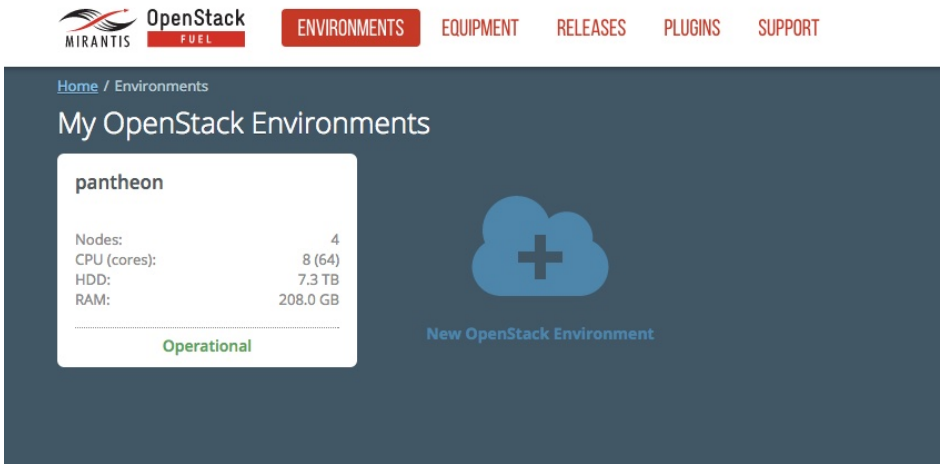

Figure 3.4: Mirantis WEB UI Home page

After these initial configurations, it is time to deploy OpenStack. As it can be seen in Figure 3.4 the OpenStack environment, in this setup, was named Pantheon and "Liberty Ubuntu 14.04" was selected along with "QEMU-KVM as hypervisor" option. After making a few more adjustments, the create button is pressed and it then the process begins. After the environment is created, a few options tabs can be seen when the environment is accessed. Main tabs are *Nodes* and *Networks.* The first one is to configure "roles" for each node (e.g. compute node or controller while the later is used to configure networking settings, for example IP segments, vlan Ids etc.

3.1.1.3 Deployment

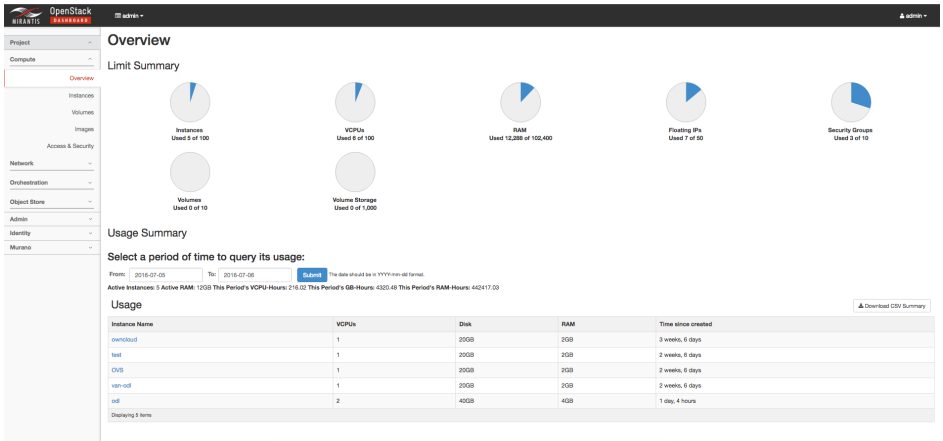

Figure 3.5: OpenStack Horizon dashboard

Finally after all the changes and configurations are made then it is ready to deploy the OPNFV environment. This is done by pressing "Ready to deploy?" button. When FUEL has finished all the proceedings, it is then possible to connect to the deployed OpenStack Environment as can be seen below in (Figure 3.5).

3.1.2 SONATA NFVI-PoP

The SONATA example PoP that is considered for the first year deployments is illustrated in Figure 3.6. According to the assumptions made for SONATA (as the project does not targets specifically the infrastructure), Figure 3.6 presents all the main components that comprise the NFVI-PoP which essentially is an execution environment that provides IT resources (computing, memory and storage) for the VNFs.

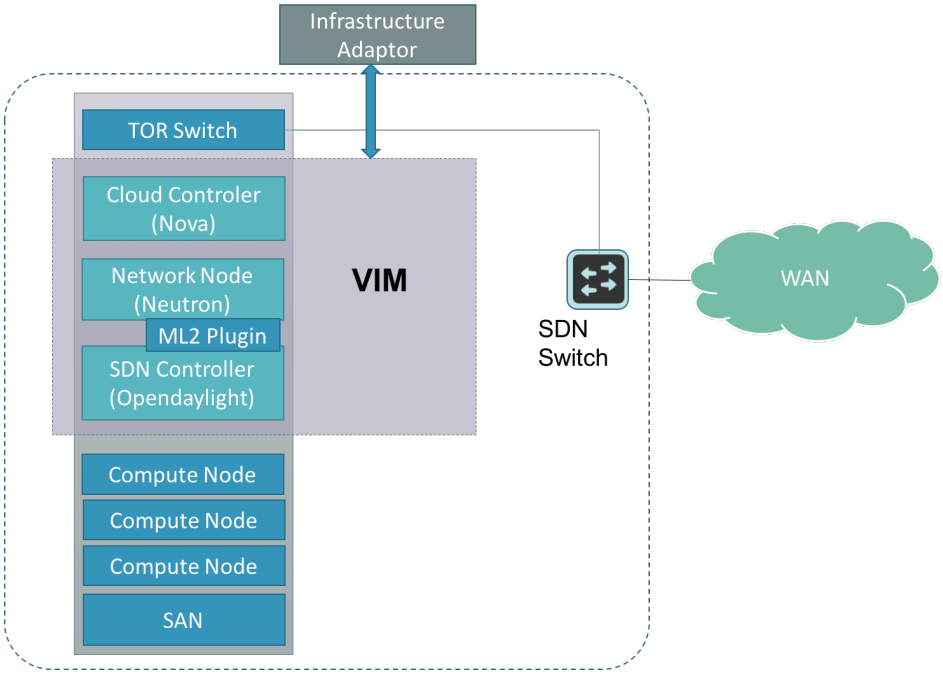

Figure 3.6: SONATA NFVI-PoP

The NFVI-PoP environment comprises:

- Compute Nodes (CNs) based on x86 architecture commodity hardware without particular platform capabilities
- a Cloud Controller node for the management and control of the available IT resources, based on Openstack platform. Currently Liberty release is running
- a Network Node (co-located with the controller node) running OpenStack Neutron service for managing the in-cloud networking created by OpenVirtualSwitch instance in each CN and also in the Network Node
- an SDN Controller, based on the recent version of OpenDayLight platform (Lithium), for the control of the virtualised network resources. The interaction and integration of the SDN controller with the OpenStack platform is achieved via the ML2 Plugin component provided

by Neutron service. It should be noted that currently tight integration of SDN with Neutron is only possible for older version of Openstack (i.e. Kilo). In this view, current PoP implementation lucks features that could automate the service function chaining in the PoP. Alternatively other mechanisms will be employed in the future release, unless the integration issues are solved

- a SAN storage serving the storage requirements of the instantiated services.

The anticipated instances of PoP in SONATA infrastructure will vary both in features and scale, servicing the purposes of the Pilots and the demonstration requirements.

3.2 Integration Infrastructure

The core element in the **Integration Infrastructure** is the Jenkins test tool and represents the beginning of the Continuous Integration cycle.

This page does a brief introduction to the requirements that justifies the need for an Integration Infrastructure and has the following structure:

- type of tests executed on the Integration environment
- the objects being tested
- the Integration test cycle

For a deeper view on Integration Tests, look at D5.1 and D5.2 documentation [17].

3.2.1 Type of Integration Tests

The kind of tests typical executed during Integration phase are:

- internal communication tests, ie, test the internal sub-system information flow
- inter-module communication, ie, test the information flow between modules
- test against VNF/NS already running at the existing VNFI, ie, test the monitoring data that comes from VNF monitoring probe

3.2.2 Integration Test goals

The object of test under Integration are the VNF/NS implemented by the Developer, so the Integration environment must instantiate resources to execute batteries of tests against those VNF/NS.

To accomplish the need to instantiate resources for testing, the Integration environment must be linked to a compatible VIM:

- for Y1 demos, the default VIM is Openstack
- for Y2, SONATA aims to support multi-VIM, so we should expect a connection to OSM, AWS or other common VIMs

NOTE: a connection to a local KVM hypervisor based on KVM tools should also be evaluated

3.2.3 Integration Test cycle

The SONATA Integration Test cycle has the following workflow:

1. after the execution of their own unitary and local tests, the Developer submits a "pull request" to the official SONATA Repository
2. this "pull request" triggers the CI process based on Jenkins
3. if Jenkins gets a successful build, then requests a Container to install the application for testing
4. if the batteries of tests successfully pass, then a Docker Image is generated and pushed to private SONATA Docker Registry

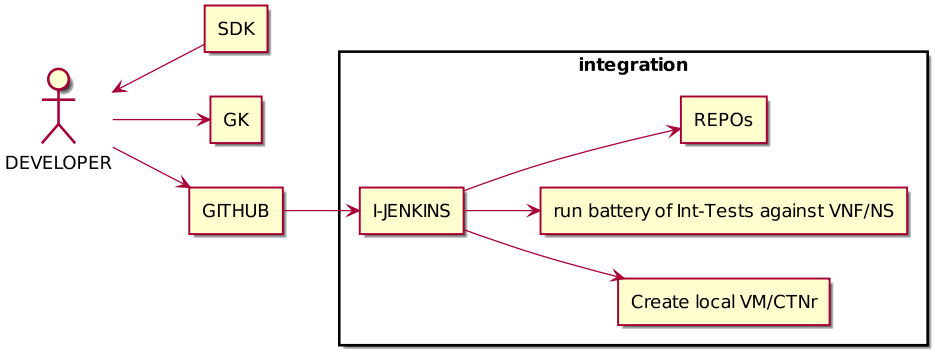

Figure 3.7: Integration Test cycle

The Integration Infrastructure is a platform to serve the developers, helping them to produce better code: it authenticates and authorises the users, giving access to the SDK, automatically generates builds and artefacts on the CICD engine from committed code, executing automated tests and producing stable versions to be handled by the next phase of the CICD process, ie, the Qualification phase.

The Integration infrastructure relies on a simplified version of the Service Platform (SP) optimised to run Continuous Integration tools, ie,

- allocation of higher resources to the CI module (ie, more computational power to Jenkins)
- allocation of lower resources to the GK, MANO and IFTA modules

For Y1, the stripped Integration Infrastructure for SP runs on 7 VMs containing the following components, as shown in the table:

Table 3.1: Integration Environment Infrastructure

VM	Function	SW tools	IP addr.	URLs	Flavour
#1	CICD engine	Jenkins	143.233.127.11	jenkins.sonata-nfv.eu	jenkins
#2	int-srv-1	SP, API, BSS, GUI	10.31.11.29	sp1.int.sonata-nfv.eu	m1.large
#3	int-srv-2	SP, API	10.31.11.33	sp2.int.sonata-nfv.eu	m1.large
#4	int-srv-3	SP	10.31.11.36	sp3.int.sonata-nfv.eu	m1.large
#5	Repos	Docker Registry	10.31.11.34	registry.sonata-nfv.eu	m1.large
#6	Monitory	MONIT	10.31.11.35	monitor.sonata-nfv.eu	m1.large
#7	Logging	LOGs	10.31.11.37	logs.sonata-nfv.eu	m1.xlarge

The computational resources allocated per flavour are shown in the table:

```
$ source openrc.int
$ openstack flavor list -c Name -c RAM -c Disk -c VCPUs
+--------------------------+-------+------+-------+
| Name                     |   RAM | Disk | VCPUs |
+--------------------------+-------+------+-------+
| graylog                  |  4096 |   30 |     1 |
| m1.tiny                  |   512 |    1 |     1 |
| m3.medium                |  4096 |  400 |     2 |
| m1.small                 |  2048 |   20 |     1 |
| m1.medium                |  4096 |   40 |     2 |
| m1.large                 |  8192 |   80 |     4 |
| m1.xlarge                | 16384 |  160 |     8 |
| m1.micro                 |    64 |    0 |     1 |
| m2.medium                |  4096 |  120 |     2 |
| jenkins-compilation-node |  2048 |   20 |     2 |
| jenkins                  |  4096 |  120 |     4 |
+--------------------------+-------+------+-------+
```

The available Integration Infrastructure (ie, a stripped version of the SP) provides the following SP services:

- **Gatekeeper** (GK)
  - The GK is the interface between Developers and Operators and the SONATA Service Platform (SP). It acts as a portal for authentication, licensing and packaging control, and metrics visualisation (KPIs)
- **CI/CD** engine
  - The Continuous Integration process relies on **Jenkins** tool and is integrated with Github
  - The Continuous Delivery process relies on Ansible playbooks for Configuration Management, Automatic Deployment and Infrastructure Management
- **Repositories**
  - The SONATA Repositories are built of Package repository, Image repository, Infrastructure repository and the Service Catalogue
- **Monitory** tools
  - to monitor Jenkins jobs
- **MANO** framework
- **VIM-Adapter**
  - The connector to a supported VNFI

There are also the following auxiliary components use by the Developer:

- SDK

  - The SONATA Software Development Kit (SDK) is the standard tool used by Developers to create compatible pieces of software to be processed by the SP. At the end of the CI/CD cycle, those pieces of software (VNFs and NSs) are packaged and are ready to deploy to a VNFI.

- Github
  - The source code version control system

The logical view of the Integration platform with CI/CD engine for Y1 looks like the picture:

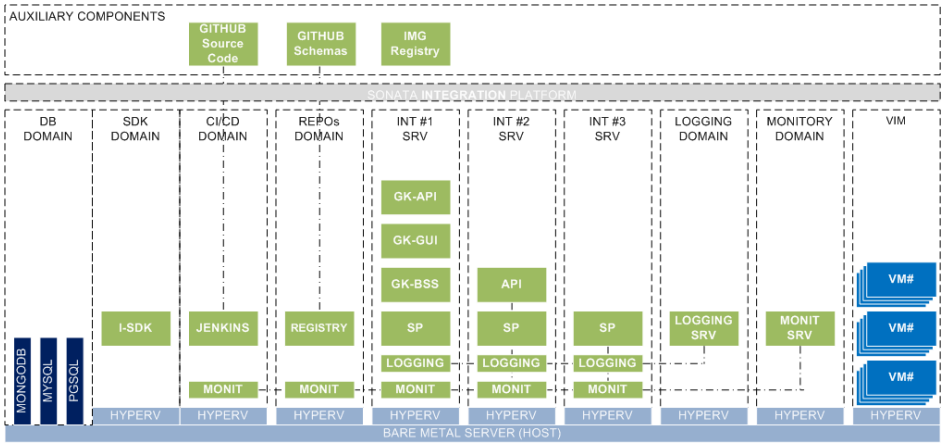

Figure 3.8: II logical view

3.2.4 Infrastructure Resources for the Integration SP

To deploy an all-in-one Integration environment for experimental purposes, the minimum requirements for a VM to run the SP Integration services are:

- 4 vCPU
- 16 GB vRAM
- 30 GB vDisk
- 500 GB Volume for Monitoring and Logging repository, used to store binary packages and images (this is convenient for backup and fault-tolerant matters), monitory and logging data.

However, to deploy a more definitive Integration platform in a production environment, the following infrastructure is advisable:

- Jenkins test tool - for Testing and Deployment purposes
  - 4 vCPU, 16G vRAM, 40G vDSK, 500GB MOUNTPOINT
- Integration Server #1 - for Testing and Deployment purposes
  - 4 vCPU, 16G vRAM, 40G vDSK
- Integration Server #2 - for Testing and Deployment purposes

– 4 vCPU, 16G vRAM, 40G vDSK,

- Monitory CI/CD - for monitoring Jenkins Jobs
  – 4 vCPU, 16G vRAM, 40G vDSK, 500GB MOUNTPOINT
- Logging CI/CD - for system logging Jenkins Jobs
  – 4 vCPU, 16G vRAM, 40G vDSK, 500GB MOUNTPOINT
- Docker Registry - for a private Docker Image Repository
  – 4 vCPU, 16G vRAM, 40G vDSK, 500GB MOUNTPOINT

Figure 3.9 shows the actual Integration environment for Y1:

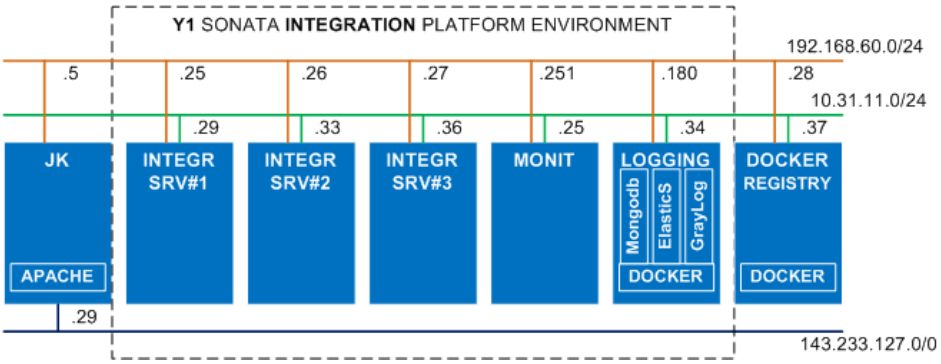

Figure 3.9: II network topology

3.2.5 Design and Specification

The Figure 3.10 below presents the Integration Infrastructure domains and components:

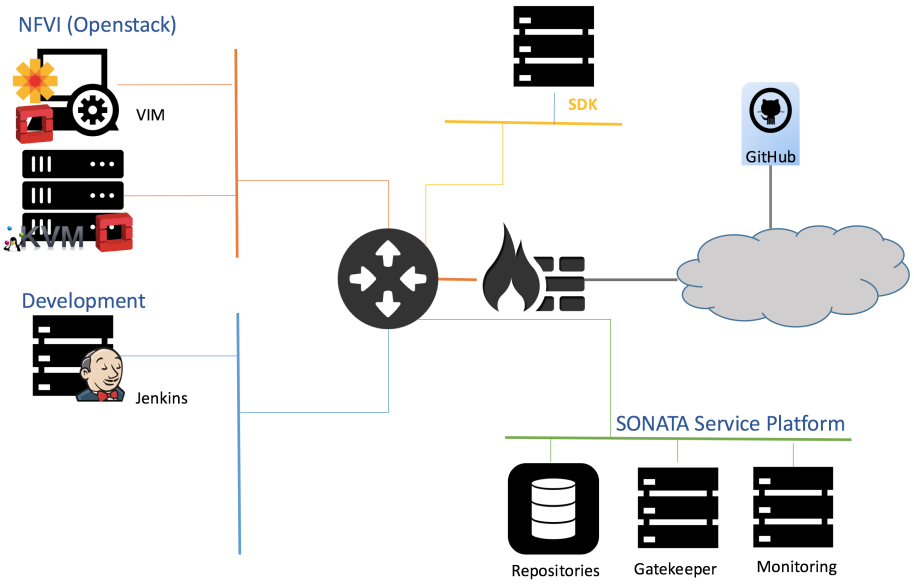

Figure 3.10: SONATA Integration Infrastructure

The Integration Infrastructure comprises 3 domains, namely:

- Network Function Virtualisation Infrastructure (NFVI) domain - used for the instantiation of the VNFs and for sandbox support during the Dev stages of the DevOps
- SONATA Service Platform (SP) domain - used for the deployment of the integrated components of the SONATA SP platform. This domain can support multiple instances of SONATA SP platform, used for integration deployment and testing
- Development domain - used for the support of CI/CD methodology followed for the whole lifecycle of the SONATA development
- SDK domain - used for the deployment of SONATA SDK framework

3.2.6 Deployment of the Integration Infrastructure

The Integration Infrastructure is the physical and logical support for the Sonata's Integration Test platform. This section presents the deployment of the following resources:

Networking

- the address space for the GK, CICD, SP, Catalogues, Infrastructure Adapter and the instantiated VNFs on a multi-site topology

For Y1 demos, the Integration platform run at the NCSRD infrastructure with IP addresses in the range of "143.233.127.0/27".

CI/CD

- the deployment of the CI/CD engine

Manual approach:

1. deploy a VM to install and configure Jenkins
2. deploy a VM to install and configure Monitory tool (prometheus.io)
3. deploy a VM to install and configure Docker Registry
4. deploy a VM to install and configure the SP modules, namely the Gatekeeper

Automated approach:

1. deploy a VM to install and configure a stripped SP (ie, containing the GK services, **Jenkins** test tool, Repositories)

For details on the CI/CD deployment for the Integration platform look at "Section 3.2.6.1" section.

NFVI

- the existing NFVI may have running VMs with VNF/NSs
- the deployment of a new NVFI (based of the "add-vim" feature) is a capacity of the Infrastructure Adapter and is in the scope of the SP

For details on the NFVI for the Integration platform look at "NFVI" section.
SP

- The SP is a stripped SP (ie, containing the GK services, **Jenkins** test tool, Repos)

For details on the SP for the Integration platform look at "SP deployment" section.
SDK

- the deployment of the SDK environment

For details on the SDK for the Integration platform look at "SDK" section.

3.2.6.1 Deployment of the Integration Infrastructure

The Integration Infrastructure is the physical and logical support of the Continuous Integration process that was described in D5.2 [17] to support the integration environment in SONATA Pipeline. This section presents the install, configure and startup process of the tools that are part of the integration environment.

Integration environment servers are:

- Jenkins
- Integration servers 1, 2 and 3
- Logging Server
- Docker Local Registry
- Monitoring Server

Integration Jenkins Server

The Jenkins [26] server in the SONATA pipeline is the key of the continuous integration workflow.

Jenkins installation process

The Jenkins installation process is described in this link, Jenkins installation. Basically, it consists of creating a VM with ubuntu and installing the Jenkins packages using apt-get tool. The current Jenkins version is 1.638(11/11/2015). SONATA have plans to upgrade Jenkins to version 2.X.

The Jenkins service is behind an apache server and it uses a certificate for https. This certificate is generated automatically by an script and use "Let's Encrypt" [20] as certificate authority.

The Jenkins server also has installed external tools for continuous integration and continuous deployment. These tools are: docker engine, pep8, and ansible.

Jenkins configuration process

Jenkins is closely integrated with Github source code version control system. It uses the github authentication plugin github-oauth. Each member of SONATA developer team have an account in github and use the same authentication to login in Jenkins.

Once the Jenkins is installed on the VM and integrated with Github, it needs some plugins for CI/CD SONATA's pipeline. The most important plugins installed are:

- build-pipeline-plugin v1.5.2

- checkstyle v3.45
- cobertura v1.9.7
- extended-choice-parameter v0.74
- extensible-choice-parameter v1.3.2
- findbugs v4.64
- github v1.17.1
- github-api v1.72
- github-oauth v0.22.3
- global-build-stats v1.3
- htmlpublisher v1.11
- jclouds-jenkins v2.8.1-1
- jobConfigHistory v2.13
- junit v1.11
- log-parser v2.0
- mailer v1.16
- maven-plugin v2.12.1
- openstack-cloud v2.8
- pipeline-rest-api v1.4
- pipeline-stage-view v1.4
- violations v0.7.11

Jenkins startup process

Jenkins will start automatically in the VM with the apache2 server. The configuration files of Jenkins are in the folder /var/lib/jenkins. If the Jenkins service is stopped, it can be started using the command "service jenkins start". If the apache2 service is stopped, it can be started using the command "service apache2 start".

Jenkins validation process

In order to validate that the Jenkins server and apache2 are running, run on the server the commands "service jenkins status" and "service apache2 status". The Jenkins dashboard should be accessible by "`https://jenkins.sonata-nfv.eu`".

The certificate issued by let's encrypt should be like the following picture:

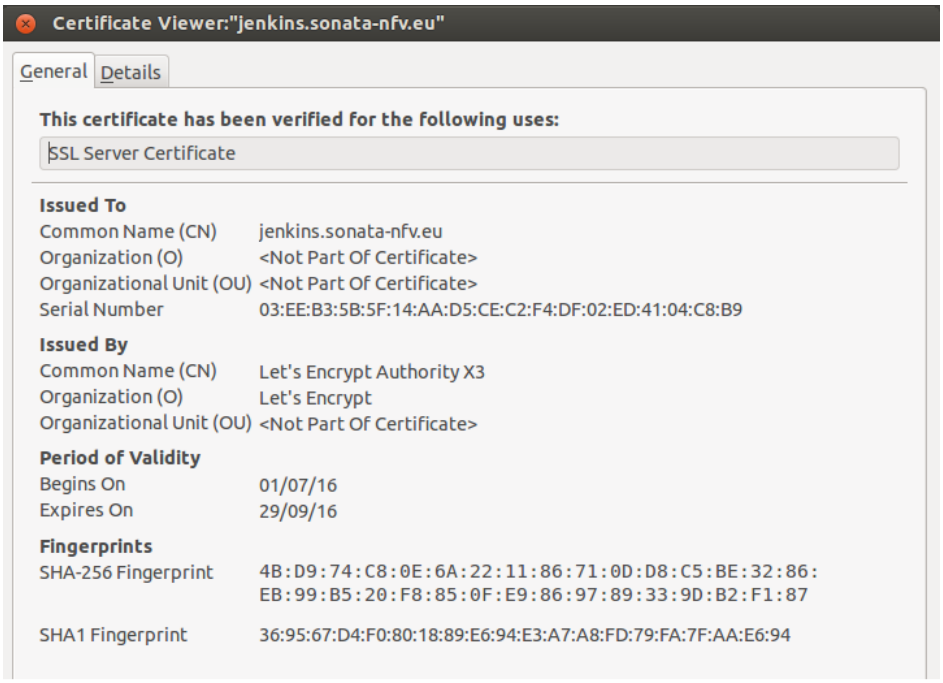

Figure 3.11: Jenkins https certificate

Jenkins Jobs

SONATA has organized the jobs per phases. Development, Integration and Qualification. Each phase was grouped by a regular expression according the start of the job's name, son-* for development phase, int-* for integration phase and qual-* for qualification phase.

The jobs that are running in the integration environment correspond with the development and integration phases defined in D5.2 [17]. This is a list of jobs per phase:

Development phase:

- son-analyze
- son-bss
- son-catalogue-repos
- son-cli
- son-emu
- son-gkeeper
- son-gui
- son-mano-framework
- son-monitor
- son-monitor-probe
- son-schema
- son-sdk-catalogue

- son-sp-infrabstract

Integration phase:

- int-bss-gkeeper
- int-emu-push-schema
- int-environment-deploy
- int-gtkpkg-sp-catalogue
- int-gui-gk-mon
- int-mano-plugin-management
- int-mon
- int-sdk
- int-sdk-pipeline
- int-sdk-son-analyze
- int-sdk-son-monitor
- int-service-instantiation-E2E
- int-slm-infrabstract

Integration Servers 1, 2 and 3

The SONATA project has 3 VMs to run the integration tests that are mentioned in the Development and Integration phases [17]. In each VM, docker engine is installed to support the SONATA Service Platform.

Integration servers installation process

The installation process consists of the deployment of a VM with the docker engine, configuration of an external IP address in Openstack and change of the default listen port of docker service.

Integration servers configuration process

To configure the docker engine, edit the file /etc/default/docker with the option --insecure-registry, expose the listen port 2375 for all the interfaces and adapt the MTU to 1390 because VxLANs issues.

The /etc/default/docker should be:

```
DOCKER_OPTS="--insecure-registry registry.sonata-nfv.eu:5000 \
-H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375 --mtu=1390"
```

To finish the configuration the server have to join to local docker registry:

```
docker login -u user -p password registry.sonata-nfv.eu:5000
```

Integration servers startup process

The docker service should start automatically when the VM is booting. In case it doesn't start automatically, then it can be started running the command "service docker start".

Integration servers validation process

To check if the docker service is running, you can use the netstat command and see if it is listen on port 2375.

```
netstat -natup | grep 2375
tcp6  0  0  :::2375  :::*  LISTEN  814/docker
```

The other check could be try to pull some container from local docker registry:
`docker pull registry.sonata-nfv.eu:5000/son-bss`

Monitoring Server

SONATA project has a monitoring server for the CI/CD environment. In this server has been configured two monitoring software, **Nagios** and **Cacti**. Nagios is used as an alert system and Cacti is used as a statistics system to analyse the status of the CI/CD VMs and dimension it.

Monitoring servers installation process

Nagios

The installation process is described in the following link. It uses the apt-get tool to install the package nagios3 with the command:

```
sudo apt-get install -y nagios3
```

It requires an apache2 server.

Cacti

The installation process uses the apt-get tool to install the package cacti and cacti-spine with the command:

```
sudo apt-get install snmpd cacti cacti-spine
```

It requires a mysql database and apache2 server.
For each CI/CD's VM, the snmpd package was installed with the command:

```
apt-get install snmpd
```

Monitoring servers configuration process

SNMPD

The Simple Network Management protocol daemon is installed in each CI/CD VM to gather the information about CPU, Disk, Load, Memory. This information will be used by Nagios and Cacti. The SNMP configuration daemon is the following:

```
###################################
#### FILE /etc/snmp/snmpd.conf ####
###################################
agentAddress udp:161,udp6:[::1]:161
```

```
view   all         included   .1                80
view   systemonly  included   .1.3.6.1.2.1.1
view   systemonly  included   .1.3.6.1.2.1.25.1
rocommunity s0n@t@  default    -V all
rouser   authOnlyUser
sysLocation    Demokritos node 1
sysContact     Me sonatanfv@gmail.com
sysServices    72
proc  mountd
proc  ntalkd    4
proc  sendmail 10 1
disk       /     10000
disk       /var  5%
includeAllDisks  10%
load   12 10 5
trapsink     localhost public
iquerySecName   internalUser
rouser          internalUser
defaultMonitors          yes
linkUpDownNotifications  yes
master          agentx
```

Cacti

The procedure to configure the cacti server to add the VMs is through the web interface. **Console -> Device -> Add**, and fill the gaps with the specific parameters of each server. The Hostname, the template to create the graph (linux generic), the snmp community.

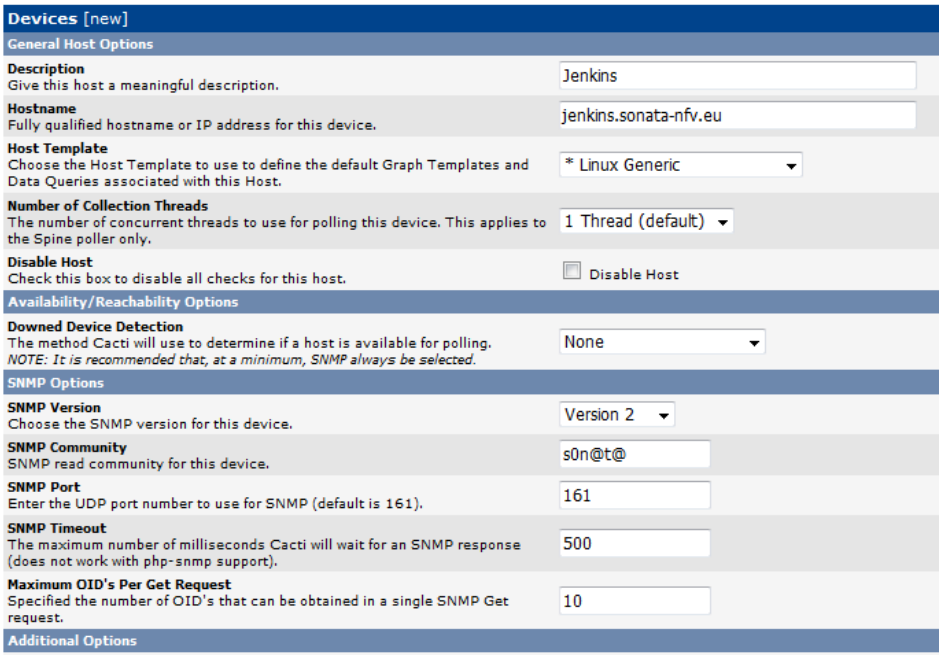

Figure 3.12: Cacti configuration screen

Nagios

The procedure to configure the nagios server is composed by the following steps:
1.Add the contact in the file contacts.cfg

```
######################
#### contacts.cfg ####
######################
define contact{
       contact_name                    root
       alias                           Nagios-sonata-nfv
       service_notification_period     24x7
       host_notification_period        24x7
       service_notification_options    w,u,c,r
       host_notification_options       d,r
       service_notification_commands   notify-service-by-email
       host_notification_commands      notify-host-by-email
       email                           felipe.vicens@atos.net
       }
```

2.Add the hosts in the monitoring.cfg file
3.Add the configuration of each servers. This is an example:

```
########################################
####/etc/nagios3/servers/jenkins.cfg####
########################################
define host {
        use                             generic-host
        host_name                       jenkins
        alias                           Jenkins Integration Server
        address                         192.168.60.5
        max_check_attempts              5
        check_period                    24x7
        notification_interval           30
        notification_period             24x7
}
define service {
        use                             generic-service
        host_name                       jenkins
        service_description             PING
        check_command                   check_ping!100.0,20%!500.0,60%
}
define service {
        use                             generic-service
        host_name                       jenkins
        service_description             SSH
        check_command                   check_ssh
}
```

Monitoring servers startup process

SNMPD

The snmpd daemon is started when the VM is booting. If it won't start, the snmpd daemon could be started using the command service snmpd start.

Monitoring servers validation process

SNMPD

The validation process of snmpd consists of a request to the VMs that have the snmpd running using the tool snmpwalk. This command could be used to validate if the snmpd is working:

```
snmpwalk -c community -v2c ip_address_of_server
```

Nagios

The webpage of nagios for SONATA project is: `http://monitor.sonata-nfv.eu/nagios3`.

Cacti

The webpage of cacti for SONATA project is: `http://monitor.sonata-nfv.eu/cacti`.

Logging Server

Graylog server installation process

The installation consists of deployment in a VM with docker engine, the containers mongodb, elasticsearch and graylog. The VM should have a large disk space to support the amount of logs that CI/CD SONATA's workflow will generate.

Graylog server configuration process

The configuration of the graylog is perfomed by altering the following parameters:

```
mongo:
 image: "mongo:3"
 volumes:
   - /home/jenkins/graylog/data/mongo:/data/db
elasticsearch:
 image: "elasticsearch:2"
 command: "elasticsearch -Des.cluster.name='graylog'"
 volumes:
   - /home/jenkins/graylog/data/elasticsearch:/usr/share/elasticsearch/data
graylog:
 image: graylog2/server:2.0.1-1
 volumes:
   - /home/jenkins/graylog/data/journal:/usr/share/graylog/data/journal
   - /home/jenkins/graylog/config:/usr/share/graylog/data/config
 environment:
   GRAYLOG_PASSWORD_SECRET: secret
   GRAYLOG_ROOT_PASSWORD_SHA2: root_password_sha2
   GRAYLOG_REST_TRANSPORT_URI: http://ip_vm:12900
 links:
   - some-mongo:mongo
   - some-elasticsearch:elasticsearch
 ports:
   - "80:9000"
   - "12900:12900"
   - "12900/udp:12900/udp"
```

It has a mongodb container with a volume in the local disk. It also has a container with elasticsearch and a volume for the elasticsearch data, as well as a docker container with graylog and a volume for the config and data. The password configuration is present in the docker-compose file and the exposed ports.

Graylog server startup process

Once the configuration is done using the docker-compose file, the startup process is:

```
docker-compose -d up
```

Graylog server validation process

```
docker-compose ps -a
```

To validate if the Docker local registry is running, you can use the command docker-compose ps and it should return the following:

```
CONTAINER ID        IMAGE                       STATUS
7bdbd4c49048        graylog2/server:2.0.1-1     Up 13 days
17dc9f48325b        mongo:3                     Up 13 days
3c3f60e0db41        elasticsearch:2             Up 13 days
```

The other way is go to the webpage `http://logs.sonata-nfv.eu` and check if it opens and works.

Docker Local Registry

For SONATA project, the Docker Local Registry is an important tool for continuous integration and continuous delivery. The function that this tool has in the workflow is to fist store the Docker images, to distribute it later between the pipeline's phases. As the docker local registry is present locally in the testbed, it saves internet bandwidth and improves the speed of deployments.

Docker Local Registry installation process

The installation consists in deploying a VM with docker engine the container registry:2 with an autogenerated certificate to allow https connections. The VM should have a large disk space to support the amount of images that CI/CD SONATA's work-flow will generate.

Docker Local Registry configuration process

The configuration of the registry is composed by the following parameters:

```
registry:
 restart: always
 image: registry:2
 ports:
   - 5000:5000
 environment:
   REGISTRY_HTTP_TLS_CERTIFICATE: /certs/registry.sonata-nfv.eu.crt
   REGISTRY_HTTP_TLS_KEY: /certs/registry.sonata-nfv.eu.key
   REGISTRY_AUTH: htpasswd
```

```
    REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
    REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
    SEARCH_BACKEND: sqlalchemy
  volumes:
    - /mnt/data:/var/lib/registry
    - /mnt/certs:/certs
    - /mnt/auth:/auth
```

This service has as requirement a self-signed certificate in the route/mnt/certs. In the folder /mnt/auth is present the htpasswd file for the docker registry authentication.

Because the certificate is self-signed, all the docker clients should have added in the default docker file the following option:

```
DOCKER_OPTS="--insecure-registry registry.sonata-nfv.eu:5000"
```

Docker Local Registry startup process

Once the configuration is done using the docker-compose file, the startup process is:

```
docker-compose -d up
```

Docker Local Registry validation process

To validate if the Docker local registry is running, you can use the command docker-compose ps and it should return the following:

```
      Name                     Command               State           Ports
-------------------------------------------------------------------------------------
docker_registry_1   /bin/registry /etc/docker/ ...   Up      0.0.0.0:5000->5000/tcp
```

The other way to validate the installation is through a remote docker client trying to upload an image to docker local registry. First the client have to do a login:

```
docker login -u sonata-user -p sonata-password registry.sonata-nfv.eu:5000
```

Download an image

```
docker pull ubuntu
```

Tag and image to the new registry

```
docker tag ubuntu registry.sonata-nfv.eu:5000/ubuntu
```

Upload the image to the new registry

```
docker push registry.sonata-nfv.eu:5000/ubuntu
```

3.2.6.2 Service Platform

The Service Platform for Integration (SP4II) tests is a stripped version of SP optimised to serve the CI engine.

The SP4II runs on a dedicated VM: the internal SP services run inside dedicated Docker containers, for isolation purposes and micro-services design principles.

SP software components

According to the defined architecture, the SONATA SP for Y1 is built of the following main components:

- **GK**
  - GK-SRV, GK-API, GK-GUI and GK-BSS containers
- **MANO framework**
  - SLM, SSM, FSM and Plugin-Manager containers
- **Monitory**
  - Prometheus Monitory server, Monitory Manager, Push-Gateway and InfluxDB
- **Infrastructure Abstraction** (aka, IFTA) containing:
  - VIM-Adapter
  - WIM-Adapter

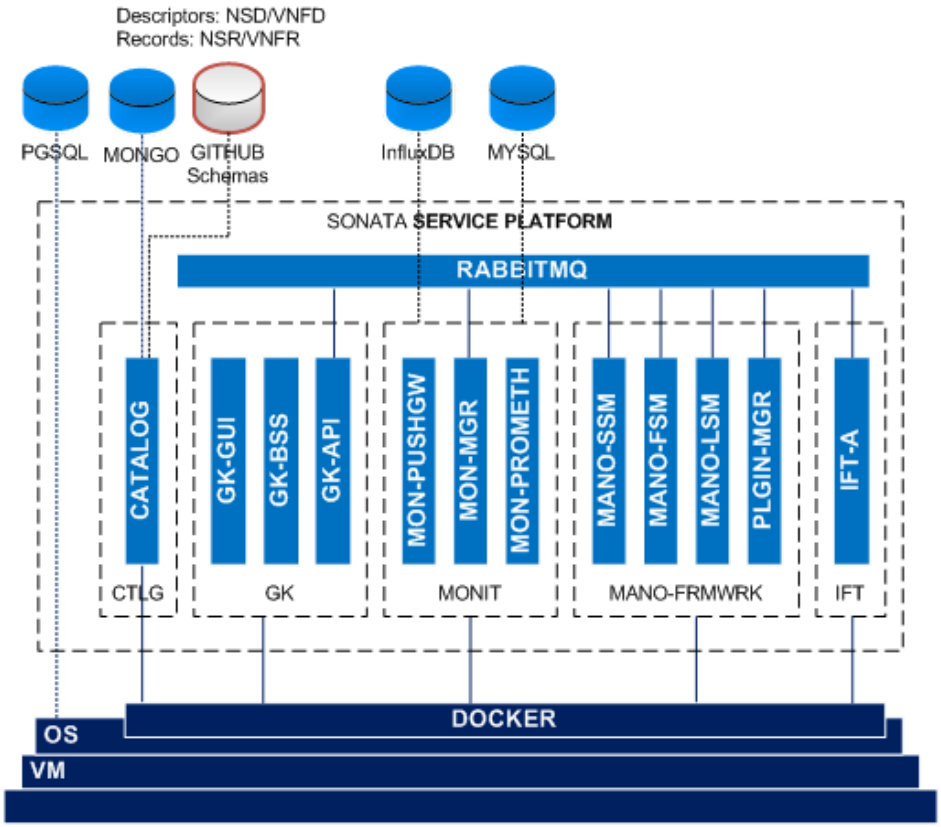

Figure 3.13: SP for II

The fundamental characteristic of the SP4II is that it doesn't necessarily need a NFVI to deploy resources, ie, it can instantiate local VM (for example, on top of KVM) for Integration test purposes.

SP initialisation process

The SP4II is started up in an automatic way by running the "son-install" playbooks available at Github.

```
$ cat deploy-sp-all.yml
---
# update database of local packages
- include: deploy-common.yml
# install, configure and start NGINX
- include: deploy-nginx.yml
# install, configure and start PostgreSQL database
- include: deploy-pgsql.yml
# install, configure and start MySQL database
- include: deploy-mysql.yml
# install, configure and start MongoDB database
- include: deploy-mongo.yml
# install, configure and start Docker engine
- include: deploy-docker.yml
# deploy RabbitMQ message bus
- include: deploy-sp-broker.yml
# deploy Gatekepper module
- include: deploy-sp-gk.yml
# deploy Repositories
- include: deploy-sp-repos.yml
# deploy MANO framework
- include: deploy-sp-mano.yml
# deploy Monitoring
- include: deploy-sp-monit.yml
```

The SP4II can be invoked by one of the following methods:

- at a Linux shell with Ansible installed
- as a Jenkins job
- as an Openstack Heat stack or Nova CLI
- as a Terraform template

3.2.6.3 NFVI

For the Integration Infrastructure flavour it was decided, for reasons of easier maintenance and reduced overhead, that the integration infrastructure and the NFVI-PoP infrastructure would be colocated on the same private cloud. The separation of the two infrastructures is achieved through the employment of the multi-tenant features of the hosting cloud management system (i.e Openstack). For the NFVI a separate tenant is introduced to the system. The configured number of PoPs available for the integration infrastructure is one.

3.2.6.4 SDK

The SDK deployment for the Integration Infrastructure is composed of several modules running inside dedicated Docker containers for a better manipulation and isolation of services.

SDK Components

The SDK is composed of the following main modules:

- son-cli which contains the tools:
  - son-workspace
  - son-package
  - son-publish
  - son-push
  - son-monitor
- son-catalogue components are:
  - MongoDB datastore with separate collections for different descriptor types
  - Front-end REST API enables access for other SDK modules to the mongoDB
- son-monitor
  - Prometheus Database which gathers exposed metrics from son-emu
  - Prometheus Push Gateway which receives exposed metrics from -son-emu
  - A set of tools and commands implemented in son-emu that can be used to monitor a service deployed in the emulator
  - A set of tools and commands implemented in son-cli so the developer can access monitor-data from the development workspace
- son-analyze !please provide description!
- son-emu consisting of:
  - son-emu
  - son-emu-cli (also integrating commands used by son-monitor)
  - son-dummy-gatekeeper

SDK Initialisation

The deployment of the SDK environment involves the individual deployment of each module, described as follows.

- son-cli: this component is deployed to a docker image, namely `registry.sonata-nfv.eu:5000/son-cli`.
  The sequences of integration tests are executed inside a container named `son-cli-int-test`. Moreover, a web terminal service is deployed at '`http://sdk.int.sonata-nfv.eu:8000/`' to perform manual tests within the integrated environment.
- son-catalogue:
  - component deployed to a docker image, namely `registry.sonata-nfv.eu:5000/sdk-catalogue` with exposed port at `4012`.

  - mongoDb database deployed to docker images, named `mongodata` and `mongo` with exposed port at 27018.
  - mongoDb database management deployed to a docker image `mongoexpress` exposed at 'http://api.int.sonata-nfv.eu:8082/'.

- son-monitor:
  - stand-alone Prometheus Database docker container (`prom/prometheus`)
  - stand-alone Prometheus Push Gateway docker container (`prom/pushgateway`)
  - A set of commands packaged together with son-cli and son-emu
- son-analyze !please provide description!
- son-emu:

The son-emu package plays a special role within the SDK since it has to be installed in a dedicated virtual machine that isolates it from the rest of the tools. This is needed because son-emu uses Containment to emulate a multi-PoP NFVI environment inside a single machine. To do so, son-emu needs to be executed with *root* privileges to create and reconfigure virtual network interfaces as well as to run its own dedicated Docker engine to execute VNFs in the emulated network.

The following steps are needed to deploy son-emu in the infrastructure.

1. create a fresh Ubuntu 14.04 LTS (or 16.04 LTS) VM
2. ```
   install requirements:\\
   sudo apt-get install ansible git
   ```
3. ```
   Add to /etc/ansible/hosts:\\
   localhost ansible\_connection=local to /etc/ansible/hosts
   ```
4. ```
   install containernet.\\
   git clone \url{https://github.com/mpeuster/containernet.git}\\
   cd \textasciitilde{}/containernet/ansible\\
   sudo ansible-playbook install.yml
   ```
5. ```
   install son-emu\\
   git clone \url{https://github.com/sonata-nfv/son-emu.git}\\
   cd \textasciitilde{}/son-emu/ansible\\
   sudo ansible-playbook install.yml
   ```

3.3 Qualification Infrastructure

The **Qualification Infrastructure** is an automated Quality Assurance platform with Continuous Deployment capabilities.

The output from the Integration phase can trigger the execution of Qualification tests on the Qualification environment.

The Qualification tests implies the deployment of resources to the available NFVI for advanced testing.

The output of successful Qualification tests produces a major (or minor) VNF/NS release automatically published on the Service Catalogue.

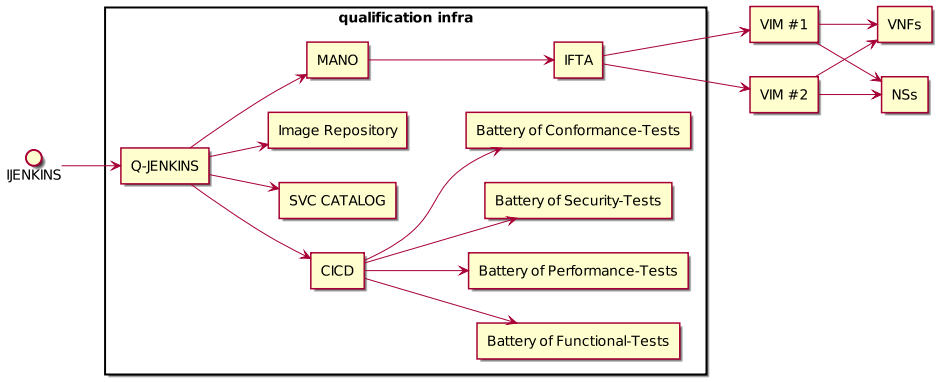

Figure 3.14: Qualification Test Cycle

As an example of required infrastructure for Qualification test purposes, consider the vCDN testing Use Case: the vCache nodes must be deployed to end-user nearest geographical location to get lower network packets latency.

3.3.1 Design and specification

3.3.1.1 Assumptions

The assumptions related to the QI are:

- The SP components should be developed as Docker containers (or any other preferred container technology type (e.g. LXC)) - currently the decision is to use Docker
- The capability of the system to support multiple SP instances is not at the moment a priority of the QI for the first year
- SP will be deployed by automated scripts via Ansible on a standalone platform (separate server or a VM provisioned in an Openstack Environment)

3.3.1.2 Component Enumeration

The figure below (Figure 3.15)

- Two NFVI PoPs based on Openstack (at least for the first year) comprised of a number of compute nodes
- SONATA SP (Multiple components deployed in a single VM as docker containers). See Figure 2 for details
- Developer Tools

Hosted tools that allow the CI/CD (i.e. Ansible Jumphost)

- Simulated WAN or a simple L3 or SDN based interconnection between the PoPs

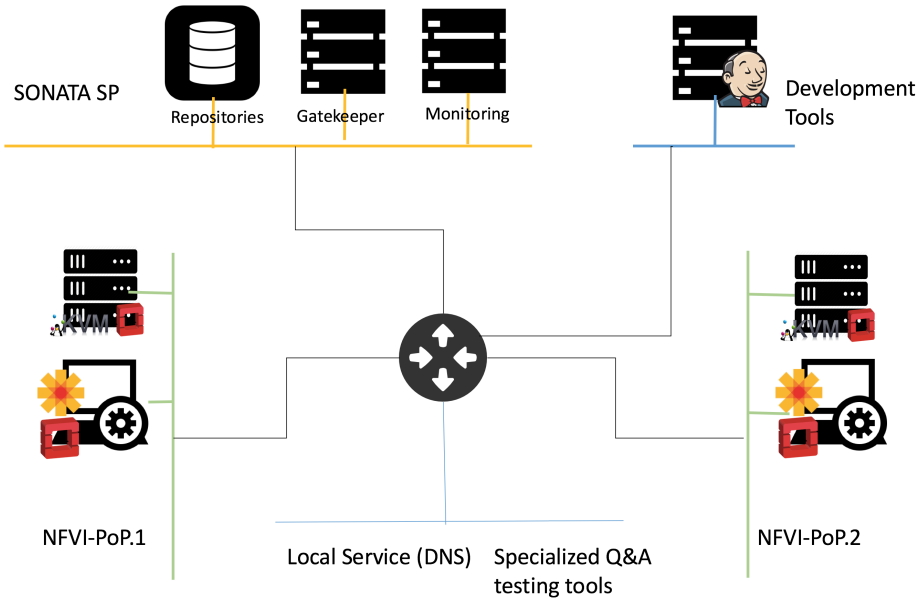

Figure 3.15: SONATA Qualification Infrastructure

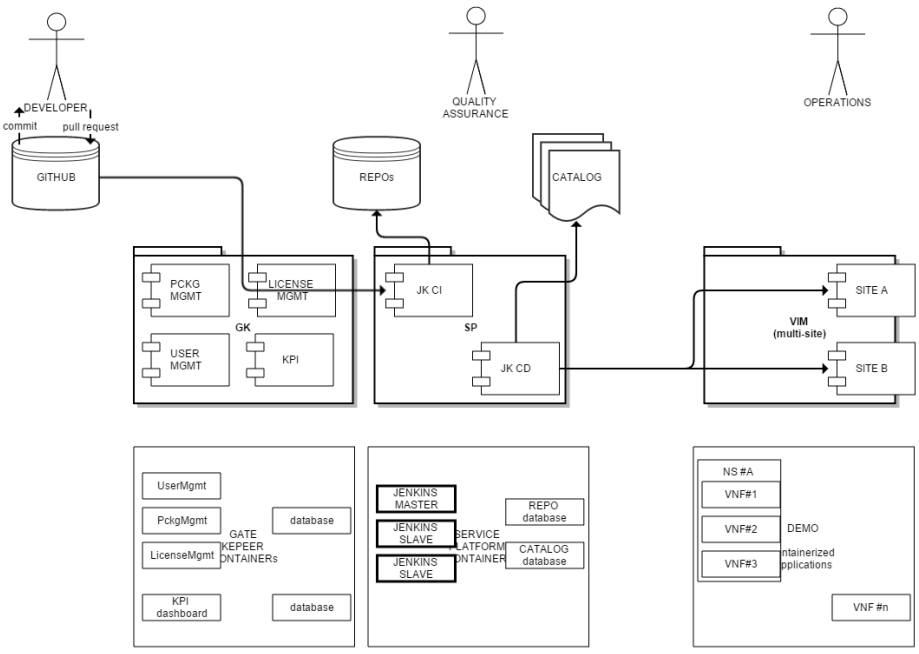

Figure 3.16: map entities to logical resources

- a GK module with 4 containers (UserMgmt + PckgMgmt + LicMgmt + KPIs) serving the SDK
- a Repo module to store packages coming from the CI
- need to choose the open source Repository Manager – between Artifactory, Maven, or Nexus
- the Repository Manager can run on a dedicated VM or on a LXC inside the SP appliance (because it won't be too large)

- the Catalog module to store the application catalogues defined with TOSCA templates or ETSI descriptors (or support both?)
- also can be a dedicated VM or a LXC with the catalog database (MySQL/MariaDB or pgSQL)
- the SP CI engine with Jenkins Master and (eventual) Jenkins Slaves to produce new or updated packages
- the SP QA engine also based on Jenkins to run functional tests and (nice to have) performance, security, conformance

3.3.2 Deployment of the Qualification Infrastructure

The deployment of the Qualification Infrastructure for Sonata is based on Ansible playbooks for automatic deployment of resources on top of a compatible VIM.

A single Linux shell is enough to pull "**son-qual-all-in-one**" from Github and automatically deploy all the computational, networking and storage resources need for the Qualification environment. The Ansible playbooks (described in the next sections) are in charge of:

- requesting one or more VMs for the SP platform
- installing all the database engines used by SP services
  - eg: pgsql, mysql, mongodb, influxdb
- pulling SP service modules
  - eg: GK, CICD, MANO, Repos, IFT-A and Monitory
- connecting to compatible multi-VIM
  - eg, Openstack

Figure 3.17 describes the automated deployment process for the Qualification environment.

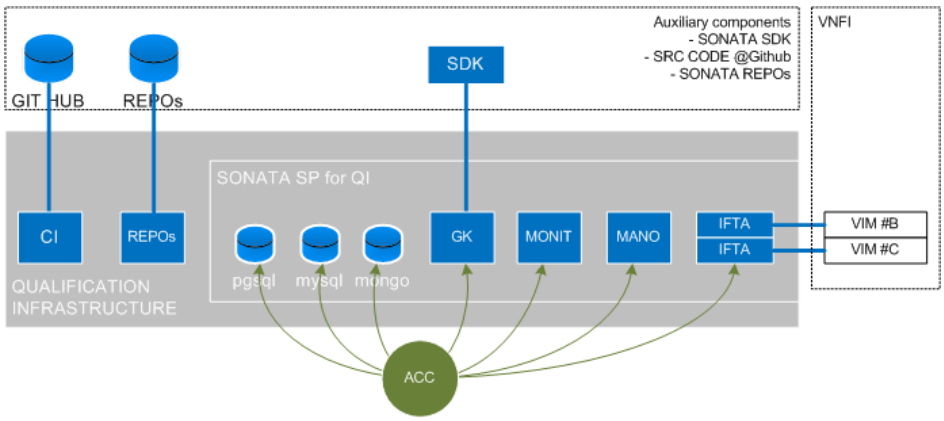

Figure 3.17: QI automated deployment

The deployment of individual services is presented in the following sections.

3.3.2.1 Deployment of the Qualification Infrastructure

The Qualification Infrastructure is the physical and logical support of the Continuous Integration process that was described in D5.2 WikiBibliography#5.2 to support the qualification environment in SONATA Pipeline. This section presents the install, configure and startup process of the tools that are part of the integration environment.

Qualification environment servers are:

- Jenkins
- Docker Local Registry
- Qualification server

Qualification Jenkins Server

It is the same server that SONATA uses in the integration Infrastructure. It provides the automatic deployment of the new Qualification VM through openstack using Jclouds Jenkins plugin.

It has an specific job to handle the installation of SONATA service platform that is qual-son-install.

Docker Local Registry

The qualification environment uses the same Docker Local Registry of integration environment. The Docker Local Registry is used by the Jenkins job qual-son-install to pull the containers.

Qualification server

The Deployment of the Qualification server is triggered by Jenkins using the Openstack Cloud Plugin. It creates a VM from scratch with the following parameters. This configuration will create a VM with a retention time about 18hours.

- Cloud Openstack:

```
Cloud Name: jclouds-ubuntu-qual
EndpointURL:http://10.31.1.3:5000/v2.0
User Identity: sonata.qual:sonata.qual
Password: sonata-password
Default Openstack Region:RegionOne
```

- Cloud Instance Templates:

```
Name: jclouds-qual
Labels: jclouds-qual
Image Name: jclouds-ubuntu-1404-735-snap
User Data: cloud-init
Max.No.of.Instances: 1
Associate floating IP from pool: admin_floating_net
Security Groups: default, testbed
Availability Zone: nova
Startup Time: 6000000
Key Pair Name: jenkins
Retention Time: 1080
```

- Cloud Init script file:

```
users:
name:jenkins
sudo:
ALL=(ALL)~NOPASSWD:ALL
groups:~sudo,adm,root
lock\_passwd:~false
plain_text_passwd: jenkins}
```

Inside the job, Jenkins runs the following installation script:

```
#!/bin/bash
set -x
set -e
set -v
sudo apt-get clean all
sudo apt-get install -y software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install -y ansible
sudo ansible-playbook deploy-sp-all.yml -e targets=localhost
```

The execution process and details about the ansible script is detailed on the section Section 3.3.2.2 Qualification Infrastructure SP Service Deployment

3.3.2.2 Service Platform for the Qualification Infrastructure

The SP for QI (SP4QI) is based on a large VM running the SP services optimised for Qualification tests.

The SP services available at the Qualification Infrastructure are:

- Gatekeeper services
- Catalog repos
- MANO framework
- IFT-Abstr Adapters
- Monitory services

The Figure 3.18 shows a global view of the Qualification Infrastructure. It includes the SP4QI and the remaining ecosystem of SONATA applications, namely:

- CI/CD engine - to continue running qualification tests
- SDK - for qualification tests
- NVFI - to instantiate VNF/NS to execute qualification tests against

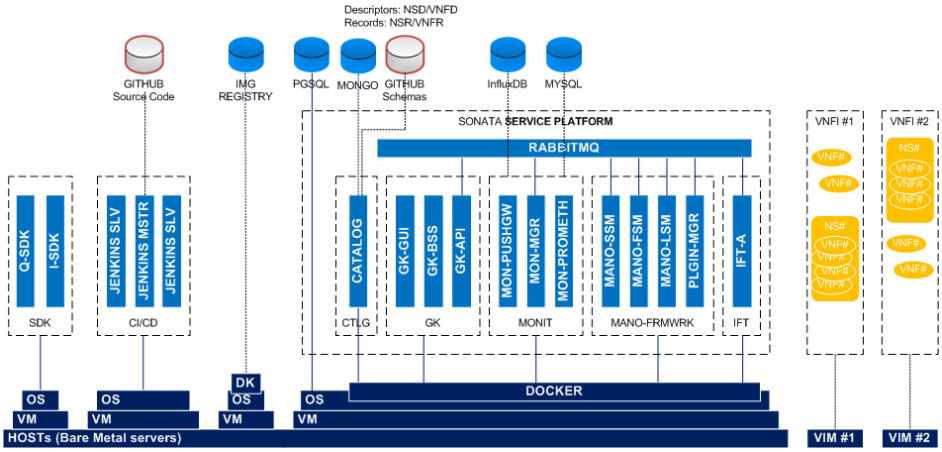

Figure 3.18: global Sonata QI

SP Initialisation process

The workflow to initialise a SP follows the steps:

```
# Build an Ansible Control Centre (get a Linux shell)
# Get son-sp-qual-all-in-one.yml Ansible playbooks
     from [https://github.com/sonata-nfv/son-install.git Github]
# Instantiate a VM for SP
# Run son-sp-qual-all-in-one.yml to deploy SONATA SP Qualification platform
```

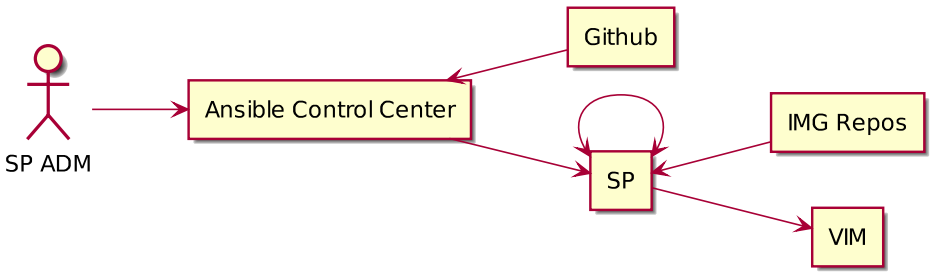

Figure 3.19: Qualification Test cycle

Building an Ansible Control Center

Ansible is a general purpose automation tool used for:

- Configuration Management
- Orchestration Management
- Dynamically Provision Infrastructure

Ansible is supported by almost every Linux distros, namely:

- Install Ansible on any **CentOS 7** based guest

```
$ sudo apt clean all
$ sudo yum update
$ sudo rpm -iUvh \
      http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-7.noarch.rpm
$ sudo yum install -y ansible
$ ansible --version
$ sudo yum install -y git
```

- Install Ansible on any **Ubuntu 16.04** based guest

```
$ sudo apt clean all
$ sudo apt install -y software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt update
$ sudo apt install -y ansible
$ ansible --version
$ sudo apt install -y git
```

Getting the deployment scripts

The all-in-one SP deployment scripts for the Qualification environment are available from Github:

```
$ git clone https://github.com/sonata-nfv/son-install.git
$ cd son-install
```

VM instantiation for SP

There are alternative methods to create a guest VM to deploy the SP. Here, only 3 methods are mentioned, namely based on:

- KVM management tools
- Openstack
  - 'nova' client
  - 'heat' client
- Terraform

Follow details for VM instantiation here: `http://wiki.sonata-nfv.eu/index.php/WP6/D61_generic_vm_deployment`

The guest VM for SP can be instantiated on top of:

- any type 1 hypervisor (like KVM, ESX, Hyper-V or Xen) or
- any type 2 hypervisor (like Oracle VirtualBox, Vmware Workstation/Player)

The baseline image for the guest VM can be obtained from already build **cloud images** (with **cloud init** feature already included) - eg:

- CentOS 7 Cloud Image
- Ubuntu Cloud Images
- other Linux flavors (eg, OEL, SUSE, Debian)

Pre-configuration of the deployment host

Ansible is an agent-less tool that transfers a specific module to the remote machine to execute a specific action. To prepare this feature, some configuration is required at Ansible Control Center, namely you must populate your Inventory file containing all the managed guests and distribute the Ansible Control Center public key to all those managed machines.

- add managed guests to the Inventory - eg:

```
$ sudo vi /etc/ansible/hosts
son-sp ansible_host=localhost
```

- exchange security keys to remotely execute playbooks on the target machines - eg:

```
// generate a key pair
$ ssh-keygen -t rsa -b 4096 -C "son-install@sonata-nfv.eu"

// distribute Ansible public key to the Openstack controllers
$ ssh-copy-id sonata@''<guests>''
```

SP services deployment

To deploy SONATA SP platform, just run the following script (here we are using the same machine: "localhost"):

```
$ ansible-playbook deploy-sp-all.yml -e targets=localhost  [-vvvv]
```

This playbook executes sequentially the following plays:

```
$ cat deploy-sp-all.yml
---
# updating local repo info
- include: deploy-common.yml
# deploying NGINX
- include: deploy-nginx.yml
# deploying PostgreSQL
- include: deploy-pgsql.yml
# deploying MySQL
- include: deploy-mysql.yml
# deploying MongoDB
- include: deploy-mongo.yml
# deploying Docker engine
- include: deploy-docker.yml
# deploying RabbitMQ message bus
- include: deploy-sp-broker.yml
# deploying Gatekepper modules
- include: deploy-sp-gk.yml
# deploying Repositories
- include: deploy-sp-repos.yml
# deploying MANO framework
- include: deploy-sp-mano.yml
# deploying Monitory
- include: deploy-sp-monit.yml
```

SP deployment workflow

The next MSG describes the steps behind the automated process of SP deployment for Qualification purposes.

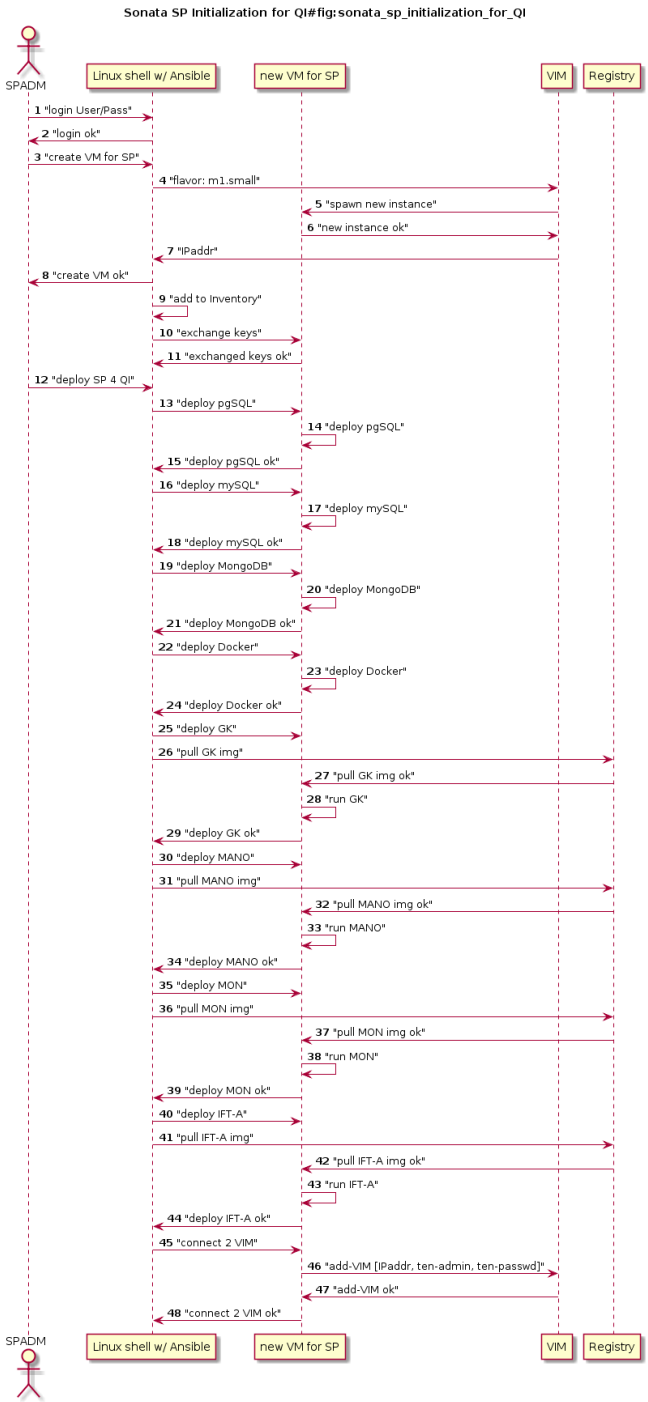

Figure 3.20: Sonata SP Initialization for QI

SP running modules

The initialisation process follows the Sonata modular application architecture, as shown in the picture.

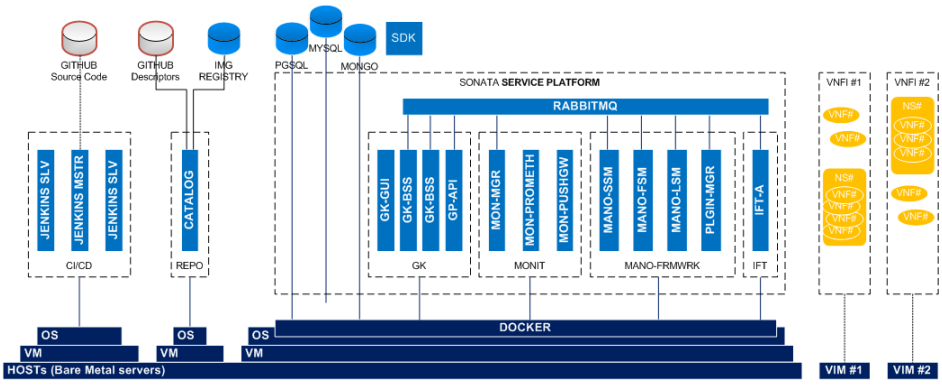

Figure 3.21: QI SP Components

3.3.2.3 Qualification NFVI

In Figure 3.22, the physical deployment of the Qualification Infrastructure is illustrated. As discussed in previous sections the SONATA SP is deployed over the same underlying datacenter infrastructure with the PoP, at least for the II and QI flavours. Two NFVI-PoPs may be observed in Figure 3.22, both running Openstack Liberty. NFVI-PoP 1 consists of a controller node and three more compute nodes (see Hosting Infrastructure section), whereas NFVI-PoP 2 is an all-in-one openstack deployment. These two PoPs are interconnected through an SDN based, laboratory scale WAN infrastructure that is managed and by WAN Infrastructure Manager (WIM) and controlled by OpenDaylight (ODL) SDN Controller.

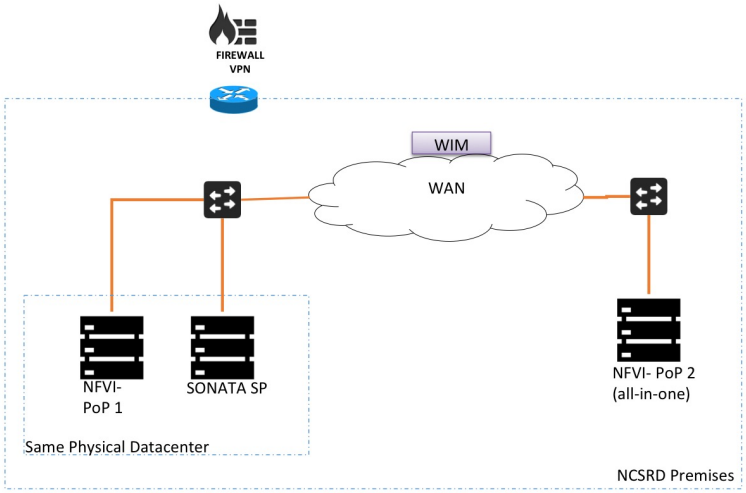

Figure 3.22: Qualification Infrastructure

3.3.2.4 SDK for the Qualification Infrastructure

The SDK modules deployed to the Qualification Infrastructure are:

- son-cli
- son-catalogue
- son-monitor
- son-analyze
- son-emu

In addition to the SDK modules, the Qualification Infrastructure is built of:

- Jenkins CICD engine responsible of running qualification tests
- Repository to store the latest built images of the modules

The modules are continually built.

SDK Initialization Process

In contrast with SP, where all modules are required to be deployed, the SDK modules are deployed individually as they have isolated functionalities. The following subsections describe the instantiation of each SDK module.

son-cli

Workflow to instantiate son-cli in the Qualification Infrastructure:

1. Prepare Environment

```
sudo apt-get install -y build-essential python-dev python-pip docker-engine
```

2. Get latest son-cli docker image from registry

```
export DOCKER_HOST="unix:///var/run/docker.sock"

echo DOCKER_OPTS=\"--insecure-registry registry.sonata-nfv.eu:5000 \
  -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375\"
  | sudo tee /etc/default/docker

sudo service docker restart

docker login -u sonata-nfv -p s0n@t@ registry.sonata-nfv.eu:5000

docker pull registry.sonata-nfv.eu:5000/son-cli
```

3. Deploy son-cli module in a container

```
docker run -d -i --name 'son-cli-int-test' \
    --net='host' --pid='host' --privileged='true' \
    registry.sonata-nfv.eu:5000/son-cli
```

son-catalogue

1. Deploy son-catalogue module in a container

```
docker pull registry.sonata-nfv.eu:5000/sdk-catalogue
```

2. Run sdk-catalogue services in docker containers

```
cd int-sdk-catalogue
docker-compose down
docker-compose up -d
```

son-monitor

The Jenkins integration test can be found in `int-sdk-son-monitor` in the Jenkins CICD environment
1. Deploy the Prometheus Database

```
docker run -d -p 9090:9090 -v \
"$PWD/misc/prometheus_jenkins.yml:/etc/prometheus/prometheus.yml" \
--name~prometheus prom/prometheus
```

2. Deploy the Prometheus Push Gateway

```
docker run -d -p 9091:9091 --name pushgateway prom/pushgateway
```

3. Deploy the son-emu container `registry.sonata-nfv.eu:5000/son-emu`

4. Deploy the son-cli container `registry.sonata-nfv.eu:5000/son-cli`

and use the son-monitor commands in the son-emu and son-cli containers.

3.4 Demonstration Infrastructure

The section that follows is not to be confused with the description of the Pilot Site deployment rather than the specification of the environment used for demonstrations. An early version of which will be the one delivered for the first Year review.

The demonstration infrastructure will comprise of all the components packetised in a way to allow deployment in Pilot Infrastructures. Deployment validation tests might be required to verify the correct deployment. However, not exhaustive qualification level testing will be required to be supported as e.g FUEL based OPNFV deployment includes test for the deployment of the OPNFV environment.

The **Demonstration Infrastructure** can be instantiated on demand by the Admin Tenant of the host infrastructure. DI will be an always-on and updated platform ready to showcase Use cases and deploy VNFs and/or NS, in practice acting as a SONATA showroom. The deployment of a DI automatically creates and manages resources on the available NFVIs, as shown on the picture.

While the Qualification Infrastructure (QI) has an intrinsic volatile characteristic (ie, to be destroyed at the end of the tests), the Demonstration Infrastructure (DI) is, by its own nature, more permanent along the time. This is the privileged platform to show VNF/NS in action.

The DI should be always-on for demonstration purposes: a team member or partner should be able to demonstrate the benefits of SONATA to potential clients or to other community projects.

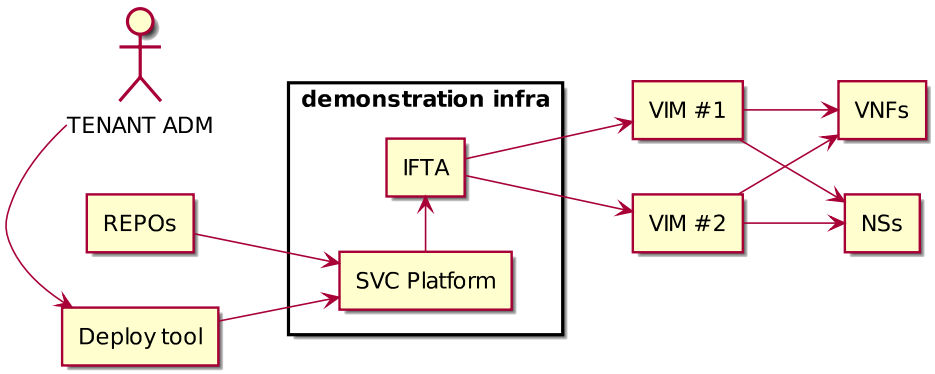

Figure 3.23: Demonstration Test cycle

3.4.1 Design and Specification

Figure 3.24 below illustrates the Demonstration Infrastructure deployment. Currently two testbeds located in Athens (NCSRD) and Aveiro (ALB) are participating. More testbeds are anticipated to be connected via VPN connections in order to scale up the demonstration infrastructure towards Pilot demonstrations.

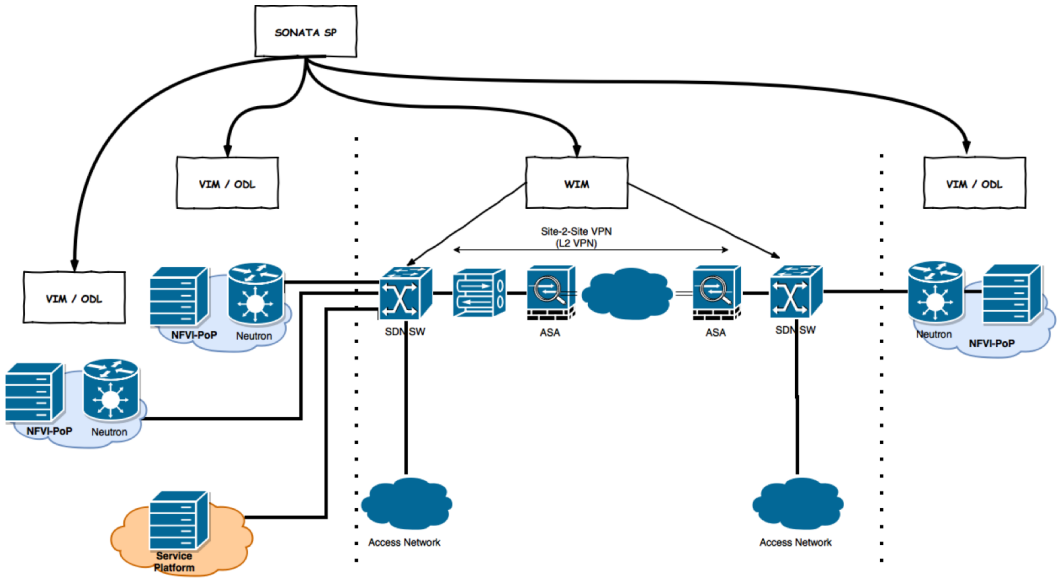

Figure 3.24: Demonstration Infrastructure topology

3.4.2 Deployment of the Demonstration Infrastructure

The Demonstration Infrastructure is the physical and logical support of the SONATA's demos. This section presents the installation, configuration and startup processes of the components that are part of the integration environment.

The adoption of the DevOps model in SONATA allows an ease of deployment of the Demonstration Infrastructure.

3.4.2.1 Demonstration Server installation process

Service Platform

The installation of the service platform is similar than Qualification Infrastructure. The server have to satisfy the software dependencies; The last version of Ansible and git, and then clone the sonata-nfv/son-install repository from Github and later run the ansible playbook to deploy the service platform on the demonstration server.

SDK

The installation of the SDK is similar to that of Qualification Infrastructure. The installation consists in the following steps:

- Add the new GPG key

```
sudo apt-key adv -\/-keyserver keyserver.ubuntu.com -\/-recv-keys D0DF34A30A4FE3F8}
```

- Add a source entry for your Ubuntu OS. For now, supported distributions are supported:
  - Ubuntu Trusty 14.04 (LTS)

```
echo "deb http://registry.sonata-nfv.eu:8080 ubuntu-trusty main"
sudo tee -a /etc/apt/sources.list
```

  - Ubuntu Xenial 16.04 (LTS)

```
echo "deb http://registry.sonata-nfv.eu:8080 ubuntu-xenial main"
sudo tee -a /etc/apt/sources.list
```

- Update and install

```
sudo apt-get update
sudo apt-get install sonata-cli
```

- Test if its working by invoking:

```
$ son-workspace -h
$ son-package -h
$ son-publish -h
$ son-push -h
```

Configuration of the Service Platform in the Demonstration Server

Once the service platform is running on the VM, the next step is configuring the VIM through the GUI. When the VIM is added to the service platform, the SONATA's Service Platform is ready to use.

Configuration of the SDK in the Demonstration Server

Once the SDK is installed on the Demonstration server, it doesn't require any additional configuration.

Startup process of the Service Platform in the Demonstration Server

The Service Platform will start automatically when finish the installation. To start the SP, just use the command "service sonata start"

Validation process of the Service Platform in the Demonstration Server

To validate that the SONATA's service platform demonstration server is running, just run on the server the command `service sonata status'`

3.4.2.2 Service Platform for the Demonstration Infrastructure

The SP for DI (SP4DI) is based on a small VM running the SP services optimised for Demonstration purposes.

The SP services available at the Demonstration Infrastructure are:

- Gatekeeper services
- Catalog repos
- MANO framework
- IFT-Abstr Adapters
- Monitory services

The NS that will be demonstrated are automatically deployed to the available NFVI-PoPs through the SP4DI as illustrated in the following picture.

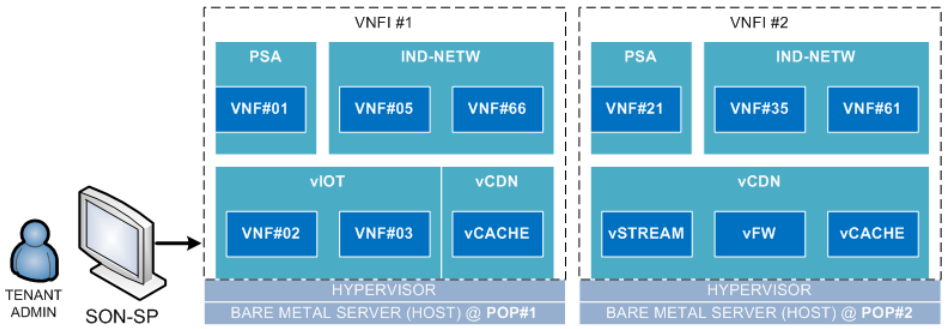

Figure 3.25: SP deployment of a example NS to DI

In this picture, an Operator (Administrator Tenant) is managing the deployment of NS to two independent VIMs/PoPs:

- 2 PoPs (VNFI #1 and VNFI #2)
- each PoP is running localised functions (VNF) and/or services (NS)
- there are services (eg, vCDN) with distributed topology that require VNFs to run at more than one PoP

SP initialisation process

In order to deploy the SP4DI, the following steps are considered:

1. get a Linux shell with Ansible Control Center
2. get "son-sp4di" YAML packages from Github
3. execute the **son-sp4di.yml** playbook
4. deploy a VNF/NS selected from the Catalog

Figure 3.26 illustrates the Demonstration Test Cycle as discussed previously.

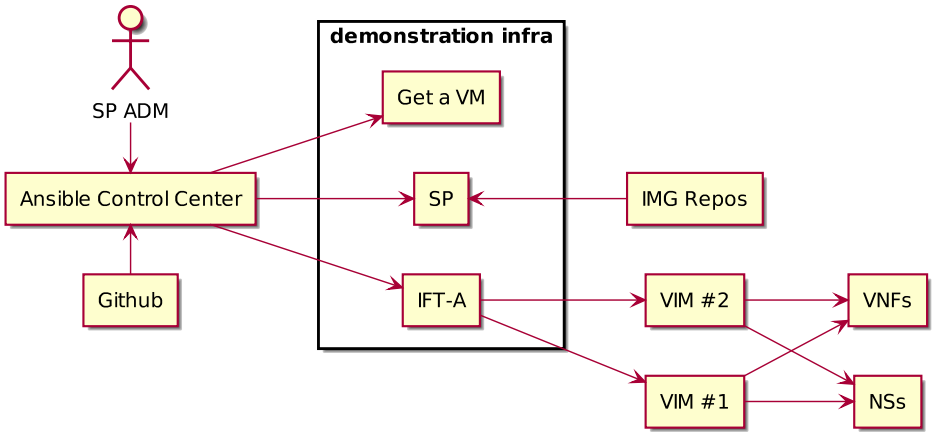

Figure 3.26: Demonstration Test Cycle

3.4.2.3 Demonstation NFVI

In Figure 3.27, the physical deployment of the Demonstration Infrastructure is illustrated. There are two available sites (i.e locations) hosting the DI. One is located at the NCSRD premises, deployed over the same underlying datacenter infrastructure as in II and QI flavours. The other NFVI-PoP is available at Altice Labs premises. As can been seen in the Figure 3.27, these two sites are interconnected through a L3VPN (IPSEC based) overlay implemented between Cisco ASA 5500 series Firewall/VPN network nodes.

3.4.2.4 SDK Demonstration Infrastructure

The SONATA SDK aims at being a light-weight collection of tools that can be run for example on a developer's laptop. Each of the SDK modules will deploy as docker container(s):

- son-cli
- son-catalogue
- son-monitor
- son-analyze
- son-emu

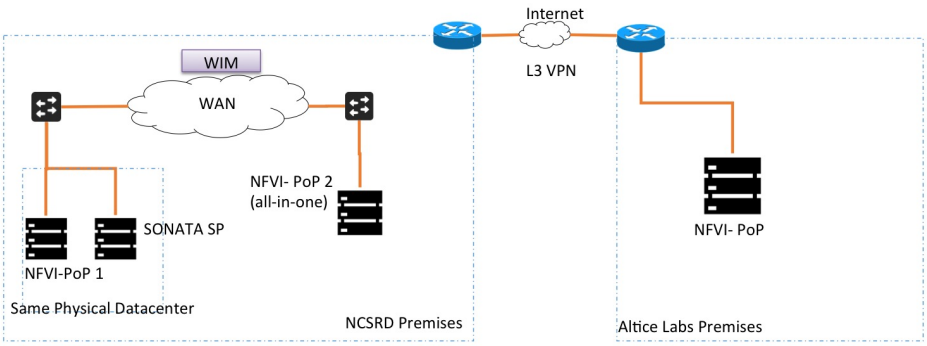

Figure 3.27: Demonstration Infrastructure View

This set of modules allows a developer to locally describe, test, debug and package a service. The SP's Gatekeeper exposes a REST-api to which the developer machine needs network access to. From son-cli, the son-push tool is the link to the SP Gatekeeper, where the service package is pushed to.

3.5 SONATA Operations

This sections describes typical Operations over the SONATA platform, namely:

- Regular maintenance tasks and activities
- Sonata services life-cycle management
- Backup policies
- Update patches
- Migration strategies
- Troubleshooting

Due to the different nature of tasks and activities when operating over the infrastructure equipment (hardware and operating system software) and the SONATA services and applications, they are presented in separate sections:

- SONATA Infrastructure equipment Operations
- SONATA Services and Applications Operations

This section refers the basic maintenance and troubleshooting operations to realise over the infrastructure environments. The monitory of hardware resources also helps to act proactively to prevent the occurrence of critical incidents.

3.5.1 Infrastructure Operations

This section covers the typical tasks and activities executed at the Infrastructure level by the Operations team, and includes:

- Network Operations Center (aka, NOC)
  - provision of VPN Remote Access service (VPN RA)
  - manage VPN site-to-site secure tunnelling connectivity (VPN L2L)
  - add/remove/update a physical network elements
- Technical Operations Center (aka, TOC)
  - add/remove/update a PoP
  - add/remove/update a VIM
  - manage Openstack (as the local VIM)
  - add/remove/update a physical server/storage elements
  - monitor resource usage (using adequate monitoring and logging tools)
- Security Operations Center (aka, SOC)
  - monitor network and system threads (using adequate monitoring and logging tools)

This available Infrastructure that serves the Integration, Qualification and Demonstration environments was shown in the previous section.

3.5.1.1 Networking Operation services

The Operation tasks and activities on networking area rely mainly on:

- VPN RA service management
- VPN L2L service management
- Sonata network space address management
- VNF/NS network address service

The network planning was presented in previous chapter of D6.1.

VPN Remote Access service

The typical VPN RA operation includes:

- manage VPN accounts
- monitor VPN service health

PoP #1
Client: Cisco AnyConnect VPN client
URL: vpn.medianetlab.gr
Username : sonata.ptin
Passwd : ********

PoP #2
Client: Cisco AnyConnect VPN client
URL: inovpn.ptinovacao.pt/SONATA
Username : usersonata
Passwd : ********

VPN site-to-site services

The typical VPN L2L operation includes:

- configure, establish and maintain a secure SSL/IPSec tunnel between SONATA platform and VNFI
- health monitor of SSL/IPSec tunnelling service

3.5.1.2 Platform Operation services

For Y1, each PoP has its own Infrastructure tools depending on the hardware manufacturer.

PoP#1 (NCSR)

- Cacti
- Nagios
- OpenNMS

PoP#2 (PTIN)

- HP System Insight Manager
- HP c7000 Onboard Administrator
- HP 3PAR Admin Console
- Cisco Prime Infrastructure

The Monitory tool should help in the following basic capacity management operations:

- file-system grow
- storage I/O
- network I/O

NOTE: the Prometheus monitory server used in SP monitory is also used for infrastructure monitoring.

3.5.1.3 Backups

For Y1 there are not a backup infrastructure allocated to SONATA. In this way, there are no official backup policy defined. However, to offer a minimum backup solution, a daily VM snapshots with one week rotation is implemented.

Worth to mention the built-in fault-tolerant storage solution, offered by he CEPH backend technology. The replication factor used by CEPH is 3.

3.5.1.4 Troubleshooting

For infrastructure troubleshooting, the previous tools should be used. The potential issues to supervision typically are:

- connectivity issues
  - verify LAN connectivity (internal datacenter connectivity)
  - verify WAN connectivity (inter-datacenter connectivity - site-to-site)
- storage issues
  - verify SAN connectivity
  - verify LAN CEPH connectivity
- computational issues
  - verify VM performance
  - analyse logging information

3.5.1.5 Migration

Along the project horizon (end of 2017) it is not expected the occurrence of hardware upgrade for the main site (NCSRD) neither for secondary site (PTIN). However, and considering that both sites are powered by Openstack, it very well acceptable the need for Openstack upgrades due to new release every 6 months, benefiting from new functionalities, stability and security.

Assuming the SP independence from Openstack, no issues are expected after Openstack upgrade.

In this context, the goal for both PoP's is to have Liberty/Mitaka release ready for September demo's.

The following software services should be considered as core infrastructure elements for the testbed:

- Jenkins server
- Integration Server 1
- Integration Server 2
- Local Docker Registry
- Monitoring Server

Openstack

At the beginning of the project, SONATA's CI/CD infrastructure was deployed on a test-bed with the following characteristics:

- Openstack kilo
  - 1 Controller node
  - 1 Storage node cinder
  - 2 Compute nodes
- Opendaylight Lithium

Openstack platform will be updated with the newest stable release after the Y1 review, in order to use the latest features of Openstack.

Jenkins

The critical VM was the Jenkins server because it is the core of the continuous integration system. The downtime should be very short in order to mitigate the impact on the project. SONATA has plans to upgrade Jenkins from version 1.638 (2015/11/11) to 2.X. The version 2 add a very useful functionality that is the pipelines and improve the User Interface.

The migration plan was to make a snapshots of the VMs, copy the snapshots to the new testbed and create the VMs keeping the local IP address. At the end of the migration, the apache service on jenkins should be stopped and the folder /var/lib/jenkins should be copied from old Jenkins to the new one. Later the Public IP address of Jenkins should be moved and all the DNS registry should be updated with the new IP address. The following list is a summary of the DNS record changes:

- Changing this records:

```
registry.sonata-nfv.eu -> 10.31.11.34
images.sonata-nfv.eu -> 10.31.11.34
sp.int.sonata-nfv.eu -> 10.31.11.29
api.int.sonata-nfv.eu -> 10.31.11.29
gui.sonata-nfv.eu -> 10.31.11.29
bss.sonata-nfv.eu -> 10.31.11.29
sdk.sonata-nfv.eu -> 10.31.11.29
sp.int2.sonata-nfv.eu -> 10.31.11.33
api2.int.sonata-nfv.eu -> 10.31.11.33
monitor.sonata-nfv.eu -> 10.31.11.35
```

- Adding this record:

```
openstack.sonata-nfv.eu -> 10.31.1.3
```

The new testbed supports more Public IP Addresses and has more disk space and resources than the old one. In the migration, some VMs were scaled as shown in the following pictures:

- Jenkins and Integration Server 1

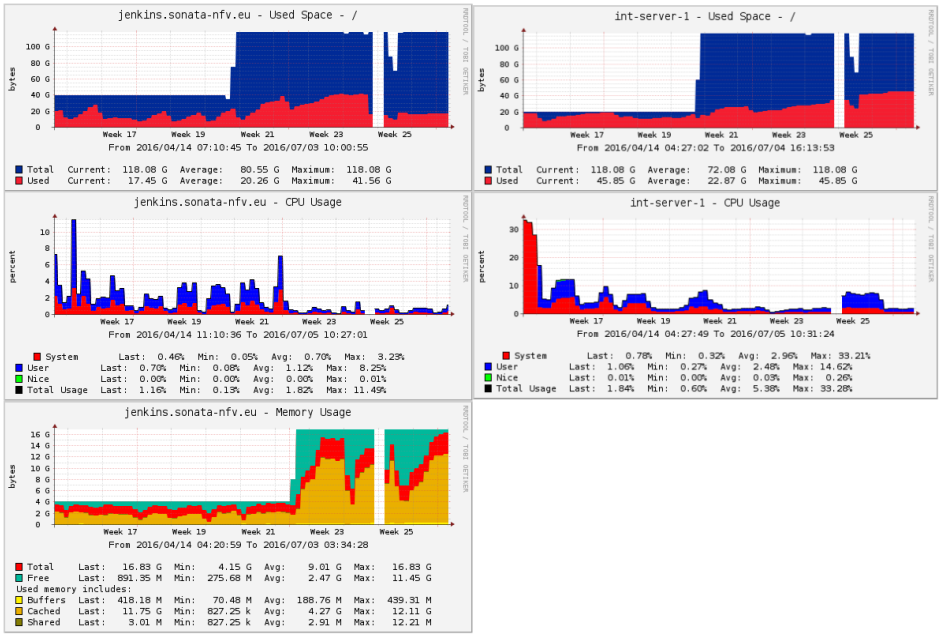

Figure 3.28: Jenkins and Integration Server Performance graphs

Docker

Docker is changing very quick and in SONATA we are updating very often to the last version of Docker.

The SONATA infrastructure has upgraded the docker version 4 times. It implies upgrade the followings VMs:

- Jenkins
- Integration server 1, 2 and 3

The steps to upgrade Docker are the following:

```
apt-get update -y
apt-get install docker
```

The current docker version is:

```
Docker version 1.11.2, build b9f10c9 (2 Jun 2016)
```

3.5.2 Operations on the Integration Infrastructure

The planned operations over the Integration Infrastructure (II) applies to the following actions:

- II platform service management
  - CI/CD platform management
  - I-SDK management
  - Integration servers management (includes SP)
  - Logging server management
  - Monitory server management

- II platform backup
- II platform upgrade
- II platform migration (eg, moving from one site to another)
- II platform troubleshooting

Currently, and for Y1 demos, the II includes +8 VM as shown in the picture.

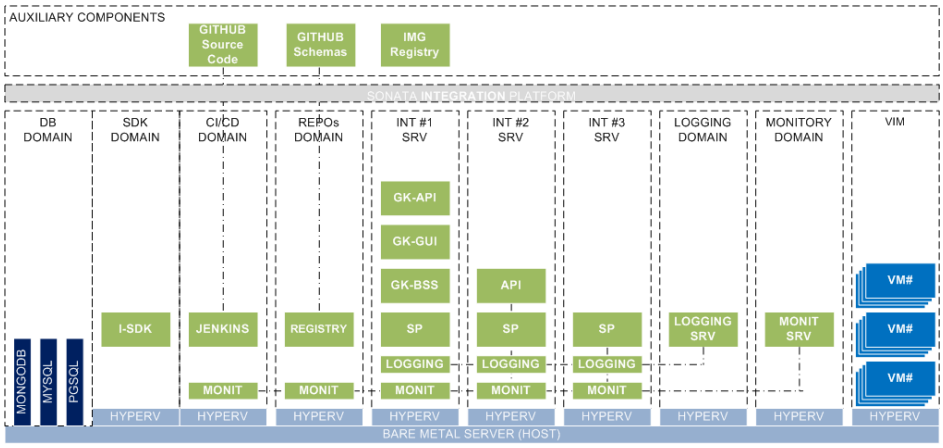

Figure 3.29: II OPS

3.5.2.1 Managing SONATA individual services at II

Managing SONATA Service Platform on the II includes the following tasks and activities:

- managing Jenkins services at VM running CI/CD engine
- managing Integration Test services at VMs running SP
- managing Integration Test services at VMs running GK
- managing Repository connectivity services at VMs running SP
- managing VIM connectivity services at VMs running SP
- managing Monitory services at VM running MON
- managing Logging services at VM running LOG

The management of SONATA services for II can be executed:

- remotely from the Ansible Control Center (aka ACC)
- inside each VM (based on local init shell script)

This sections describes the Ansible based command line interface to centrally manage SONATA services for II.

Managing Jenkins services (CI/CD engine)

`$ service jenkins [status|stop|start|restart]`

The Jenkins installation process assures the init scripts for Jenkins to start at boot time (ie, Jenkins runs as a Linux daemon):
`/etc/init.d/jenkins`

Managing SP Integration Infrastructure services

The SONATA individual services can managed by Linux shell scripts or by Ansible playbooks:

- local management

`/etc/init.d/son-sp-svc.sh` *SVC* `[status|stop|start|restart]`

- remote management

```
$ ansible-playbook son-sp-mng.yml -e "env=II,
svcid=SVC,
action=status,stop,start,restart"
```

Where the service identification (*SVC_ID*) is one in the table:

Domain	SVC	SVC_ID
GK	**GK ALL**	"'gtk-all "'
	GK SRV	gtk-srv
	GK API	gtk-api
	GK BSS	gtk-bss
	GK GUI	gtk-gui
REPOs	**Repo Catalog**	**repo-all**
MANO	**MANO FRMWRK**	**mano-all**
	MANO SLM	mano-slm
	MANO SSM	mano-ssm
	MANO FSM	mano-fsm
	MANO PLGMGR	mano-plugin-mgr
IFT-A	**IFTA ALL**	"'ifta-all "'
	IFTA-VIM	ifta-vim
	IFTA WIM	ifta-wim
LOG	**LOG ALL**	**log-all**
	LOG SRV	log-srv
	LOG ELK	log-elk
	LOG GRAYLOG	log-graylog
MON	**MON ALL**	"'mon-all "'
	MON SRV	mon-srv
	MON MGR	mon-mgr
	MON PUSH GW	mon-pushgw
ALL	**SP ALL**	**all**

3.5.2.2 Managing ALL SP services at II

The totality of SONATA services can be started up at once, in one of two ways:

- including an initialisation script at "/etc/init.d" to run as a service
- via Ansible playbooks

To start SONATA SP for II at boot time, a shell script is included at installation phase for each machine:

`/etc/init.d/son-sp-svc.sh`

To start the SP on-demand for II, the playbook is invoked with the parameters:

- local management

```
$ ansible-playbook son-sp-mng.yml -e "env=II, svcid=ALL, action=start"
```

3.5.2.3 Backup SONATA services at II

Currently (but only for Y1) no backup policy is implemented. However, a basic backup system should be implemented through VM snapshot based on hyper-visor technology. Moreover, all the SP services run inside containers. Due to the volatile nature of containers, it's a good practice to store its data in an external mount-point. It also facilitates the container life-cycle management: to execute a SP service update, just stop the old container and start the new container that mounts the same "/data" directory. This "/data" directory should be included in a backup policy managed by the VIM owner - for Y2.

3.5.2.4 Upgrade SONATA services at II

The modular SONATA architecture enables the individual treatment of each module. And for each module, individual update is envisaged, like:

- Database upgrade - For each database engine, the SONATA services that use these engines must be stopped previously

Database engine	Y1 version	Application Dependencies
PostgreSQL	9.5	son-gkeeper son-sp-infrabstract
MySQL	5.5	Prometheus
Mongo	3.2	son-catalog-repo

- I-SDK upgrade - The SDK for Integration Infrastructure

SDK products	Y1 version	Application Dependencies
son-catalogue	v1.0	DEV
son-monitor	v1.0	DEV
son-analyze	v1.0	DEV
son-emu	v1.0	DEV

- CI/CD upgrade - The CI/CD engine

CI/CD products	Y1 version	Application Dependencies
Jenkins	1.638	ALL

- INT-SRV's upgrade

INT-SRV products	Y1 version	Application Dependencies
son-catalog-repos	v1.0	VIM

INT-SRV products	Y1 version	Application Dependencies
son-gkeeper	v1.0	ALL
son-mano-framework	v1.0	ALL
son-sp-infrabstract	v1.0	VIM

- MON-SRV's upgrade

MON-SRV products	Y1 version	Application Dependencies
mon-mgr	v1.0	ALL
mon-srv	v1.0	ALL
mon-pushgw	v1.0	ALL
mon-agnt	v1.0	ALL

- LOG-SRV's upgrade

LOG-SRV products	Y1 version	Application Dependencies
logging	v1.0	ALL

To update ALL SONATA services, run the playbook (available at Github) from Linux shell with Ansible (ACC).

```
$ ansible-playbook son-sp-upgr.yml -e "env=II"
```

3.5.3 Upgrade individual SONATA services at II

The modular architecture of SONATA enables the managed of individual services in an autonomous way.

To update an individual SP service:

```
$ ansible-playbook son-sp-upgr.yml -e "env=II, svcid="
```

3.5.3.1 Migrate SONATA services at II

Currently, it is not supposed to occur a II migration services: for Y1 it corresponds to the deployment of a new II.

3.5.3.2 Troubleshooting Sonata services at II

Currentlly, the service specific log analysis should be watched:

```
// service specific logs
$ tail -f /var/log/.log
```

A set of non-invasive health-check tests is foreseen to be implemented - eg:

- communication tests among services
- VIM connectivity tests
- Repos connectivity tests

3.5.4 Operations on the Qualification Infrastructure (QI)

For Y1 demos, the QI includes 4 VM and a VNFI as shown in the picture:

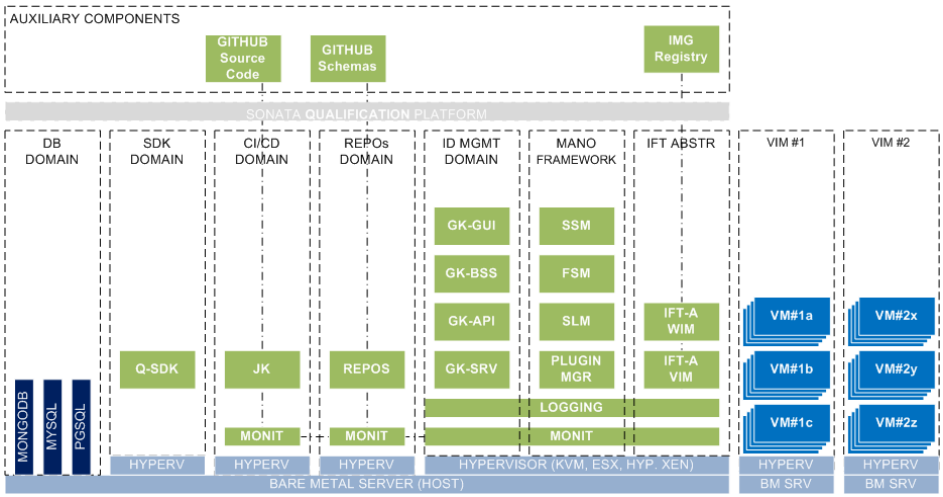

Figure 3.30: QI OPS

3.5.4.1 Manage SONATA services at QI

The management of SP services at the QI relies mainly on the following type of Operations:

- start/stop the SP of QI
- start/stop individual services
- add/remove a new VIM

The procedure to manage the SP as a whole, should be by using the available playbooks for that specific actions like start, stop, restart or inquire the service status. These commands are executed from a Linux shell with Ansible, aka, Ansible Control Center (ACC).

```
$ ansible-playbook son-sp-mng.yml -e "env=QI, svcid=, action="
```

NOTE: when "SVC_ID" = "ALL", then the selected action applies to all software components of the SP.

3.5.4.2 Backup SONATA services at QI

The backup service for the SONATA on the QI assumes the following actions:

- databases backup
- SP backup
- VNF/NS backup

Database backups

The QI makes use of 3 different database engines (pgsql, mysql, mongdb), so the backup of its data files and configuration files must be assured. The method for database backup is using their own database engine backup tools, namely:

- PostgreSQL

`$ pgdump`

- MySQL

`$ mysqldump`

- MongoDB

`$ mongodump`

For Y1, the backup policy for QI databases is simple daily execution of an 'sqldump'. For Y2, a enforcement of this policy must be specified and implemented.

SP backups

For Y1, no SP backup needs are envisaged. Nevertheless, the following backup services are envisaged:

- the Docker Image Repositories, stored at a private Registry, must be under backup policy
- the SP databases (mysql, pgsql, mongo) must be under backup policy

The following resources already have backup, due to its service nature:

- the service Descriptors and Schemas are stored at an external Repo (Github)
- the infrastructure Descriptors and Schemas are stored at an external Repo (Github)

VNF/NS backups

For Y1, the VNF/NS running at the VNFI will be under a daily snapshot backup policy. For Y2, a enforcement of this policy must be specified and implemented, namely at its locally generated data - eg, the tenant security rules at a vFW service.

Nevertheless, the original VNF/NS image files deployed to a VNFI is stored at Docker Registry.

3.5.4.3 Upgrade all SONATA services at QI

Along the SONATA development cycle, frequent delivery of new SP releases are expected. The upgrade of a single service is a complex task due to service dependencies, as well as the upgrade of the global SP.

NOTE: The playbook is responsible for the correct execution of stopping, updating and starting the SP in a controlled way.

For a full SP upgrade, the following sequence is applied:

1. stop listen ports at the firewall, ie, no more new connections are allowed
2. stop SP services in a predefined order (the opposite from start)
3. stop database services
4. backup configurations and data files
5. update individual services
6. start database services

7. start individual services in a predefined order (the opposite from stop)

8. execute health-check test to validate update

The global update of SONATA services is centrally executed at a Linux shell with Ansible by running the appropriate playbooks available at Github.
`$ ansible-playbook son-sp-upgr.yml -e “env=QI”`

3.5.4.4 Upgrade individual Sonata services at QI

Thanks to the modular architecture most of the services can be managed individually.
The following playbooks execute the update of individual SP services:
`$ ansible-playbook son-sp-upgr.yml -e “env=QI, svcid=”`

3.5.4.5 Migrate SONATA services at QI

For Y1, migration of services is not supported, but expected for Y2. Until there, service de-localization corresponds to a new deployment.

3.5.4.6 Troubleshooting SONATA services at QI

3.5.5 SP Operations on the Demonstration Infrastructure (DI)

The Operation of DI is very similar to QI but simpler: at DI, the show-case demonstrations require a permanent environment and, as so, doesn’t need to be destroyed like in QI when the Qualification tests are done.
The focus of DI is the life-cycle management of a VNF/NS, ie, the execution of typical Operations as Tenant Admin - for example:

1. deployment of a new VNF/NS to a VIM

2. update an existing VNF/NS already running on a VIM

3.5.5.1 Manage SONATA services at DI

Currently, there are several optional technologies to manage infrastructure and deploy applications in an orchestrated fashion, namely:

- Jenkins based
- Ansible based
- Openstack Heat based
- Terraform based

Here we describe the **Ansible** option to manage SP services, namely for:

1. deploy a VNF/NS to a VIM

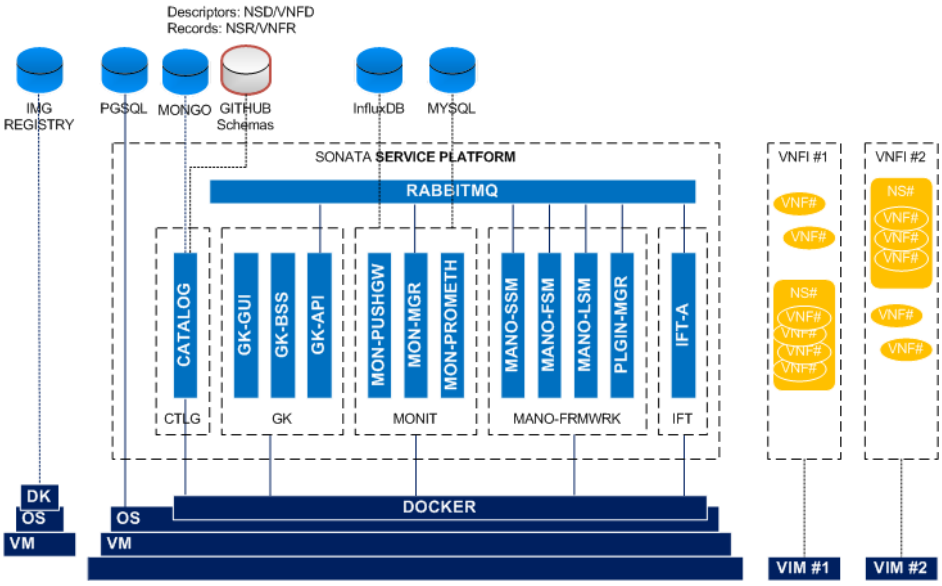

Figure 3.31: SP 4 DI

2. start/stop a VNF/NS on a VIM

3. update a VNF/NS on a VIM

NOTE: the complexity of managing dependencies when deploy/start/stop/update/remove of SP services is controlled inside the Ansible playbooks. Example for the 'upgrade' of a VNF:

1. transfer the new VNF to the target VNFI

2. stop the old VNF services

3. backup configuration files and data files from the running VNF

4. mount data files and start the new VNF services

deploying a VNF/NS to a VIM

```
$ ansible-playbook son-vnfns-dply.yml -e "vim=, vnfns=<VNF/NS_ID>"
```

manage the life-cycle of a VNF/NS on a VIM

```
$ ansible-playbook son-vnfns-mng.yml -e "vim=, vnfns=<VNF/NS_ID>, action="
```

update an existing VNF/NS on a VIM

```
$ ansible-playbook son-vnfns-upgr.yml -e "vim=, vnfns=<VNF/NS-ID>"
```

removal of an existing VNF/NS on a VIM

```
$ ansible-playbook son-vnfns-dele.yml -e "vim=, vnfns=<VNF/NS-ID>"
```

3.5.5.2 Backup SONATA services at DI

For Y1, and at the time this Deliverable is written, only basic backup solution is implemented. For Y2, more advanced solutions should be accomplished.

Currently, two levels of backup are considered:

- the baseline image of a VNF/NS is available at the Image Repository
- the VM running at the VIM should have daily snapshots implemented

3.5.5.3 Upgrade SONATA services at DI

When a new version of a VNF/NS is available at Service Catalog (currently, the Image Repository), the Tenant Admin is able to update this VNF/NS.

The steps to update a VNF/NS are:

1. login to the SP with TEN-ADM credentials
2. upload the new image (VM or Container)
3. stop the running VNF
4. start the new Image

NOTE: the complex actions are automatically executed by the Ansible playbook, which is responsible to control the start, stop and its precedence.

3.5.5.4 Migrate SONATA services at DI

For Y1, migration of services is not supported, but expected for Y2. Until there, service delocalisation corresponds to a new deployment.

Support for Migration services in Y2 can include:

- moving the SP to another Infrastructure
- moving a VNF/NS to another VIM

3.5.5.5 Troubleshooting SONATA services at DI

The actions need to analyse and fix the reported issues at DI may have different origins. Then, a layered inspection should be taken. Considering that a VNF or the whole NS can be hosted by an external Service Provider (eg, AWS), then a top-down approach should be followed (because, in most cases, the IaaS and PaaS platform is owned by the Service Provider):

1. start by analising the application layer, ie, the VNF and/or NS itself
2. finally analyse the VIM infrastructure layers

At the VIM infrastructure, the following resources should be inspected:

- LAN and WAN networking analysis
- storage and SAN connectivity analysis
- compute resources

3.5.6 SP Operations cheat sheet

This section discusses a summary of the most commons playbooks used to remotely manage: SP services

3.5.6.1 SP playbook families

```
# INSTALL SP
$ ansible-playbook son-sp-inst.yml -e list_of_parameters


# MANAGE SP
$ ansible-playbook son-sp-mng.yml -e list_of_parameters


# UPGRADE SP
$ ansible-playbook son-sp-upgr.yml -e list_of_parameters


# REMOVE SP
$ ansible-playbook son-sp-dele.yml -e list_of_parameters
```

3.5.6.2 SP playbook parameters

The 'environment' parameter gets the value of one of the possible Infrastructure types:

```
  # Environment: Integration Infrastructure
env=II


# Environment: Qualification Infrastructure
env=QI


# Environment: Demonstration Infrastructure
env=DI
```

3.5.6.3 SP Service Identification

The SP "SVC_ID" is show in table:

Domain	SVC	SVC_ID
GK	**GK ALL**	"'gtk-all "'
	GK SRV	gtk-srv
	GK API	gtk-api
	GK BSS	gtk-bss
	GK GUI	gtk-gui
MANO	**MANO FRMWRK**	**mano-all**
	MANO SLM	mano-slm
	MANO SSM	mano-ssm
	MANO FSM	mano-fsm
	MANO PLGMGR	mano-plugin-mgr
IFT-A	**IFTA ALL**	"'ifta-all "'
	IFTA-VIM	ifta-vim
	IFTA WIM	ifta-wim

Domain	SVC	SVC_ID
LOG	**LOG ALL**	**log-all**
	LOG SRV	log-srv
	LOG ELK	log-elk
	LOG GRAYLOG	log-graylog
MON	**MON ALL**	"'mon-all "'
	MON SRV	mon-srv
	MON MGR	mon-mgr
	MON PUSH GW	mon-pushgw
ALL	**SP ALL**	**all**

3.5.6.4 SP management

SP4II

```
# SP Initialization scripts for Integration Infrastructure
$ ansible-playbook son-sp-mng.yml -e "env=II,\
svcid=SVC_ID, action=status|stop|start|restart"
```

SP4QI

```
# SP Initialization scripts for Qualification Infrastructure
$ ansible-playbook son-sp-mng.yml -e "env=QI,\
svcid=SVC_ID, action=status|stop|start|restart"
```

SP4DI

```
# SP Initialization scripts for Demonstration Infrastructure
$ ansible-playbook son-sp-mng.yml -e "env=DI, \
svcid=SVC_ID, action=status|stop|start|restart"
```

3.5.7 VNF/NS Operations cheat sheet

3.5.7.1 VNF/NS playbook families

```
# DEPLOY VNF/NS
$ ansible-playbook son-vnfns-dply.yml -e list_of_parameters


# MANAGE VNF/NS
$ ansible-playbook son-vnfns-mng.yml -e list_of_parameters


# UPGRADE VNF/NS
$ ansible-playbook son-vnfns-upgr.yml -e list_of_parameters


# REMOVE VNF/NS
$ ansible-playbook son-vnfns-dele.yml -e list_of_parameters
```

3.5.7.2 VNF/NS parameters

```
# VIM
vim=VIM_ID
```

```
# VNF/NS
vnfns=VNF_ID or NS_ID
```

3.5.7.3 VNF/NS deployment

```
# DEPLOY
$ ansible-playbook son-vnfns-dply.yml -e "vim=VIM_ID, vnfns=VNF/NS_ID"
```

3.5.7.4 VNF/NS management

```
# MANAGE
$ ansible-playbook son-vnfns-mng.yml -e "env=VIM_ID,\
vnfns=VNF/NS_ID, action=status|stop|start|restart"
```

3.5.7.5 VNF/NS upgrade

```
# UPGRADE
$ ansible-playbook son-vnfns-upgr.yml -e "vim=VIM_ID, vnfns=VNF/NS_ID"
```

3.5.7.6 VNF/NS removal

```
# REMOVE
$ ansible-playbook son-vnfns-dele.yml -e "vim=VIM_ID, vnfns=VNF/NS_ID"
```

4 SONATA Pilots

For first phase (i.e. Y1), the integrated SONATA pilots will be specified. These will be based on the "Demonstration" infrastructure (c.f Section 3.4) which will be deployed across multiple pilot sites and on which the SONATA components will be installed. The specification phase includes:

- the identification of the pilot architecture;
- the specification of all system modules with fall-back solutions if necessary;
- the description of actual equipment to be used, as well as the procedure for installing the SONATA components.

The Pilots will be based on further elaboration on the UCs including the required VNFs implemented by WP5, Task T5.2

4.1 Use Case Summary

During the first year of the project a number of Use Cases have been defined that allowed the project to elicit requirements regarding SONATA framework and progress with the preliminary implementation and the definition of a project wide Continuous Development/Continuous Integration work-flow. This work-flow is intended to be used not only at the development of SONATA components but also are proposed as a paradigm for SONATA adopters. In order the document to be self-contained a brief summary of the UCs is included below. The full description of the UCs has been contributed in Deliverable D2.1 [12].

4.1.1 UC1: vCDN

This section elaborates on the initial description of the virtual content delivery network (vCDN) use case.

4.1.1.1 Description

This use case focuses on showcasing and assessing the SONATA system capabilities in order to enhance a virtual CDN service (vCDN) with elasticity and programmability. Customers wishing to play the role of the vCDN operator use the system to dimension the service with regard to (IT and network) resources use and deploy vCaches as VNFs reacting to network/user triggering events coming through the monitoring system. Using the feedback from the service metrics (i.e. traffic load, packet drops, content type etc), the vCDN service can be automatically re-configured according to policies set by the customer (vCDN operator). Two different scenarios have been identified for this UC:

- Scenario 1: Distribution of highly popular content, where the content originated from a content provider, distributed across the vCaches and eventually delivered to a huge number of subscribers.

- Scenario 2: Edge caching and adaptation of user-generated content, aiming at supporting the increased load associated with user-generated content (photos, audio and video) in e.g. a flash crowd event (for example hundreds of subscribers sharing high-definition feeds from a live show).

4.1.1.2 Stakeholders revision

The initial considered stakeholders of the vCDN UC are:

1. the end users of a service (e.g., a private person) = content consumer
2. the developer of a software artefact (e.g., an NFV or a composition of NFVs into a service)= content provider = vCDN operator
3. the service platform operator (SONATA service provider), who runs a platform that manages the execution of services, and
4. the actual infrastructure operator, who often, but not necessarily, will be the same as the service platform operator

The revision of the above stakeholders after the first year assumes that:

1. Content consumers are the end-users of the actual service
2. Platform Operator/Infrastructure Owner deploy the SONATA Service Platform over his infrastructure and offer to their customers slicing, isolation, programmability features in order to develop and deploy their services
3. Customer is the VNO or the Content Provider that through SONATA capabilities is able to develop, deploy and operate their services.

4.1.1.3 Components revision

The list below is a revised view of the UC components

Software components

- vDPI: VNF for classification and monitoring of the video streams. The information from this VNF will be consumed by SSM in order to make scaling and placement decisions for the rest NS included VNFs (i.e vCache)
- vCache: VNF for content cashing at the edge of the network topology. It's functionality could be minimal (i.e. a content proxy where requests could be redirected)
- vCPE: End User equipment capable of being redirecting EU requests to the closest or most appropriate vCache. Could be omitted
- vCDN EMS decomposed to SSM plugins

Hardware components

- None besides the NFVI PoPs required to deploy the VNFs.

Auxiliary components (used for the demonstration purposes of the UC)

- Video Server (root)
- Clients (several)
- Traffic Generator (in case of simulation of several network conditions/load etc)
- Latency
- Throughput
- Availability
- Cache hit ratio (vCache related metrics)

4.1.2 UC2: IoT

This section elaborates on the initial description of the optimised IoT use case.

4.1.3 Description

This use case aims to show SONATA's ability to monitor, classify, and optimise IoT network traffic. A set of classification VNFs will be deployed at the edge of the network to identify each type of IoT flow. Flows will be classified and redirected to an IoT gateway at a local NFVI-PoP, where an appropriate VNF will perform optimisations specific to the type of flow (like transcoding media, or batching requests). Finally the gateway will pass the optimised traffic to the target end system, both reducing the network congestion as well as the load on the central components. The SONATA Service Platform will be able to deploy new classifiers where they are needed, scale, and appropriately locate each kind of IoT Gateway to ensure compliance with SLAs, as well as upgrade components with zero downtime to the service.

4.1.3.1 Stakeholders revision

- ISP that provides the communication service to the IoT sensor network will use the SONATA platform deployed in its NFVI.
- The detection and forwarding of IoT traffic is detected at the entry point of the ISP network infrastructure, i.e. at the access point.

4.1.3.2 Components revision

Software Components

- Traffic detection: VNF for monitoring the traffic between the sensor and the network. This VNF also depends on the sensor type.
- IoT Gateway: VNF responsible to digest the data produced by sensors. This component receives sensors' data from access points and performs optimisations according to the type of flow, such as batching and transformation of data. Following this procedure, it forwards the information to the correct IoT Datacenter.
- Control centre: On top of all the described components, a control centre VNF is responsible to trigger an elastic deployment of additional VNFs, such as access points and IoT gateways.

Hardware Components

- Access Point: VNF technologically capable of communicating with one or more sensor types. As the communication technology of sensors may differ, several types of VNF may be deployed, each able to communicate with a specific sensor type. It is responsibility of the access point to classify and forward the sensor traffic to the correct IoT gateway. Furthermore, it can also trigger the deployment of new VNFs if a new sensor type is detected.
- IoT Datacenter: IoT aware Datacenters can provision resources according to the request load.

4.1.3.3 Validation (non-functional)

The non-functional aspects validation of this UC will include:

4.1.4 UC3: vEPC

This section elaborates on the initial description of the virtual EPC use case.

4.1.4.1 Description

This use case focuses on showcasing and assessing the SONATA system capabilities in order to enhance a virtual EPC service with elasticity, placement optimisation, resilience and DevOps methods.The use case considers the case of an MVNO wishing to operate their vEPC as a network slice on top of a virtualised infrastructure provided by infrastructure provider in order to gain more flexibility to continuously dimensioning the service, function of the end-users activity, to optimise placement with regards to the users' locations, to increase resilience by optimally adapting the system to failure or security attack, and to bring new maintenance work-flow using DevOps principles.

4.1.4.2 Stakeholders revision

Initial view of the stakeholders

1. The vEPC vendor
2. The vEPC service provider (SP)
3. The NFVI provider
4. Mobile device users

Revised view of the UC stakeholders:

1. A Network Operator/Infrastructure Operator deploys the SONATA Service Platform and proposes a vEPC as a network slice or as a service
2. Customer: Mobile Virtual Network Operator (MVNO) that operates the vEPC and exploits a provisioned slice
3. End-Users: user's terminals (mobile devices) connected to the vEPC

4.1.4.3 Components revision

Revised view of the UC components **Software Components**

- vHSS: VNF with the Home Subscriber Server, a database containing user-related information
- vMME: VNF with the Mobility Management Entity controlling the UE's (User Equipment) access
- vS-GW: VNF containing the Serving Gateway for routing and forwarding user data packets
- vP-GW: VNF containing the PDN Gateway for connecting UE to external networks
- vPCRF: VNF with the Policy and Charging Rules function which supervises and monetise data flow in the LTE network

Hardware Components

- RAN: Radio Access Network with some eNode-B and user equipment (if vEPC software is provided)

Auxiliary components

- Displaying a full vEPC experience requires additional hardware: antennas, mobiles.
- No partner was able to provide an industrial-grade level EPC. A demonstration using dummy vEPC components would be unable to show any mobile services. The free EPC Open Air Interface[1] software is still not mature. The components are not flexible enough to match the UC vEPC's architecture (for example, the two gateways are merged together). Additionally, no partner has any experience with Open Air to provide support.

4.1.4.4 Requirements

- No partner provides a vEPC software. In consequence, the use case can demonstrate some feature using empty box with vEPC label, but cannot show a real mobile service.
- If a partner vEPC software, the demo will require to setup a Radio Access Network with some eNode-B and user equipments to show the end user service.
- Connectivity among all the involved components, End User devices, Servers in a Multi-PoP environment

4.1.4.5 Demonstrated SONATA Features

- Scalability of vEPC component
- Resilience of vEPC component
- Placement of vEPC component to move close the the end-users
- Adaptation of the Service Graph to change the QoS of end-users

[1]Open Air Interface: `http://www.openairinterface.org/`

4.1.5 UC4: iNets

This section elaborates on the initial description of the iNets use case.

4.1.5.1 Description

A wind park operator uses SONATA to define a service that collects measurement data of the wind turbines in a local wind park. The data is forwarded to a central server via the local communications network. At the server the data is processed in real-time and commands are sent back to the wind turbines adapting certain turbine control parameters. An additional tenant service is defined over the local wind park communications network. For the sake of software maintenance and upgrade, it provides access to a select set of control modules in selected wind turbines to an external company. The turbine control service must not be impacted negatively by the additional service. At any point of time, the wind park operator wants to be able to see what has been accessed when by which tenant, i.e. have a complete communications trail about all tenants and applications in the wind park network.

4.1.5.2 Stakeholders revision

The initial considered stakeholders of the iNets UC are:

- 3rd party turbine SW providers: supplies software modules (and currently also hardware) for controlling the operations of the wind turbine
- wind park operator: company that runs the wind turbines, the communication network in between (even though that may be partially leased) and the control centres with SCADA and other server software
- potentially network provider (for now, we assume self-owned network infrastructure)

The revised view on the stakeholders, has not changed. The network provider in this UC that will deploy the SONATA SP is the park operator himself. The SONATA SP functionalities are exploited for his own needs.

4.1.5.3 Components revision

Software Components

- SCADA servers
- AAA server
- turbine control SW (sensors, actuators)
- network management system (SDN controller)
- analytics functions

Hardware Components

- Turbines or a simulated turbine logic

4.1.5.4 Validation (non-functional)

The non-functional aspects validation of this UC will include:

- Latency: The entire round trip time needs to be kept below a fixed delay threshold and resilience mechanisms must be in place to ensure message delivery with very high probability.
- Role based access (RBAC)
- Multi-tenancy: allow multiple tenants over the wind park network.

4.1.6 UC5: PSA

This section elaborates on the initial description of the Personal Security Application use case.

4.1.6.1 Description

ISP providers create and offer a security service catalog for their customers. Users aware of security risk when using the network can search the catalog and decide to subscribe to a security service. The Customer selects the preferred business model the service provider offers: a subscription based on time of use, volume of data, a flat rate, etc.. BSS channels triggers based on user identify on the network the orchestration, deployment and monitoring process of a set of vNSFs that combines the services required by the user. The Customer may decide to change his/her service's configuration (by applying some additional incoming filters, for example) and SONATA reconfigures the service accordingly. For example, this could result in adding and deploying a new vNSF with Parental control and the list of blocked webpages as part of the configuration.

4.1.6.2 Stakeholders revision

The initial view of the stakeholders is:

- ISP Subscribers. The user is a customer of an ISP with a personal vNSF demand. The role of this actors is basically demand his own security services.
- Network Operator or ISP. The owner of the network or with the capacity to deliver the security Services based on user demands or own criteria. Usually DevOps units in ISP use the vNSF monitoring security reports for automatically triggering of actions when the Operation unit requires the action.
- vNSF Developer. A vendor with products that creates Security network oriented VNF. Their role is focused in sell their security products from their external catalog adapted to the ISP and user needs.

No revision is required to the above list of stakeholders.

4.1.6.3 Components revision

Software Components

- vPSA: Personal Security application VNF
- Load Balancer

- PSA management SSM / FSM

Hardware Components

- Server nodes: COTS servers with virtualisation infrastructures where different personal security services are deployed.
- Network nodes: ISP works usually with dedicated nodes combined with virtual network nodes.

4.1.6.4 Validation (non-functional)

The non-functional aspects validation of this UC will include:

- scalability
- monitoring
- time to deployment

4.1.7 UC5: twoSPS

This section elaborates on the initial description of the two SPs use case.

4.1.7.1 Description

This use case describes the situation where there is more than one service provider involved in the establishment of an NFV network service and where both are running independent instances of SONATA. In particular, the case is considered where one service provider is responsible for the VNFs and their composition into a network service while a different service provider is providing the NFV infrastructure on which the VNFs are hosted. The model is a 'client/server' model between service providers rather than a 'peer-peer' model.

4.1.7.2 Stakeholders revision

The initial view of the stakeholders is:

- SP with SONATA Service Platform deployed
- SP with SONATA Service Platform deployed, acting as a customer for the first SP.

No revision is required to the above list of stakeholders.

4.1.7.3 Components revision

No internal to the SONATA framework interactions are considered. The interaction between two instances of the SONATA system treated as a whole rather than specific interactions between components of the SONATA system. The main requirement captured by this use case relates to the business model rather than the detailed architecture of the SONATA system. The use case does not therefore detail specific functional interactions between the two instances of the use case; such details are appropriate for a later stage in the design process.

4.1.7.4 Validation (non-functional)

The non-functional aspects validation of this UC will include:

- response time to requests for information about capabilities
- response time to requests for new service designs
- response time to to requests for new service instances
- scalability

4.2 Pilot selection methodology

The implementations of numerous SONATA components are still in progress, the full visibility on the Pilots that may be realised is still not possible. This section elaborates on the definition of certain criteria against which the initially defined use cases will be checked. This process will provide insight into SONATA functionalities that are not represented or non-functional requirements that are not included in the initial use case scenarios. In the case that the SONATA functionalities are not covered fully by the use case based pilots, new pilots of focused PoCs will be defined. The above strategy will ensure the full coverage of the SONATA pilots and allow optimal demonstration of the added value functionalities.

The aforementioned categorisation criteria are:

Project KPIs: Check against the main KPIs of the project as laid out in the SONATA DoW.

Project Objectives: Check against the main project objectives as laid out in the SONATA DoW.

SONATA functionalities coverage: Check against the coverage of SONATA functionalities and identify gaps

Workload characteristics variety: Check against the workload characteristics of main VNFs embedded in the use case scenario.

Feasibility: Check against the availability of UC components and the consortium experience in supporting the scenario described.

The following sections elaborate on the above criteria and conclude on the applicability (at this phase of the project) of the UC towards the SONATA pilots.

4.2.1 Project KPIs

Table 4.1: SONATA KPIs

SONATA KPIs	UC1: vCDN	UC2: IoT	UC3: vEPC	UC4: iNets	UC5: PSA	UC6: 2SPs
Resource usage	x	x	x	o	x	-
Service performance	x	x	x	o	x	-
SLA fulfilment	x	x	x	o	x	x
Service mapping	x	x	x	x	x	-
Energy consumption	o	x	o	o	o	-
Service Setup Time	x	x	x	x	x	x
Generated code efficiency	x	x	o	x	x	-
Scalability	x	x	x	o	x	-

(x) *denotes: is part of the UC*; (o) *denotes: is not part of the UC*; (-) *denotes is not applicable*; (?) *denotes not decided yet if part of the UC*

Table 4.1 attempts an overview of the technical KPIs coverage versus the UCs.

4.2.2 Project Objectives

The table below attempts an overview of the Objectives coverage versus the UCs.

Table 4.2: SONATA Objectives

SONATA Objective	UC1: vCDN	UC2: IoT	UC3: vEPC	UC4: iNets	UC5: PSA	UC6: 2SPs
Reduce time to market for networked services by shortening service development	x	x	x	x	x	-
Optimising resource utilisation and reduce cost of service deployment and operation	x	x	x	o	x	-
Accelerate the adoption of software networks in industry	-	-	-	-	-	o
Promotion and standardisation of project results	-	-	-	-	-	-

(x) *denotes: is part of the UC*; (o) *denotes: is not part of the UC*; (-) *denotes is not applicable*; (?) *denotes not decided yet if part of the UC*

4.2.3 SONATA functionalities coverage

The table below presents the SONATA functionalities as introduced by Workpackages 3, 4, 5 versus the coverage achieved by the original Use Cases as defined in WP2.

Table 4.3: SONATA Functionalities

SONATA Functionality	UC1: vCDN	UC2: IoT	UC3: vEPC	UC4: iNets	UC5: PSA	UC6: 2SPs
VNF development	x	o	o	o	o	o
SSM development	x	-	x	-	-	-
FSM development	x	-	x	-	-	x
Package (VNF/NS) On-boarding	x	-	-	-	-	x
Network Slicing	?	o	o	x	?	x
Service Function Chaining	x	x	x	x	x	x
VNF Placement (initialisation)	x	x	x	x	x	x
VNF Placement (scaling)	x	x	x	o	o	-
Network scaling	x	o	x	-	o	-
single-PoP deployment	x	x	x	x	o	x
multi-PoP deployment	x	x	x	x	x	x
monitoring	x	x	x	x	x	x
NS/VNF updating	?	?	?	?	?	-
NS/VNF upgrade	?	?	?	?	?	-
multi (heterogeneous) VIM	x	-	-	-	o	x
multi (heterogeneous) WIM	x	-	-	-	o	x

(x) *denotes: is part of the UC*; (o) *denotes: is not part of the UC*; (-) *denotes is not applicable*; (?) *denotes not decided yet if part of the UC*

4.2.4 Workload characteristics variability

The following figure presents a brief mapping of the VNFs to the taxonomy as provided by ETSI NFV ISG at document [25]. The above table provides a birds-eye view on the workloads expected by the realisation of each UC. We extrapolate the result for each VNF that participated in the

UC and we categorise the whole UC (actually the NS) to the taxonomy that had been defined in [25]. The categorisation reveals that the vCDN use case cover most of the categories defined in this taxonomy. The only class not represented in the vCDN is the cryptography and the signal processing. Moreover the storage usage is intensive since the existence of vCache imposes need for storage facilities at the PoPs. Encryption, which might require some hw assist by specific platform capabilities (EPA), is covered by UC3 and UC4, however for UC3 it can be the case where the IPSEC tunnelling required could be offered by Physical Network Functions (PNFs) which fall outside of the focus of this project. In UC4 the encryption is used for realisation of personalised VNP solutions and could be interesting to address this particular use case in the pilots. Signal processing is used in UC2 IoT, through the use of transcoding/transrating VNFs. For this workload, the approach is to use the GPU on-board the server chassis in order to accelerate the processing.

Use Cases	Data Plane			Control Plane					Storage	
	Edge VNF	Intermediate NFs	Intermediate NF supp Encryption	Routing	Authentication	Session Management	Scaling/Placement	Signal Processing	Non-intensive	R/W Intensive
UC1:vCDN	X	X	-	X	X	X	X	-	-	X
UC2:IoT	X	X	-	X	X	X	X	X	X	-
UC3:vEPC	X	X	X	-	X	X	X	-	X	-
UC4:iNets	X	X	-	-	X	X	X	-	X	-
UC5:PSA	X	X	X	-	X	X	X	-	X	-
UC6:2SPs	-	-	-	-	-	-	-	-	-	-

Figure 4.1: Workloading Map

4.2.5 Feasibility of pilot implementation

This subsection addresses the ability (at the current state) for the project to transform the UCs to pilots. The realisation of some of the aforementioned UC spawn beyond the scope of the project and the experience this consortium sustains. The conception of the UCs early in the life of the project served ultimately the goal of requirement elicitation, rather than intention to support at the moment the UCs under discussion. In summary the plausibility of implementation for each US is:

- UC1:vCDN, this UC components and requirements are feasible and the consortium can provide efficient support for it's implementation.
- UC2:IoT, this UC component require IoT infrastructure deployed and operational, as well as some of the control plane components to be realised as deployable SSM and FSM plugins. Due to the complexity of such control and management functionalities the implementation as part of the project activities will not be realistic.
- UC3:vEPC, this UC assumes beside the existence of an actual 4G physical network, virtualised components of an EPC system. At the moment, although this UC could provide an scenario

close to the 5G vision (by including a mobile network), no partner was able to provide an industrial-grade level EPC. A demonstration using dummy vEPC components would be unable to show any mobile services. The free EPC Open Air Interface 1 software is still not mature. The components are not flexible enough to match the UC vEPC's architecture (for example, the two gateways are merged together). Additionally, no partner has any experience with Open Air to provide support.

- UC4:iNetsm this UC would require actual collaboration of SONATA with VIRTUWIND project beside architecture alignment. At the moment this UC is under investigation, considering future discussions between the two projects.
- UC5:PSA the components that are available for this UC, supported by TID are not easily ported to SONATA Service Platform. Currently the UC has been realised in the frame of another EU funded project and is currently running as part of a testbed using proprietary orchestration for the deployment. In addition licensing restrictions would make impossible for some components to be reused. As the PSA UC is interesting for this project, the final decision on providing a pilot around it will be taken within the next months of the project.
- UC6:2SPs, this UC is mostly oriented in the business aspects, the internal components are irrelevant. This is not a UC that will be piloted. The requirements imposed by this UC have already been considered in the architecture definition.

4.2.6 Conclusion

Taking into account the analysis of the UC, most significant criterion is the feasibility of turning a UC to a pilot. Second of importance among the criteria is the ability of a UC to demonstrate most if not all of the functionalities of the project. Next in sequence would be the mapping of KPIs and Objectives of the project. Last but not least is to consider in our decision the timeframe of the project. For some of the UC however their significance or their added-value it is impossible taking into account the current state to accept them as pilots.

4.2.6.1 Preliminary pilots selection

Taking into account the above, the UC1:vCDN is currently the only UC that is definitely going to be piloted. vCDN selection is justified because of the broad usage of functionalities and facilities:

```
 ** includes several VNF aggregated in one NS, with a variety of workloads.
** is geo-localised (need multi-site)
```

In addition it can demonstrate a great number of SONATA functionalities, and covers enough non-functional requirement that will be investigated during evaluations.

4.2.6.2 Focused pilots (demos)

In addition to the above UC certain features of SONATA that are not covered by the above Pilot will be showcased in selected focused pilots, that will be elaborated in the course of the project. Contemplating the above categorisation, the following pilots (demos) look interesting and will be discussed in future deliverables:

- Automatic deployment of the SONATA Service Platform and SDK
- VNF/NS Updating and Upgrading

- Heterogenous VIM support
- Network Slicing and multi-tenancy
- Recursiveness

4.3 Pilot infrastructure description

The PoC for Y1 will run in 2 PoPs, to show the multi-site capability running the same type VIM (Openstack) The infrastructure scenario with 2 PoPs is illustrated in Figure 4.2:

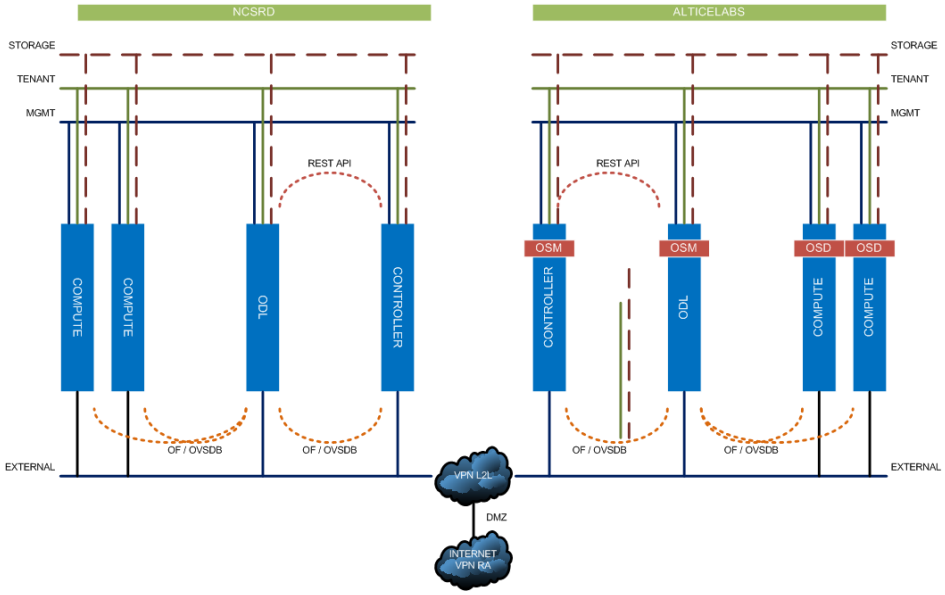

Figure 4.2: 2 PoPs Infra

- PoP#1 - based on infrastructure hosted by NCSRD for running a standalone Service Platform (SP) and the VNFI to run the demo use cases on a local site
- PoP#2 - based on infrastructure hosted by PTIN for running a second VNFI on a remote site

4.3.1 Infrastructure for pilots - NCSRD Site

This page presents the available infrastructure at NCSRD premises and used for PoC purposes. This site hosts besides the II and QI (multi-PoP) one NFVI-PoP that will be used for the Demonstrators as well as the whole DevOps framework (i.e. SDK, SP) of SONATA. This section describes the computational, networking and storage resources allocated to the pilots occurring by the end of the first year of the project life cycle (Y1).

4.3.1.1 Computational Resources

The servers hardware are consisted from the following elements:

- One Dell R210 used as the Fuel jump host

  - 1 x Intel(R) Xeon(R) CPU X3430 @ 2.40GHz
  - 4GB RAM
  - 1TB HDD

- One Dell T5500 functioning as the Controller
  - 2 x Intel(R) Xeon(R) CPU X5550@2.67GHz
  - 16GB RAM
  - 1.8GB HDD

- Three Dell R610 utilized as Compute Nodes
  - 2 x Intel(R) Xeon(R) CPU E5620@2.40GHz
  - 64GB RAM
  - 1.8GB HDD

- One more Dell R310 used as NFVI-PoP
  - 1 x Intel(R) Xeon(R) CPU X3450@2.67GHz
  - 16GB RAM
  - 465GB HDD

4.3.1.2 Networking resources

The networking resources allocated for the NCSRD PoP Infrastructure are:

- A Dell PowerConnect 5524 Gigabit switch used for the storage network
- A Digi MIL-S3160 for PXE
- A Cisco SPS2024 Gigabit switch for Management,Public,Data networks

The routing, firewall and access server equipment is based on a Cisco ASA 5510 series.

4.3.1.3 Storage resources

As storage for the NCSRD pilot site, the Ceph plugin of OpenStack is being utilised, that is a scalable storage solution that replicates data across nodes. Ceph is being used along all Controller Nodes, each with 1.8TB HDD, with a replication factor 3. This means that all the agents of Ceph have the same data stored on their disks so that an automatic migration is possible.

4.3.1.4 NCSRD site

In the following Figure 4.3 the physical view of the deployed hosting Infrastructure of SONATA can be observed. Currently this deployment corresponds to the Qualification Infrastructure. It comprises of the Service Platform, some Quality Assurance tools (as iperf, D-ITG etc), two NFVI-PoP (single node all-in-one) and a WAN that interconnects the PoPs and the Access Networks hosting the End-User and an Application Server. The SP is deployed over separate hosting infrastructure as described previously in Section 3.1.

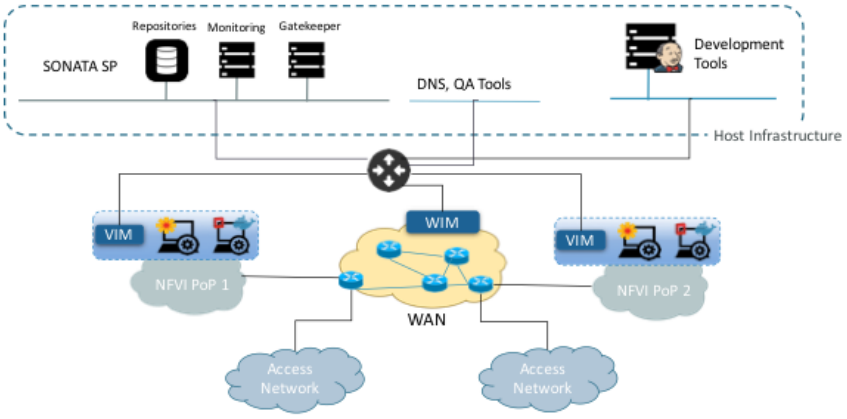

Figure 4.3: SONATA NCSRD Infrastructure Topology

4.3.1.5 Access to NCSRD Infrastructure

In order to access the NCSRD hosted Infrastructure, AnyConnect VPN or OpenConnect VPN is needed to be used. Partners in order to access the Infrastructure, must point their VPN client towards "`https://vpn.medianetlab.gr`" and log-in with the provided credentials.

4.3.2 Infrastructure for pilots - Altice Labs site

This page presents the available infrastructure at Altice Labs premises and used for PoC purposes. This site will host the second PoP in the demonstrations events. This section describes the computational, networking and storage resources allocated to the pilots occurring by the end of the first year of the project life cycle (Y1).

4.3.2.1 Computanional Resources

The server hardware is based on HPE BladeSystems converged infrastructure with the following elements:

- 1 HPE c7000 BladeSystem Enclosure
- 8 HPE Proliant Blade server 460c G8
  - 2x Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz (8 Cores)
  - 128GB
  - 2x 300G SAS RAID-1 disks
- redundant interconnects Ethernet switches: VirtualConnect (VC) Flex-10
- redundant SAN switches: 8Gbps FCP
- admin console Onboard Administrator (OA)

4.3.2.2 Networking resources

The networking resources allocated to the Altice Labs PoP infrastructure is:

- 2 redundant 1Gbps network switches for LAN connectivity and a corporate VLAN segment in the "10.112.xx.0/24 address space.
- 1 Access Server as VPN concentrator for Remote Access service (RA) and L2 secure tunnelling connectivity (L2L)

The switching equipment relies on 2 redundant Cisco 2960 Ethernet switches to connect the SONATA network traffic to the datacenter network.

Altice Labs datacenter network relies on a hierarchical topology with:

- 2 redundant central switches Cisco 4500 series
- 10 Gbps distribution switches Cisco 3750 series
- 1 Gbps edge switches Cisco 2960 series

The routing, firewall and access server equipment is based on a Cisco ASA 5500 series.

4.3.2.3 Storage resources

For Y1, the allocated storage resources rely on a HP 3Par 7200 series

- 2 redundant controllers
- 1 shelf with 12x 300GB SAS disk
- admin console

4.3.2.4 Altice Labs site

The Altice Labs' platform is running Openstack as the Infrastructure Manager, aka VIM. A graphical view is shown in the pictures:

Physical view

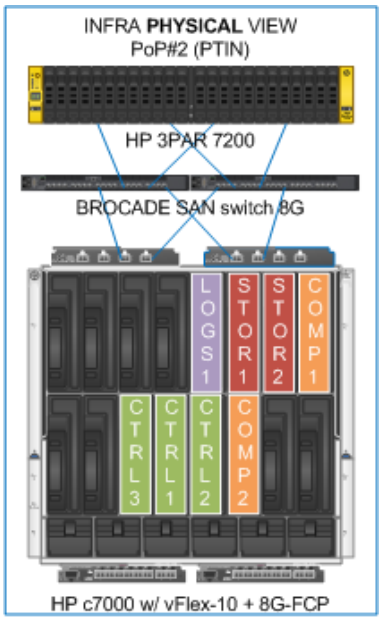

Figure 4.4: PTIN Infra PHY view

Logical view

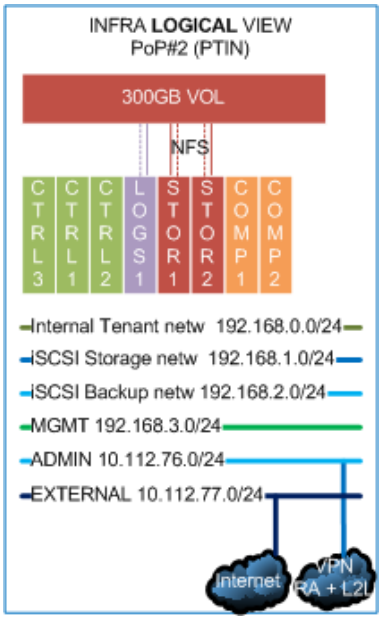

Figure 4.5: PTIN Infra logical view

4.3.3 Infrastructure for pilots - Nokia Site

This page presents the available infrastructure at Nokia IL (CloudBand) premises that could be used for SONATA during the second year of the project for PoCs and demonstrators. This section describes the available computational, networking and storage resources for this purpose.

4.3.3.1 Computanional Resources

The server hardware is based on a HP ProLiant SL390s Generation 7 servers [22] with the following specification:

- 5 computes (servers) - HP ProLiant SL390s G7
- CPU - 60 (5 computes, each 2cpu X6 Cores of 2.67 MHZ)
- RAM - 96GB
- Storage - 20TB

4.3.3.2 Networking

External access - The site can be connected to the internet via a 10G uplink and can be accessed via OpenVPN

4.3.3.3 Storage resources

The storage is managed by Ceph plugin of OpenStack. Ceph is a scalable storage solution that replicates data across nodes [3]

4.3.3.4 Nokia site

The Nokia infrastructure is running Openstack as the Infrastructure Manager [32]. Other relevant specifications:

- Configuration Management - Puppet [34]
- Virtualisation Technology - KVM [2]

Section 4.3.3.4 Is a sketch of the infrastructure that will be available at Nokia at year 2

4.3.4 Infrastructure for pilots - UCL site

This page presents the available infrastructure at UCL premises, used for PoC purposes. This site will host a NFVi-PoP used for future works on multi-VIM/WIM support and slicing. This section describes the computational and storage resources allocated to the pilots occurring by the end of the first year of the project life cycle (Y1).

4.3.4.1 Computational and Storage resources

- 1 Dell R730 servers with 1 Intel Xeon E5-2680 v3 (12 cores) running at 2.5GHz, with 192 GB memory, and 1 X 600 GB disc.
- 2 servers with 2 Quad-Core AMD Opteron 2347HE CPUs (8 cores) running at 1.9GHz, with 32GBs of memory, and 2 X 750 GB discs.
- 1 server with a single Intel Core 2 Quad CPU Q9300 (4 cores) running at 2.50GHz, with 32GB of memory, and 1 X 250 GB disc.
- 3 servers with 4 Intel Xeons E5520 (16 cores) running at 2.27GHz, with 32GB of memory, and 1 X 146 Gb disc.

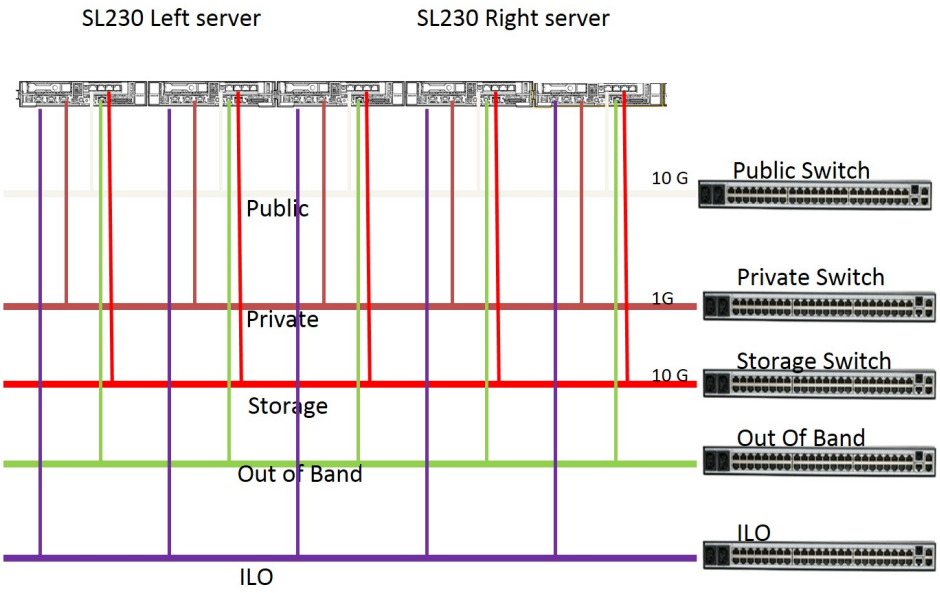

Figure 4.6: Sketch of Nokia's Site Available for Year 2

4.3.4.2 UCL site

This testbed is used inside UCL for various development requirements and for various testing scenarios. It allows us to evaluate different virtualization strategies and technolgies, as well as different management and orchestration techniques for dealing with the virtualized infrastructure.

The testbed can be configured as a NFVi Point-of-presence for testing and evaluation for Y2.

5 Conclusions

This deliverable, focus was two-fold, (i) provide specifications for the SONATA infrastructure flavours and define the Pilots stemming from the use cases and (ii) act as a technical report, providing guidelines for the deployment of SONATA infrastructures, also providing description of the pilot sites that will host the pilots and an operational view on the SONATA platform.

This deliverable, introduced and used a selection methodology based on 5 criteria in order to categorise the use cases and provide a most probable selection of the pilots. This criteria are based on the project set objectives and KPIs but also take into account non-functional characteristics implied by each UC. The process concluded with the selection of vCDN as the selected pilot, plus a list that will be elaborated in the future with smaller, focused pilots.

A Bibliography

[1] AlticeLabs. Wicm: Wan infrastructure and connectivity management. Website, 2016. Online at https://github.com/T-NOVA/WICM.

[2] Archlinux. Kvm. Website, June 2016. Online at https://https://wiki.archlinux.org/index.php/KVM.

[3] Ceph Storage Communit. Ceph. Website, June 2016. Online at http://ceph.com/.

[4] Qemu Community. Qemu. Website, 2016. Online at http://wiki.qemu.org/Main_Page.

[5] The OpenStack Community. Openstack ceilometer project. Website, October 2015. Online at https://wiki.openstack.org/wiki/ceilometer.

[6] The OpenStack Community. Openstack cinder project. Website, May 2016. Online at https://wiki.openstack.org/wiki/cinder.

[7] The OpenStack Community. Openstack glanceproject. Website, May 2016. Online at https://wiki.openstack.org/wiki/Glance.

[8] The OpenStack Community. Openstack monasca project. Website, May 2016. Online at https://wiki.openstack.org/wiki/Monasca.

[9] The OpenStack Community. Openstack neutron project. Website, May 2016. Online at https://wiki.openstack.org/wiki/neutron/.

[10] The OpenStack Community. Openstack nova project. Website, May 2016. Online at https://wiki.openstack.org/wiki/nova/.

[11] The OpenStack Community. Openstack tacker project. Website, May 2016. Online at https://wiki.openstack.org/wiki/tacker/.

[12] SONATA Consortium. Sonata deliverable 2.1: Use cases and requirements.

[13] SONATA Consortium. Sonata deliverable 2.2: Architecture design.

[14] SONATA consortium. D5.1 continuous integration and testing approach. Website, December 2015.

[15] SONATA consortium. D3.1: Basic sdk prototype. Website, May 2016.

[16] SONATA consortium. D4.1: Orchestrator prototype. Website, May 2016.

[17] SONATA consortium. D5.2: Integrated lab based sonata platform. Website, June 2016.

[18] T-NOVA consortium. D2.32: Specification of the infrastructure virtualisation, management and orchestration - final. Website, October 2015. Online at http://www.t-nova.eu/wp-content/uploads/2016/03/TNOVA_D2.32-_Specification-of-the-Infrastructure-Virtualisation-Management-and-Orchestration_v1.0.pdf.

[19] Open Platform for Network Function Virtualisation (OPNFV). Opnfv. Website, 2016. Online at http://www.opnfv.org/about.

[20] Linux Foundation. Documentation let's encrypt. Website, July 2016. Online at https://letsencrypt.org/docs/.

[21] G. Garcia and A. Hoban. Mwc'16 architecture overview and workflow, 2016. Online at https://osm.etsi.org/wikipub/images/d/d4/OSM%2816%29000018r1_MWC16_architecture_overview.pdf.

[22] Quickspecs - hp proliant sl390s generation 7 (g7). Website, November 2014. Online at http://www8.hp.com/h20195/v2/GetPDF.aspx/c04123322.pdf.

[23] Docker Inc. Docker: An open platform for distributed applications, August 2013, howpublished=Website. Online at http://www.docker.com/.

[24] ETSI European Telecommunications Standards Institute. Network functions virtualisation (nfv); management and orchestration v1.1.1. Website, December 2014. Online at http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf.

[25] ETSI NFV ISG. Nfv performance & portability best practices, etsi, v1.1.1, 2014. Online at http://www.etsi.org/deliver/etsi_gs/NFVPER/001_099/001/01.01.01_60/gs_nfv-per001v010101p.pdf.

[26] Jenkins. Jenkins documentation. Website, June 2016. Online at https://jenkins.io/doc/.

[27] M. Tim Jones. Virtual linux - an overview of virtualization methods, architectures, and implementations. Website, Dec 2006. Online at https://web.archive.org/web/20080327111126/http://www-128.ibm.com/developerworks/linux/library/l-linuxvirt/?ca=dgr-lnxw01Virtual-Linux.

[28] Mark Lambe. Dimensioning openstack neutron for nfv systems. Website, September 2014. Online at https://www.sdxcentral.com/articles/contributed/dimensioning-openstack-neutron-nfv-systems-mark-lambe/2014/09/.

[29] linuxcontainers.org. Lxc introduction. Website, June 2016. Online at https://linuxcontainers.org/lxc/introduction/.

[30] NFVLabs@TID. Openmano, 2016. Online at https://github.com/nfvlabs/openmano.

[31] OSM. Google guice: Agile lightweight dependency injection framework, 2016. Online at https://osm.etsi.org/.

[32] The OpenStack Project. OpenStack: The Open Source Cloud Operating System. Website, July 2012. Online at http://www.openstack.org/.

[33] The OpenStack Project. Openstack foundation report: Accelerating nfv delivery with openstack global - telecoms align around open source networking future. Website, 2016. Online at https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf.

[34] Puppet. Ceph. Website, June 2016. Online at https://puppet.com/.

[35] Csaba Rotter, Lóránt Farkas, Gábor Nyíri, Gergely Csatári, László Jánosi, and Róbert Springer. Using linux containers in telecom applications. *Innovations in Clouds, Internet and Networks, ICIN*, 2016.

[36] year=2015 Szabo, R and Kind, M. Towards recursive virtualization and programming for network and cloud resources. `https://tools.ietf.org/html/draft-unify-nfvrg-recursive-programming-02`.

[37] Giacomo Vacca. Docker from scratch. Website, February 2016. Online at `http://www.slideshare.net/GiacomoVacca/docker-from-scratch`.

[38] wikipedia. chroot. Website, June 2016. Online at `https://en.wikipedia.org/wiki/Chroot`.

[39] ZHAW. Opensource sdk for sdn - service function chaining. Website, 2016. Online at `https://blog.zhaw.ch/icclab/gui-for-netfloc/`.