



TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D7.1

Early Pilot Site Deployment

Editor G. Xilouris (NCSR)

Contributors C. Xilouris, S. Kolometsos, E. Trouva, C. Sakkas (NCSR), J. Carapinha, J. Bonnet, M. Dias, J. Silva (PTIN), I. Trajkovska, D. Baudinot, P. Harsh (ZHAW), A. Ramos, J. Melian (ATOS), G. Alexiou, E. Markakis (TEIC), P. Comi, Engo Figini (ITALTEL), L. Zuccarro (CRAT), A. Petrini (UNIMI), M. McGrath (INTEL), G. Gardikis (SPH).

Version 1.0

Date December 31st, 2015

Distribution PUBLIC (PU)

Executive Summary

This Deliverable presents in details the first phase of the activities associated with the deployment of the pilot sites. During this first phase, the focus has been on the primary pilot site, hosted in Athens in NCSR D premises. An early view on the deployed infrastructure is provided. 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 constitute a rough technical guide for the installation and integration of T-NOVA components. Therefore, it is addressed to any members of the wider research/industrial community who wish to replicate (all or part of) the T-NOVA architecture in their own lab infrastructure.

In specific, the deliverable presents the installation and configuration of:

- The Networking infrastructure, comprising of physical and virtual switches and routers as well as a VPN concentrator allowing remote access
- The IT cloud infrastructure, organised in two Openstack clusters
- The Virtualisation Infrastructure Management components. These include the Cloud and Network controllers (Openstack and OpenDaylight), the SFC (Service Function Chaining) framework and the Monitoring framework, as well as the WICM
- The Orchestrator components, including the TeNOR core services, the Infrastructure Repository, the Service Mapping as well as the Gatekeeper.
- The Marketplace components along with the NF Store.

For each of the above mentioned subsystems, the installation procedure is detailed along with any hardware and software dependencies, as well as testing/validation procedures.

Currently, all partners are engaged in an iterative continuous integration procedure, in which updated versions of the modules are integrated into the testbed and validated.

As soon as the pilot reaches an acceptable maturity level, most of the components will also be deployed to the rest pilots and inter-pilot scenarios will be experimented upon. These advances will be included in the second edition of this deliverable (D7.2).

Table of Contents

1. INTRODUCTION	8
1.1. T-NOVA PILOTS AND INTEGRATION STRATEGY	8
1.2. DELIVERABLE INTERDEPENDENCIES	10
2. CONFIGURATION AND DEPLOYMENT OF BASIC NETWORK INFRASTRUCTURE	11
2.1. PHYSICAL NETWORK PROVISIONING	11
3. INFRASTRUCTURE VIRTUALISATION AND MANAGEMENT DEPLOYMENT....	14
3.1. NFVI TOPOLOGY OVERVIEW	14
3.1.1. <i>NFVI Logical View</i>	14
3.1.2. <i>NFVI Physical View</i>	15
3.1.3. <i>Validation of connectivity among NFVI components</i>	19
3.2. VIRTUALIZED INFRASTRUCTURE MANAGEMENT (VIM) COMPONENTS DEPLOYMENT	21
3.2.1. <i>Openstack Controller – HEAT Service</i>	21
3.2.2. <i>SDN Controller</i>	23
3.2.3. <i>SFC Framework</i>	25
3.2.4. <i>Monitoring Framework</i>	30
3.3. WAN INFRASTRUCTURE CONNECTIVITY MANAGER (WICM)	33
3.3.1. <i>WICM Installation</i>	33
3.3.2. <i>WICM demonstration</i>	34
4. ORCHESTRATION LAYER DEPLOYMENT	38
4.1. TENOR	38
4.1.1. <i>Main Installation file</i>	38
4.1.2. <i>Cassandra Installation</i>	39
4.1.3. <i>LogStash Installation</i>	39
4.1.4. <i>MongoDB Installation</i>	40
4.1.5. <i>Micro-services registration</i>	42
4.1.6. <i>Byobu Installation</i>	42
4.2. INFRASTRUCTURE REPOSITORY	43
4.2.1. <i>Prerequisites</i>	45
4.2.2. <i>EPA Controller</i>	45
4.2.3. <i>EPA Agent</i>	46
4.2.4. <i>API Middleware</i>	46
4.2.5. <i>Resolved Issues</i>	47
4.3. SERVICE MAPPING	47
4.3.1. <i>Manual Installation</i>	48
4.3.2. <i>Automatic Installation</i>	51
4.3.3. <i>Service Mapper module configuration</i>	51
4.3.4. <i>Starting and stopping the Service Mapper module</i>	52
4.4. GATEKEEPER	52

4.4.1. <i>Expression Solver (assurance formula evaluator)</i>	53
5. MARKETPLACE AND NF STORE DEPLOYMENT	56
5.1. NF STORE.....	56
5.1.1. <i>Build NF Store</i>	56
5.1.2. <i>Deploy NF Store</i>	57
5.1.3. <i>Start NF Store</i>	58
5.2. MARKETPLACE	59
5.2.1. <i>Prerequisites</i>	59
5.2.2. <i>Marketplace Deployment</i>	61
6. OVERVIEW OF T-NOVA DEPLOYMENT.....	63
7. CONCLUSIONS	65
8. REFERENCES	66
9. LIST OF ACRONYMS	68

Index of Figures

Figure 1 Overall view of the T-NOVA Pilots	9
Figure 2 Physical Network Topology	11
Figure 3 Cisco ASA VPN Tunnel Statistics	13
Figure 4 Cacti Infrastructure monitoring	13
Figure 5 Overall view of Athens Pilot	14
Figure 6 Logical view of NFVI-PoP	15
Figure 7 NFVI-PoP1 physical view	16
Figure 8 Example Component Deployment View	17
Figure 9 NFVI-PoP2 physical view	18
Figure 10 Internet access test	19
Figure 11 Intra-PoP connectivity test.....	20
Figure 12 Inter-PoP connectivity test.....	20
Figure 13 Inter-VM testing for the same tenant.....	20
Figure 14 Inter-VM testing for different tenants.....	21
Figure 15 Orchestration (HEAT) view in Openstack Dashboard	23
Figure 16 NFVI-PoP 1 Discovered topology.....	24
Figure 17 NFVI-PoP 2 Discovered topology.....	25
Figure 18 OpenStack and OpenDaylight nodes in Athens testbed	26
Figure 19 OVS configuration on SDN switch.....	27
Figure 20 API POST call for service chain creation	29
Figure 21 Sender VM console as pinging to receiver resumes after the flow has been installed	29
Figure 22 ICMP messages in the dummy VNF (left) and SFC flows on the switch (right)	30
Figure 23 - WICM demo setup	35
Figure 24 Infrastructure Repository on ILE's Github	44
Figure 25 Go runtime, environment and dependencies deployment.....	52
Figure 26 Gatekeeper started and listening for requests.....	53
Figure 27 Excerpt from Expression Solver log	54
Figure 28 Validation of the service deployment.....	54
Figure 29 Component network topology.....	63

Index of Listings

Listing 1: Configuration of InfluxDB docker container	31
Listing 2: Back-end deployment.....	31
Listing 3: Grafana Deployment	32
Listing 4: Configuration of collectd network plugin.....	32
Listing 5: TeNOR's main modules installation (file install.sh).	39
Listing 6: Installing Cassandra (installation_cassandra.sh).	39
Listing 7: Installing LogStash (file installation_logstash.sh).....	39
Listing 8: Installing MongoDB (file installation_mongodb.sh).....	41
Listing 9: Loading micro-services (file loadModules.sh).....	42
Listing 10: Setting up all sessions using Byobu.....	43
Listing 11 Infrastructure Repository deployment.....	44
Listing 12: Extract of the EPA Configuration File	46
Listing 13: Ruby gems required for the Service Mapper module installation.....	49
Listing 14: Compiling the jsonConverter and solver applications	50
Listing 15. Message on successful automated installation of the Service Mapper module.....	51
Listing 16 Gatekeeper Configuration	53
Listing 17 Expression Solver configuration file.....	54
Listing 18 NF Store build command output.....	57
Listing 19 NF Store Configuration File.....	58
Listing 20 Apt repository GPG key installation.....	59
Listing 21 Docker composition configuration file	61

Index of Tables

Table 1 Deliverable interdependencies	10
Table 2 Physical Network Nodes	12
Table 3 NFVI-PoP1 nodes' specifications	17
Table 4 Specification of NFVI-PoP2 Nodes	18
Table 5 Environment variables for the communication with Openstack	32
Table 6 Gatekeeper Components.....	54
Table 7 NF Store Configuration.....	58

1. INTRODUCTION

This deliverable serves both as a technical progress report but also as an early handbook for the installation and deployment of the T-NOVA components. The approach followed in this document is to reference details of components that have been implemented and tested in other technical WPs (i.e. WP3, WP4 and WP5) and provide the technical deployment and integration details.

The document is structured as follows:

Chapter 1 is the introduction of the deliverable discussing (briefly) the integration strategy and the T-NOVA Pilots. It also provides the interlinked documents that are released by the project and provide details on various aspects of the integration.

Chapter 2 specifies the architecture of the testbed physical infrastructure, focusing on the networking part.

Chapter 3 discusses the deployment and integration of the components of the NFVI and Virtualised Infrastructure Management (VIM) layer, including the OpenStack and OpenDaylight controllers, the SFC framework, the monitoring system and the WICM.

Chapter 4 presents the deployment of the Orchestration layer components. Detailed instructions for the deployment of the TeNOR micro-services are provided (Cassandra, LogStash, MongoDB and Byobu). Separate sub-sections describe the deployment of the Infrastructure Repository, the Service Mapping and the Gatekeeper modules.

In Chapter 5 guidelines for the NFStore and MarketPlace components installation are described.

Chapter 6 presents an overall view of the deployed pilot and, finally, Chapter 7 concludes the document.

1.1. T-NOVA Pilots and Integration Strategy

As it is specified and planned in the T-NOVA Description of Work, three pilots are anticipated. These Pilots have been presented in the Deliverable [2.51] and [2.52]. A brief overview of the Pilots is given in Figure 1.

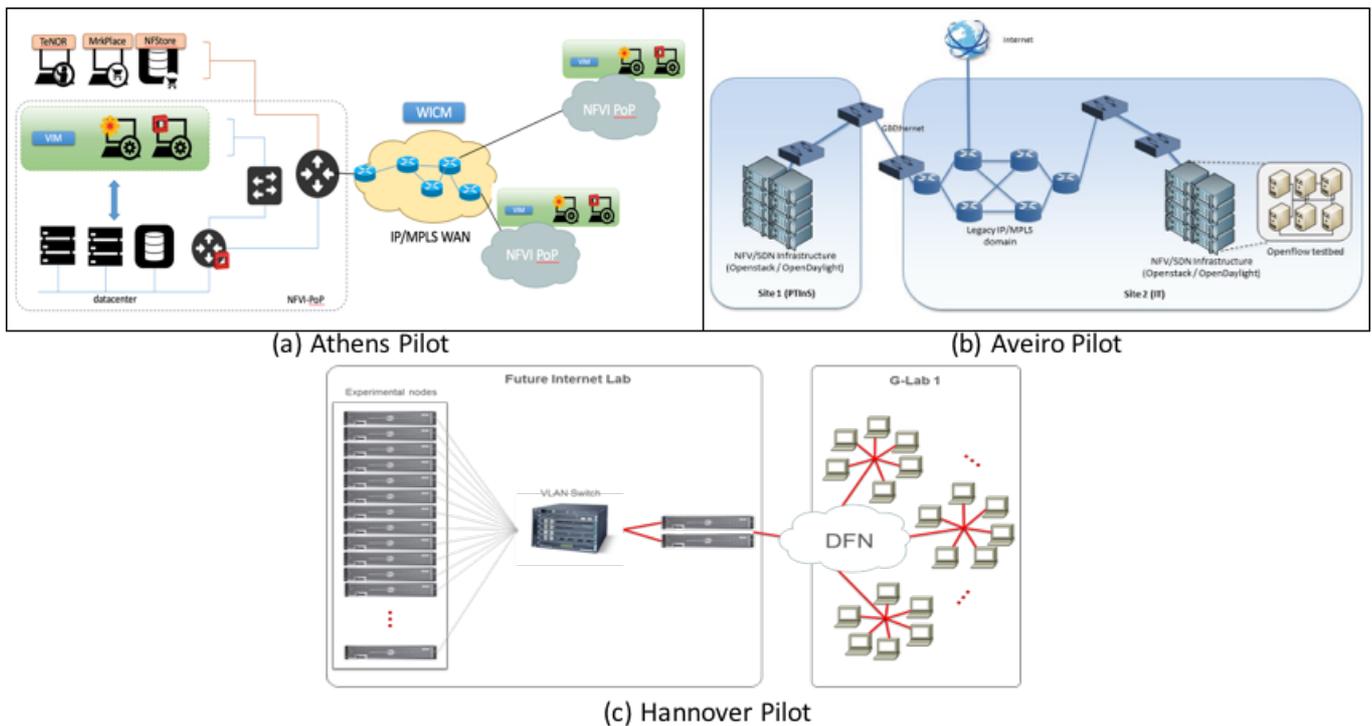


Figure 1 Overall view of the T-NOVA Pilots

The T-NOVA consortium initially has focused in integration efforts in the primary Pilot i.e the Athens Pilot (Figure 1a). The integration of developed components in the pilot started already during the early implementation efforts and the Pilot infrastructure has been continuously evolved over the second year of the project. The efforts in WP4 (i.e Deliverable [D4.51]) founded a usable NFV infrastructure on which the rest components were gradually installed.

Initially, the WP5 developments were deployed and validated. The focal points of these efforts was the automatic deployment of the NFVI using HEAT templates and the stand-alone operation of each VNF. The next step was the integration of WP3 components, namely the Orchestrator, Infrastructure Repository and the Monitoring framework. Finally, the last part was devoted to the deployment and integration of the Marketplace with the Orchestration modules. The Marketplace components were made available as deployable Docker [docker] containers which proved to be a very convenient way for Marketplace deployment.

The following steps in the integration, is to transfer the components and the accumulated know-how in order to effectively deploy all the required - for the planned experimentation - components to the rest Pilots. The actual deployment in all the Pilots and the validation of the end-to-end chain is subject of the Deliverable 7.2 to be delivered at the end of the 3rd year of the project.

1.2. Deliverable interdependencies

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

Table 1 Deliverable interdependencies

Deliverable Name	Description	Reference
D2.51/D2.52 - Planning of trials and evaluation	D2.51/2.52 present a birds-eye view of the T-NOVA Pilots and testbeds as well as the purpose and scope of each one.	[D2.51], [D2.52]
D3.1 - Orchestrator Interfaces D3.2 - Infrastructure Resource Repository D3.3 - Service Mapping D3.41 - Service Provisioning, Management and Monitoring - Interim	WP3 deliverables discuss the implementation of various Orchestration components. They also provide information on the appropriate interfaces used for the intra and inter-component communications. Information on the deployment of each component in the NFV infrastructure is extracted.	[D3.1], [D3.2], [D3.3], [D3.41]
D4.51 - Infrastructure Integration and Deployment	D4.51 focuses in the implementation of a testbed comprised the implementations of the functional entities of the T-NOVA IVM layer namely the NFVI, and VIM. The task will integrate the network and cloud assets into a composite network infrastructure. The aim is not only to present technical advances in individual components, but to demonstrate the added value of the integrated IVM architecture as a whole.	[D4.51]
D5.1 - Function Store	D5.1 focuses in the implementation of the Network Function Store and VNF packetisation. System specifications are used in order to provide the necessary resources.	[D5.1]
D5.31 - Network Functions Implementation and Testing - Interim	This presents a description of the Virtual Network Functions (VNFs) developed in the T-Nova project. Those VNFs have been developed in order to demonstrate, with real-life applications, the capabilities offered by the overall T-Nova framework.	[D5.31]
WP6 Deliverables	WP6 deliverables describe the interfaces with the Orchestration layer and with the NF Store.	[D6.1], [D6.2], [D6.3], [D6.4]

2. CONFIGURATION AND DEPLOYMENT OF BASIC NETWORK INFRASTRUCTURE

This section summarises the configuration and deployment of the required physical network elements and their interconnection so as to form the core backbone for the testbed.

2.1. Physical Network provisioning

In order to initiate the deployment of the NFVI PoPs in Athens Pilot, a series of configurations need to take place to put in place the physical network that will allow the proper connectivity of the PoPs with the Internet as well as ensure the connectivity between them. The Figure 2 depicts the topology of the physical network components in the Pilot.

The network is divided in three main network segments/branches; two for the two PoPs and one for the higher-layer T-NOVA services (Orchestrator, Marketplace etc.)

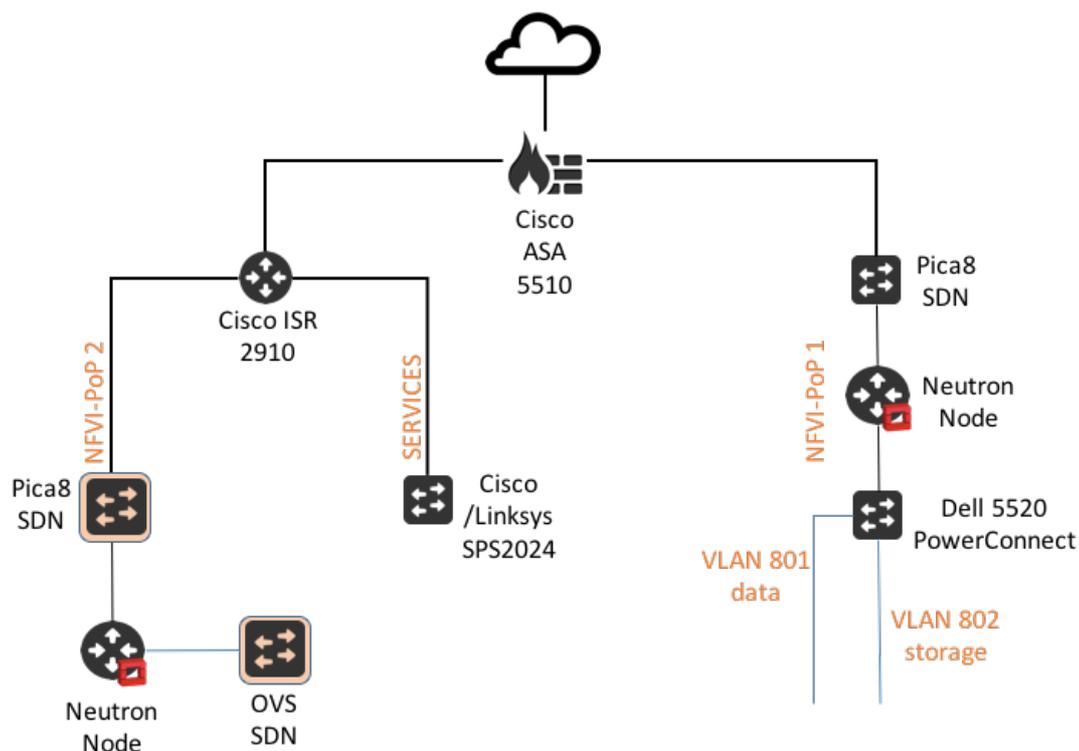


Figure 2 Physical Network Topology

The network is configured flat at the PoP-2 segment while it is configured with VLANs for the PoP-1. VLAN tag 801 is used for the data network (connectivity for Compute Nodes) and VLAN tag 802 for the storage network for access to the NAS storage.

Table 2 below enumerates the network nodes used for the realization of the aforementioned topology.

Table 2 Physical Network Nodes

Id	Vendor	Model	Capabilities
1	Cisco	ASA 5510	Firewall, IDS, IPS and VPN concentrator. 4x10/100Mbps interfaces, 1GB RAM
2	Cisco	ISR 2910	Integrated Services Router, nBAR, 4x 10/100/1000Mbps interfaces
3	Pica8	P3297	48 x 10/100/1000BASE-T RJ45 port base unit, with four 10 GbE SFP+ uplinks, Layer 2 / Layer 3 protocols with industry-leading OpenFlow 1.4 / Open-vSwitch (OVS) 2.0 integration, switching fabric capacity of 176 Gbps
4	OVS SDN	OVS 2.0	Custom made 5x 10/100 BASE-T Interfaces supporting OpenFlow 1.4
5	Dell	PowerConnect 5524	2x24px10/100/1000BASE-T RJ45. Not Openflow compatible. Supporting Jumbo frames and iSCSI optimization.
6	Cisco-Linksys	SPS2024	24-port 10/100/1000 Ethernet Switch. Not OpenFlow compatible.

In order to allow connectivity with the Internet, NAT services are offered by the CISCO ASA network element. The NAT is configured either to be *dynamic* in order to allow all the hosts to reach internet or public addresses, or static NAT to allow also access to specific services from the inside networks to be reachable outside the firewall.

Additionally, all the partners are able to remotely connect to the infrastructure either via software SSL clients (Cisco AnyConnect) or over Site-to-Site IPSEC VPN connections (see Figure). This allows instant reachability for all the development teams or experimenters currently working in T-NOVA as well as seamless expansion of the Pilot infrastructure with other Pilots or infrastructure located in other geographical locations (e.g. expansion of the Athens Pilot to TEIC test-bed). In order to increase security and also manage connectivity problems each partner is provided with separate credentials and connection profile.

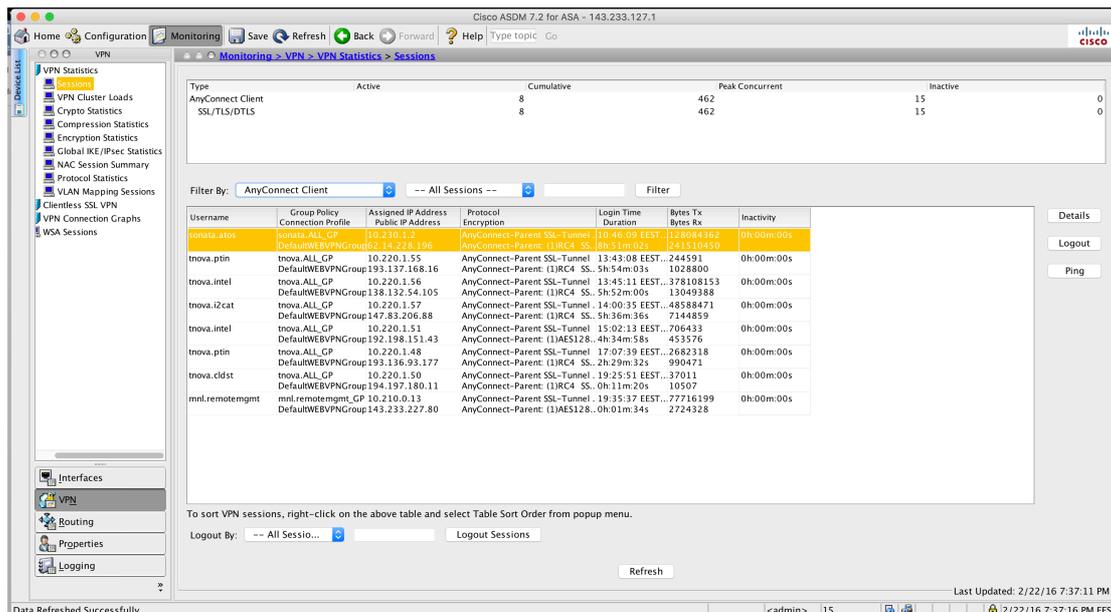


Figure 3 Cisco ASA VPN Tunnel Statistics

An overall network monitoring solution is also provided, as an additional measure for ensuring network performance and problem solving during operation on the infrastructure. Figure 4 presents statistics gathered for the Firewall and Gateway services using the network and service monitoring platform [Cacti].



Figure 4 Cacti Infrastructure monitoring

Having configured the basic networking infrastructure, the integration phase proceeds with the deployment and configuration of the NFV infrastructure and all the related components.

3. INFRASTRUCTURE VIRTUALISATION AND MANAGEMENT DEPLOYMENT

This chapter presents details on the deployment of the NFVI and VIM (IVM) layers. In contrast to the deliverables which refer to WP4 integration (D4.51/D4.52) and which contain overall description of the IVM layer deployment, this chapter presents the actual sequence of steps taken to deploy the IVM components in the Athens pilot.

3.1. NFVI Topology Overview

This section presents the topology of the Athens Pilot as deployed currently in NCSR premises. Various views of the NFVI infrastructure are presented along with the deployment of other components integrated to it. It was selected that one PoP which would essentially be the more powerful one in terms of IT resources would also be used to deploy the T-NOVA components. As presented previously in D2.52, the Athens Pilot is comprised of at least two NFVI PoPs. In later stages the PoPs will be interconnected by an emulated WAN in order to test the full end-to-end chain as it is envisaged by T-NOVA project. The overall view is illustrated in Figure 5.

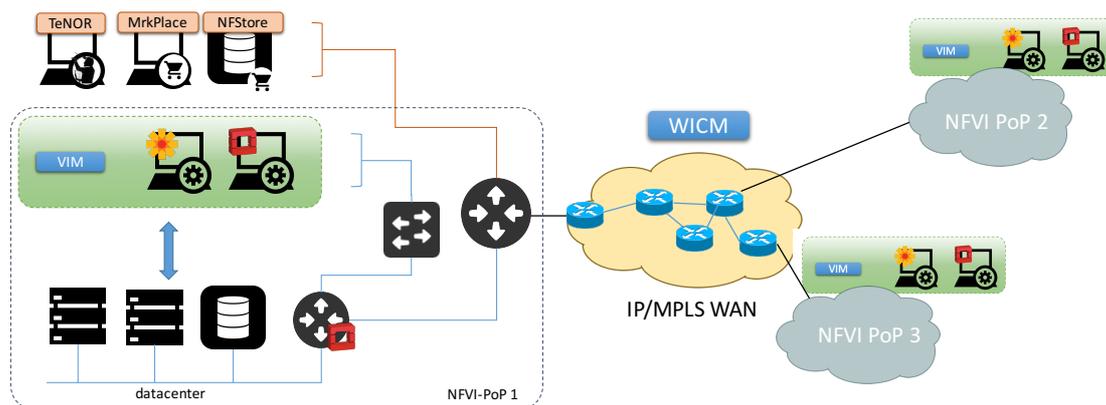


Figure 5 Overall view of Athens Pilot

In order to provide insights in the topology and deployment of components, more views of the Pilot will be presented later in this document.

3.1.1. NFVI Logical View

The logical view of the reference T-NOVA NFVI – PoP is illustrated in Figure 6. It can be noticed that all the main components are depicted. Detailed description of this figure is provided in [D2.52]. For simplicity it was selected that i) the TeNOR, ii) the Marketplace, iii) the NFStore and iv) the WICM will be deployed in one of the NFVI-PoPs implemented for the pilot.

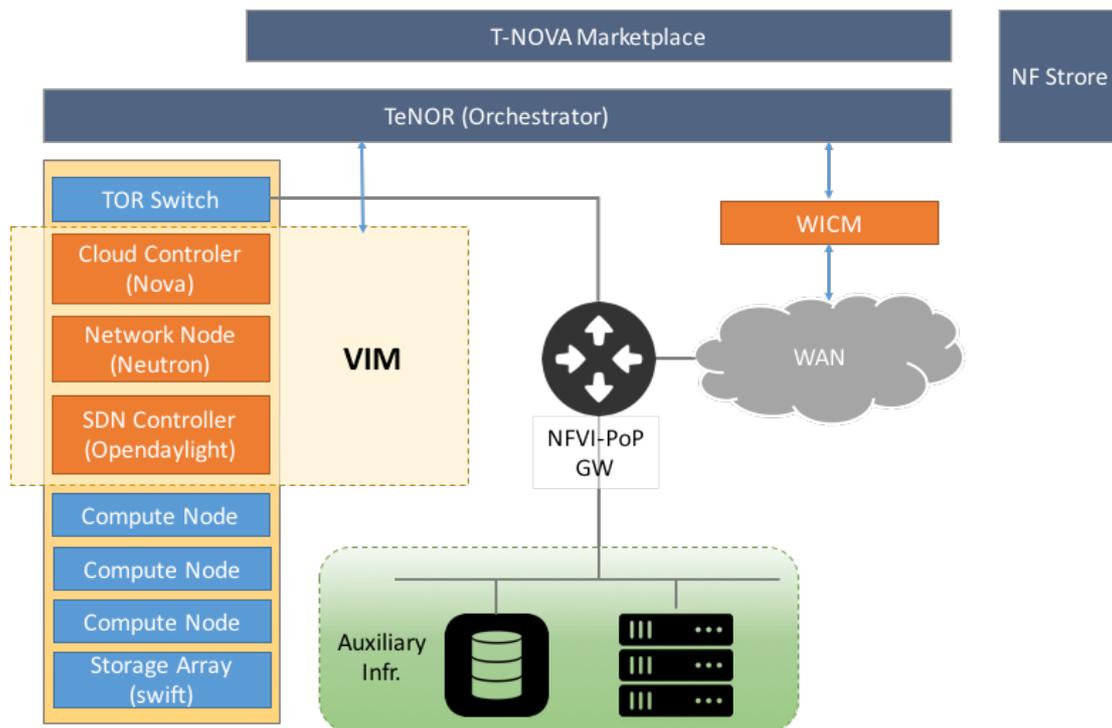


Figure 6 Logical view of NFVI-PoP

3.1.2. NFVI Physical View

The Pilot is comprised of two separate NFV infrastructures (NFVIs) interconnected at the moment via a L3 network. Each NFVI is considered as a unique NFVI-PoP with its own management entities i.e. the VIM. Currently different versions of the same components (compute nodes, network node, SDN controller, storage nodes, etc.) constitute each NFVI-PoP. Moreover, each PoP has a different number of compute nodes, resulting in different amounts of IT resources for VNF provisioning. The network interconnecting the two PoPs will be replaced by a dedicated subsystem emulating a wide-area MPLS network.

3.1.2.1. NFVI – PoP1

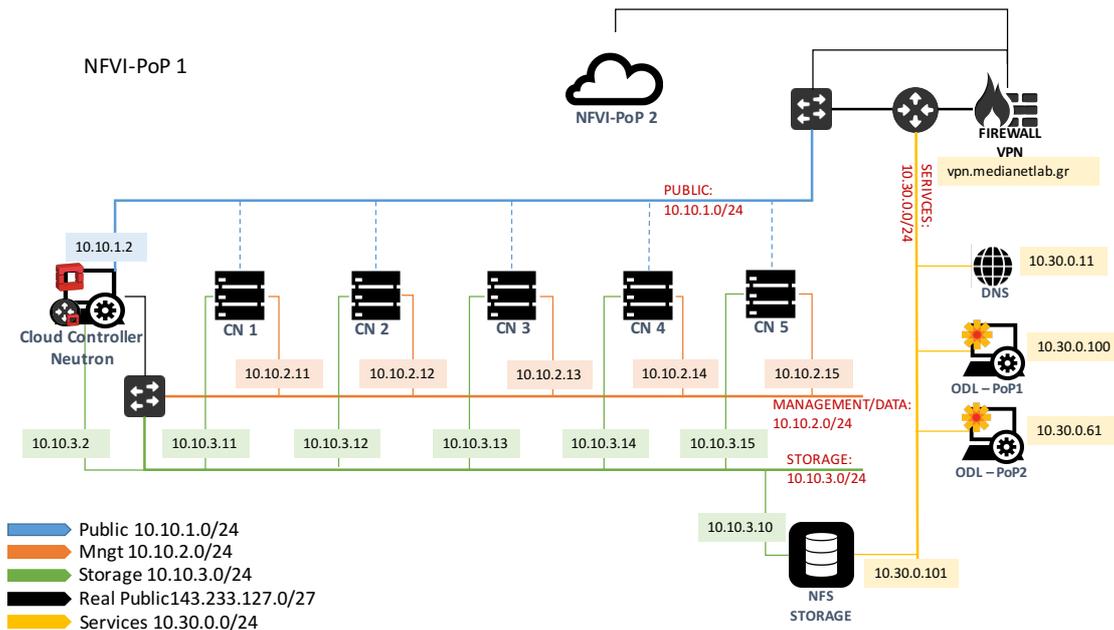


Figure 7 NFVI-PoP1 physical view

Figure 7 illustrates the physical topology of the network nodes, compute nodes and network links that constitute the NFVI-PoP1. This PoP as explained later in this document provides networking to the VNFs via the Openstack Neutron service. Thus all the networking is taken care automatically by Openstack provided that the physical networks required for the proper operation of Openstack are provided. For this reason, three network segments are provisioned: i) Public network for assignment of “public” IPs at the VNFs; ii) Management network used for communication of the various Openstack distributed services and iii) Storage network used for provisioning of shared storage between the Compute Nodes via Network File System [NFS]. The shared storage stores the VNF instances allowing for fast migration and minimizing the latencies due to network transfers.

In addition to the above, a “Services” network segment was created in order to host auxiliary and common services for the aforementioned NFVI-PoPs. In this segment common Domain Name Server (DNS) for all PoPs and ODL instances for each PoP are hosted.

Due to the resource availability in this PoP it was selected to deploy all the T-NOVA components in the provided infrastructure. The figure below depicts a snapshot of this deployment. Blue rectangles represent native Openstack services while the grey rectangles represent the T-NOVA components.

As shown there are five compute nodes in this deployment. The first three are serving with the expected OpenStack role of compute node in a common OpenStack deployment, servers on which the users will create their virtual machines. The other two compute nodes in the infrastructure provide additional virtualisation capabilities. Compute Node 4 offers 2x10Gbps NICs that support SR-IOV for enhanced networking performance and Compute Node 5 offers GPU acceleration via nVidia graphics card.

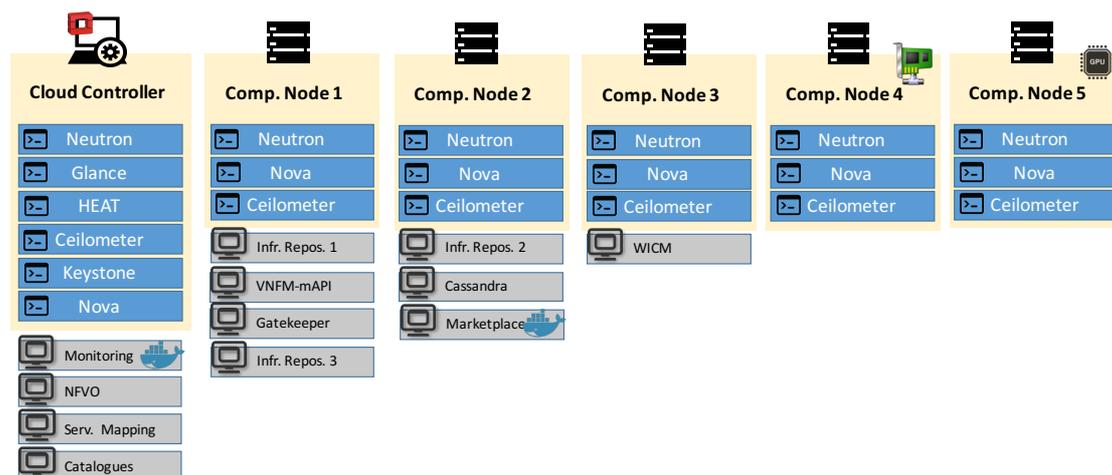


Figure 8 Example Component Deployment View

The specifications of the nodes that are deployed in this PoP are enumerated at the table below. Particularly two nodes offer acceleration characteristics to the deployed VNFs. Compute Node 5 is offering GPU acceleration support for workload configured to utilise the multiple processing cores available in the GPU, while SR-IOV capable 10Gbps Network cards in compute node 4 allow for direct path access for enhanced network performance.

Table 3 NFVI-PoP1 nodes' specifications

ID	Role	Vendor	CPU			RAM	Storage	Extra Features
			Model	CPU sockets	Cores			
1	Cloud Controller / Neutron Node	HP Proliant DL380e Gen 8	Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz	1	12	16	1 TB	-
2	Compute Node 1	Dell R630	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz	2	16	32	500 GB	-
3	Compute Node 2	Dell R630	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz	2	16	32	500 GB	-
4	Compute Node 3	Dell R630	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz	2	8	32	500 GB	-
5	Compute Node 4	Dell R630	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz	2	16	32	250 GB	SR-IOV
6	Compute Node 5	HP Z230	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz	1	8	16	500 GB	GPU Acc.

3.1.2.2. NFVI-PoP2

Figure 9 illustrates the physical topology of the network nodes, compute nodes and network links that constitute the NFVI-PoP2. This PoP is focused on providing full SDN integration with the Openstack environment. Due to incompatibilities of the latest Openstack version, Openstack Liberty, with ODL Helium, a previous version of Openstack (i.e. Juno) is used. The PoP is comprised of 3 Nodes: i) a Neutron Node running only the Neutron Service, ii) a Cloud Controller node running the nova service acting also a compute node – running the nova-compute service and iii) a compute node running nova-compute service. Networking across the PoP is achieved with OVS at each node and the Pica8 SDN switch for physical interconnection. As in the case of PoP1, there is a separate network segment where the ODL node for this PoP is connected. Particularly this PoP serves the purpose of experimentation for the SFC and WICM integration. Finally, the PoP is serviced by the same DNS server as the PoP1.

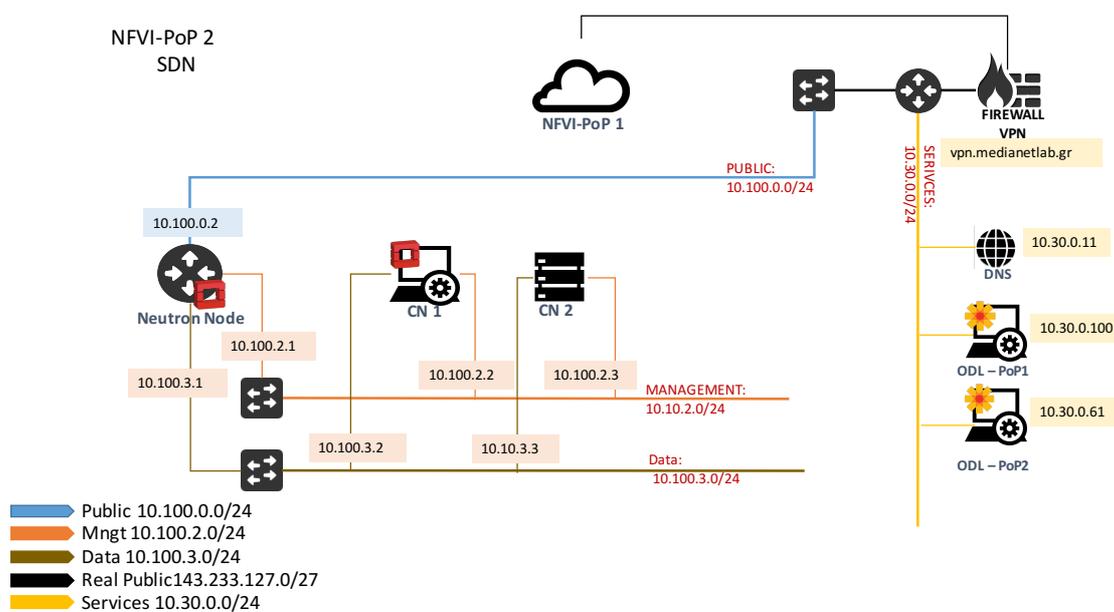


Figure 9 NFVI-PoP2 physical view

The table below summarised the specifications of the server nodes used for the realisation of the second PoP (i.e NFVI-PoP 2).

Table 4 Specification of NFVI-PoP2 Nodes

ID	Role	Vendor	CPU			RAM	Storage	Extra
			Model	CPU sockets	Cores			
1	Neutron Node	BTO Workstation	Intel(R) Pentium(R) 4 CPU 3.00GHz	1	2	4	300 GB	-

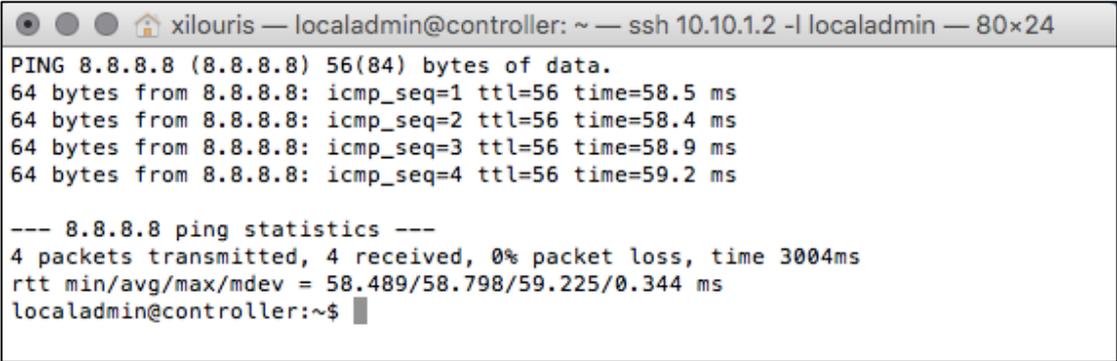
2	Cloud Controller	Dell R210 II	Intel(R) Xeon(R) CPU E3-1240 V2 @ 3.40GHz	1	8	16	1 TB	-
3	Compute Node 1	Dell R210 II	Intel(R) Xeon(R) CPU E3-1240 V2 @ 3.40GHz	8	16	16	1 TB	-

3.1.3. Validation of connectivity among NFVI components

Following the deployment of the NFVI, some simple connectivity tests (via ping) are executed in order to verify the connectivity between the NFVI infrastructure tenants and the proper operation of the infrastructure.

3.1.3.1. Testing connectivity to the internet

The primary test is to test connectivity from a host within the PoPs to the Internet.



```
xilouris — localadmin@controller: ~ — ssh 10.10.1.2 -l localadmin — 80x24
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=58.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=58.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=58.9 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=59.2 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 58.489/58.798/59.225/0.344 ms
localadmin@controller:~$
```

Figure 10 Internet access test

The test is successful with zero packet loss and 60ms RTT.

3.1.3.2. Testing connectivity intra-PoP

This test aims to validate that the two VMs within the PoP are able to communicate.

```

localadmin@controller: ~
localadmin@controller:~$ ping cnode1 -c 4
PING cnode1 (10.10.2.11) 56(84) bytes of data.
64 bytes from cnode1 (10.10.2.11): icmp_seq=1 ttl=64 time=2.32 ms
64 bytes from cnode1 (10.10.2.11): icmp_seq=2 ttl=64 time=0.133 ms
64 bytes from cnode1 (10.10.2.11): icmp_seq=3 ttl=64 time=0.130 ms
64 bytes from cnode1 (10.10.2.11): icmp_seq=4 ttl=64 time=0.148 ms

--- cnode1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.130/0.684/2.327/0.948 ms
localadmin@controller:~$
localadmin@controller:~$ ping cnode4 -c 4
PING cnode4 (10.10.2.14) 56(84) bytes of data.
64 bytes from cnode4 (10.10.2.14): icmp_seq=1 ttl=64 time=3.24 ms
64 bytes from cnode4 (10.10.2.14): icmp_seq=2 ttl=64 time=0.147 ms
64 bytes from cnode4 (10.10.2.14): icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from cnode4 (10.10.2.14): icmp_seq=4 ttl=64 time=0.161 ms

--- cnode4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.147/0.935/3.241/1.331 ms
localadmin@controller:~$

```

Figure 11 Intra-PoP connectivity test

The test is successful with zero packet loss and 0.2ms RTT, using the cloud data network (10.10.2.0/24).

3.1.3.3. Testing connectivity between PoPs

This test aims to validate that different VMs in different PoPs can communicate.

```

localadmin@controller: ~
localadmin@controller:~$ ping 10.100.0.3 -c 4
PING 10.100.0.3 (10.100.0.3) 56(84) bytes of data.
64 bytes from 10.100.0.3: icmp_seq=1 ttl=63 time=0.708 ms
64 bytes from 10.100.0.3: icmp_seq=2 ttl=63 time=0.614 ms
64 bytes from 10.100.0.3: icmp_seq=3 ttl=63 time=0.648 ms
64 bytes from 10.100.0.3: icmp_seq=4 ttl=63 time=0.648 ms

--- 10.100.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.614/0.654/0.708/0.042 ms
localadmin@controller:~$

```

Figure 12 Inter-PoP connectivity test

The test is successful with zero packet loss and 0.6ms RTT, using the public neutron node addresses (10.10.1.2 and 10.100.0.3 respectively).

3.1.3.4. Testing inter-vm communication for the same tenant

```

Connected (unencrypted) to: QEMU (instance-00000076)
gatekeeper@gatekeeper-2:~$ ping 172.16.6.31 -c 4
PING 172.16.6.31 (172.16.6.31) 56(84) bytes of data.
64 bytes from 172.16.6.31: icmp_seq=1 ttl=64 time=0.430 ms
64 bytes from 172.16.6.31: icmp_seq=2 ttl=64 time=0.319 ms
64 bytes from 172.16.6.31: icmp_seq=3 ttl=64 time=0.188 ms
64 bytes from 172.16.6.31: icmp_seq=4 ttl=64 time=0.217 ms

--- 172.16.6.31 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.188/0.288/0.430/0.096 ms
gatekeeper@gatekeeper-2:~$ _

```

Figure 13 Inter-VM testing for the same tenant

The test is successful with zero packet loss and 0.3ms RTT, using the provisioned virtual network IPs.

3.1.3.5. Testing inter-vm communication for different tenants

```

Connected (unencrypted) to: QEMU (instance-0000034f)
vagrant@pxaas:~$ ping 10.10.1.67 -c 4
PING 10.10.1.67 (10.10.1.67) 56(84) bytes of data.
64 bytes from 10.10.1.67: icmp_req=1 ttl=62 time=3.23 ms
64 bytes from 10.10.1.67: icmp_req=2 ttl=62 time=0.758 ms
64 bytes from 10.10.1.67: icmp_req=3 ttl=62 time=0.787 ms
64 bytes from 10.10.1.67: icmp_req=4 ttl=62 time=0.729 ms

--- 10.10.1.67 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.729/1.377/3.236/1.073 ms
vagrant@pxaas:~$

```

Figure 14 Inter-VM testing for different tenants

The test is successful with zero packet loss and 1.4ms RTT, using the floating (public) IPs assigned to the VMs.

3.2. Virtualized Infrastructure Management (VIM) components deployment

3.2.1. Openstack Controller – HEAT Service

In order to install the Orchestration Service, also known as Heat, the following steps have to be completed.

First, a database must be created and configured:

```

mysql -u root -p
CREATE DATABASE heat;
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' IDENTIFIED BY
'HEAT_DBPASS';
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' IDENTIFIED BY 'HEAT_DBPASS';

```

Then, to create the service credentials:

```

source admin-openrc.sh
openstack user create --domain default --password-prompt heat
openstack role add --project service --user heat admin
openstack service create --name heat --description "Orchestration"
orchestration
openstack service create --name heat-cfn --description "Orchestration"
orchestration

```

Afterwards, the Orchestration service API endpoints have to be created:

```

openstack endpoint create --region RegionOne orchestration public
http://controller:8004/v1/\$\(tenant\_id\)s
openstack endpoint create --region RegionOne orchestration internal
http://controller:8004/v1/\$\(tenant\_id\)s

```

```

openstack endpoint create --region RegionOne orchestration admin
http://controller:8004/v1/%\(tenant\_id\)s
openstack endpoint create --region RegionOne cloudformation public
http://controller:8004/v1
openstack endpoint create --region RegionOne cloudformation internal
http://controller:8004/v1
openstack endpoint create --region RegionOne cloudformation admin
http://controller:8004/v1
openstack domain create --description "Stack projects and users" heat
openstack user create --domain heat --password-prompt heat_domain_admin
openstack role add --domain heat --user heat_domain_admin admin
openstack role create heat_stack_owner
openstack role add --project demo --user demo heat_stack_owner
openstack role create heat_stack_user

```

At this point the installation and the configuration of the following components are required:

```
apt-get install heat-api heat-api-cfn heat-engine
```

Wait until the installation is complete. Then edit the /etc/heat/heat.conf and add the following lines under the following sections of the file.

In the [database]:

```
connection = mysql+pymysql://heat:HEAT_DBPASS@controller/heat
```

In the [DEFAULT]:

```

rpc_backend = rabbit
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/waitcondition
stack_domain_admin = heat_domain_admin
stack_domain_password = HEAT_DOMAIN_PASS
stack_user_domain_name = heat
verbose = true

```

In the [oslo_messaging_rabbit]:

```

rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS

```

In the [keystone_authoken]:

```

auth_uri = http://controller:5000
auth_uri = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = heat
password = HEAT_PASS

```

In the [trustee]:

```

auth_plugin = password
auth_url = http://controller:35357
username = heat
password = HEAT_PASS
user_domain_od = default

```

In the [clients_keystone]:

```
auth_uri = http://controller:5000
```

In the [ec2authtoken]

```
auth_uri = http://controller :5000
```

Finally, to complete the installation, populate the database and restart the services:

```
su -s /bin/sh -c "heat-manage db_sync" heat
service heat-api heat-api-cfn heat-engine restart
```

If all done correctly, the Orchestration tab should now have been added to the OpenStack dashboard as illustrated in Figure 15.

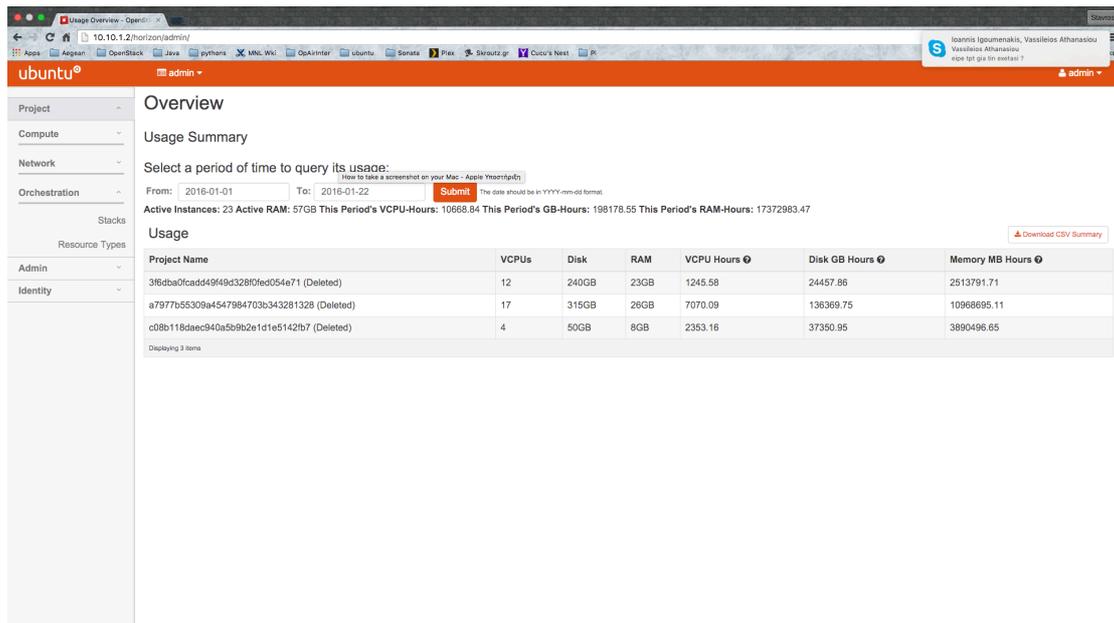


Figure 15 Orchestration (HEAT) view in Openstack Dashboard

Additionally it should also be possible from the command line to access the Orchestration service and execute commands.

3.2.2. SDN Controller

The SDN Controller (i.e OpenDayLight) can be integrated in the Pilot by two means. The first one is a light integration in which the controller controlling the HW switched in the infrastructure but not the OVS instances inside the Compute Nodes. This method is followed for the NFVI-PoP 1. The second method of integration is the tight integration where the ODL communicates via the ml2 plugin with Neutron service (Openstack) and therefore controls also the OVS instances inside the compute nodes. The latter integration mode is used at the NFVI-PoP2. Both modes support a smooth operation of the NFVI environment, however the second mode is not currently compatible with Openstack Liberty release.

3.2.2.1. Light Integration

As said previously the light integration allows the provision of a L2 physical network over which the Openstack Neutron service can deploy the GRE based virtual networks for its tenants. The only interaction of Neutron with ODL is done at the VIM level. It can also be used in production environment where the Provider Networks networking model is used for the deployment. In such cases VLANs are used for realisation of the networking and the virtual networking is provided solely by the Operator NMS. The discussed deployment for T-NOVA is based on OpenStack Liberty, deployed over Ubuntu 14.04 (LTS) OS and OpenDayLight Lithium.

The deployed components are:

- Infrastructure switch: Pica8 (10.30.0.101)
- Openstack Controller/Neutron: 10.10.1.2/24
- ODL Controller: 10.30.0.100

The virtual bridge created at the Pica8 OVS instance is configured to connect to the ODL controller. As soon as the switch establishes connection with the controller, the topology of the network is discovered. The results of the above process are illustrated in Figure 16 that shows the OpenDaylight SDN Controller topology view.

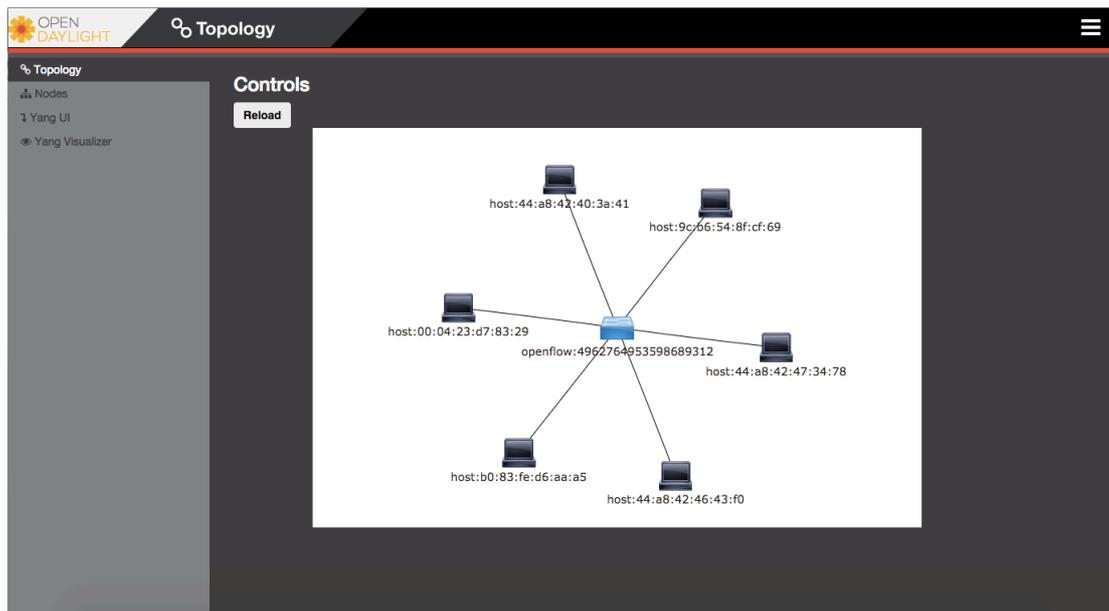


Figure 16 NFVI-PoP 1 Discovered topology

3.2.2.2. Full Integration

Full integration is achieved using OpenStack Juno version deployed over Ubuntu 14.04 (LTS) with OpenDaylight Lithium version. This integration resumes all operations using OF protocol for the control of the OVS instances and used the ml2-plugin for the ODL – Openstack Neutron communication.

This deployment is used for the NFVI-PoP 2 and uses the following components:

- OpenStack nodes, 10.100.2.0/24

- Neutron Node, 10.100.2.1/24
- OpenDaylight controller, 10.30.0.61/24

Complete configuration instructions are available on [D4.51], Annex B. Figure 17 depicts the OpenDaylight SDN Controller topology view, in which there are apparent 3 OVSs (one per OpenStack node). GRE tunnels (not visible) are used in order to realize the virtual networks used by the tenants and their VMs.

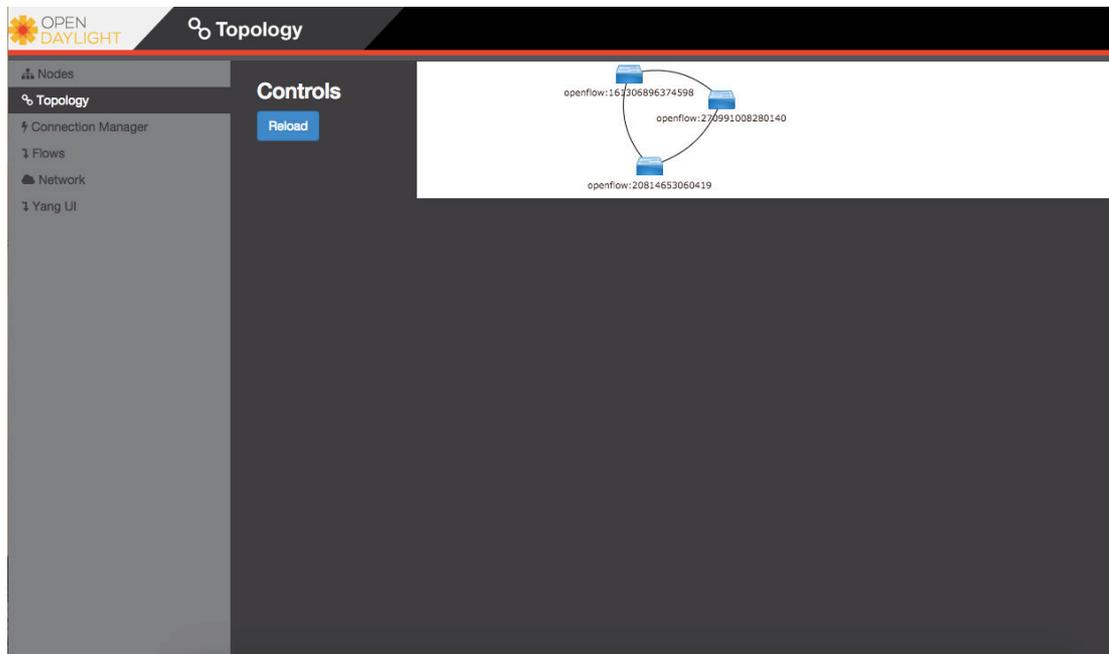


Figure 17 NFVI-PoP 2 Discovered topology

3.2.3. SFC Framework

The SDK4SDN has been implemented in Java programming language and it is currently integrated as a holistic component that includes a Lithium version of the OpenDaylight controller. It has been to date tested and supported using OpenStack Kilo (in the ZHAW SDN testbed) and OpenStack Juno (in the Demokritos testbed).

To deploy the SDK in the Athens T-NOVA Pilot, a designated SDN NFVI-PoP was used, as shown in . The PoP consisted of three OpenStack nodes (Juno), one node reserved for the SDN controller and one physical switch.

To be able to use SDK4SDN the network needs to be fully SDN enabled through the OVS switches. The network interfaces of the switch are connected with the interfaces of the OpenStack and the ODL nodes.

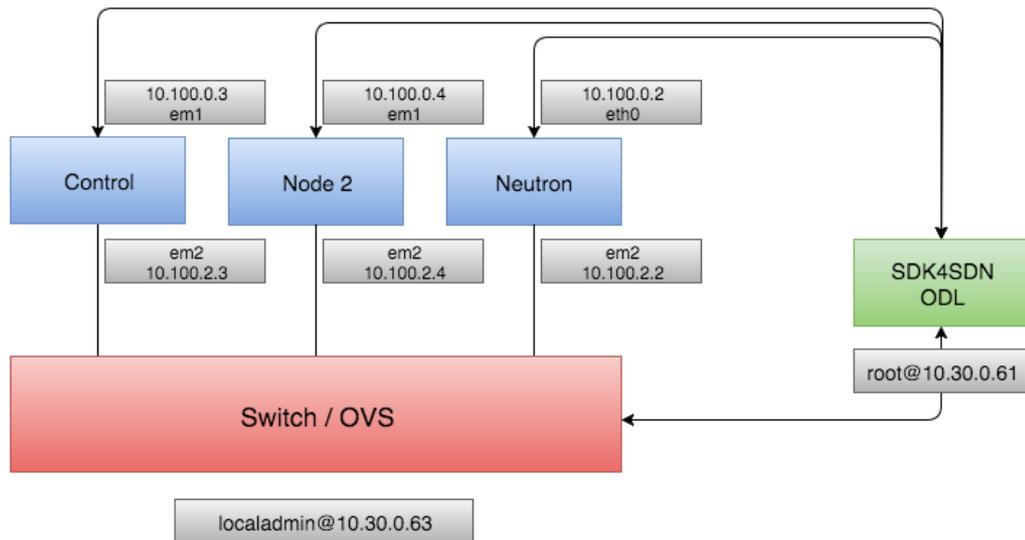
Athens T-NOVA Pilot
NFVI-PoP 2 SDN

Figure 18 OpenStack and OpenDaylight nodes in Athens testbed

Firewall Configuration on Neutron Node

The default SFC implementation requires disabling iptables rules that would prevent non-standard forwarding to OpenStack instances.

This should be edited on the cloud control and compute node in the file */etc/neutron/plugins/ml2/ml2_conf.ini*:

```
[securitygroup]
enable\_security\_group = False
firewall\_driver = neutron.agent.firewall.NoopFirewallDriver
```

To stop nova-compute from creating the iptables rules, it was configured to use its Noop driver in */etc/nova/nova.conf*:

```
[DEFAULT]
security\_group\_api = nova
firewall\_driver = nova.virt.firewall.NoopFirewallDriver]]
```

Finally, neutron-server, neutron-openvswitch-agent, nova-api and nova-compute services should be restarted.

Deployment and Installation

The SDK4SDN can only detect OVS topology. A Physical switch which supports OVS is for example the Pica8 switch in OVS mode. In the current integration, the Pica8 switch was substituted due to the following two reasons:

- Currently the ODL controller cannot be setup as Manager of the OVS on the Pica8 switch. It has been reported as current Pica8 issue in the community and it is under a process of resolution.
- The SDK requires a clear OVS setup and network state as it takes over the whole networking topology. The Pica8 switch in the PoP 2 had already some services attached to specific ports that if the SDK was attached, would had to be deleted.

The OVS configuration on the compute nodes has to be setup such that the switches are connected in one layer 2 domain. For each compute node, an OVS port was configured that bridges a physical interface. Note that if a compute host has a single physical interface then the IPs on that interface need to be attached to the internal interface of the OVS bridge. The following image shows the ovs setup of the SDN switch:

```
root@nfvipop2-switch:/home/localadmin/root@nfvipop2-switch:/home/localadmin# ovs-vsctl show
625107ec-0338-4d7e-8c12-d6e2422cc5fd
  Manager "tcp:10.30.0.61:6640"
  Bridge "br0"
    Controller "tcp:10.30.0.61:6633"
    Port "eth1"
      Interface "eth1"
    Port "eth2"
      Interface "eth2"
    Port "eth0"
      Interface "eth0"
    Port "br0"
      Interface "br0"
        type: internal
```

Figure 19 OVS configuration on SDN switch

The OpenDaylight controller needs to be reachable via TCP 6640 and TCP 6633 from every OVS and reachable via HTTP from the Orchestrator so it can accept the SFC instructions. After the deployment, the OpenDaylight controller can be started.

The steps in order to setup the environment are the following:

1. Connect to the pilot environment using VPN
2. Login to all OpenStack nodes, the switch and ODL node as *root*
3. Download the SDK4SDN from the GitHub page (to be added to the TNova code repositories). Compile the code using: *mvn clean install*.
4. Configure the OVSs on each of the nodes to connect to the IP of the OpenDaylight controller as well as set up the ODL as manager on port 6640:

```
ovs-vsctl set-manager tcp:Controller_IP:6640
ovs-vsctl set-controller tcp:Controller_IP:6633
```

With these basic steps the SDK is ready to be started as long as we have the environment prepared for this. To make sure all the state is clear before running the SDK, the following steps and checkups are required:

- OVS running in all of the nodes (run *ovs-vsctl show* to confirm and also check the configuration of the OVS)
- SDK not running: *./karaf/target/assembly/bin/status*
- Cleanup OpenStack environment (faster on dashboard, URL: 10.100.0.3/horizon): delete all VMs, delete router interfaces, delete routers, and delete all networks.
- Source the admin file: *source keystone_admin*
- Make sure you delete the following directories:

```
rm -rf karaf/target/assembly/data
rm -rf karaf/target/assembly/journal
rm -rf karaf/target/assembly/snapshots
```

After the above steps start the SDN controller (the SDK):

```
./karaf/target/assembly/bin/start
```

To watch the logs on the SDN node run:

```
tail -f ./karaf/target/assembly/data/log/karaf.log
```

You should see that links get discovered, the initialization process etc. The SDK startup is done when the last log message from ODL displays: GraphListener. Next, go to the controller and start the OpenStack startup script that installs the basic networking components: `./quickneutron.sh`.

While creating the network elements in OpenStack, the SDK log displays how the neutron ports get detected and few seconds later, the same for the ovs ports. After that, the paths get detected and flows are pushed to the `br-int` of the controller. On the controller you can check the created ports and the flows: `ovs-vsctl show; ovs-ofctl dump-flows br-int`.

Service Function Chaining

From the dashboard, create three VMs: *sender* and *receiver* VM with CentOS and *tiny* flavor, connecting them to the internal private network; one VM destined for the dummy VNF (it can be created from snapshot `dummy_vnf`). After inserting the private keys, connect the VM to the private network. In the control node, create two neutron ports `vnf_in` and `vnf_out` and attach them to the dummy VNF VM:

```
neutron port-create vnf_in
neutron port-create vnf_out
nova interface-attach --port-id [PORT_ID_vnf_in] vnf
nova interface-attach --port-id [PORT_ID_vnf_out] vnf
```

The SDK log displays moreover the added new paths by creating VMs within the new subnet. Also now the broadcast and path flows get created. To check this, go to the node where a VM is spawned, and `ovs-vsctl show` to see the flows installed. Besides the NORMAL and the LLDP flows, you can see other flows: broadcast flows to the Router, the DHCP and the other VMs; flows for connections between the VMs.

After the VNF has the two interfaces assigned from the other two private networks, associate a floating IP and access the VNF. Meanwhile login via the console to the *sender* VM and ping the *receiver* VM to confirm connectivity.

Next step is to call the REST APIs via cURL in the command line of the ODL node in order to invoke the creation of the chain. For the API call, the neutron port IDs of the VMs need to be specified in order "*sender, vnf_in, vnf_out, receiver*" as a string of comma separated values. To retrieve the Neutron ports of the VMs, run on the controller: `neutron ports-list` and match the port-id with the IPs of both, the endpoints and the VNF. Finally, the POST API call looks as shown below:

```

curl -H 'Content-Type: application/json' -X POST -d '{"input": {"neutron-ports": "90ec8cae-2552-4f8d-84c8-60ala6092677,3383c8b1-2590-4ffd-9ba9-933682bd249f,ebd413b5-3f21-4df1-b968-028e6662b253,953e85b9-5309-4f6f-a75d-bc1edcd47f3d"}' --verbose -u admin:admin http://127.0.0.1:8181/restconf/operations/netfloc:create-service-chain

> POST /restconf/operations/netfloc:create-service-chain HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: 127.0.0.1:8181
> Accept: */*
> Content-Type: application/json
> Content-Length: 180
>
* upload completely sent off: 180out of 180 bytes
< HTTP/1.1 200 OK
< Content-Type: application/yang.operation+json
< Transfer-Encoding: chunked
< Server: Jetty(8.1.15.v20140411)
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
{"output":{"service-chain-id":"1"}}root@odl:

```

Figure 20 API POST call for service chain creation

After correct chain establishment, there is 200 OK response message from the controller. The *sender* and the *receiver* can no longer detect each other through ping communication. This shows that there is intermediate node (the dummy_vnf) blocking the chain. In order to get the traffic through, the VNF includes a flow:

```
ovs-ofctl add-flow [vnf_br] in_port=1,actions=output:2
```

Make sure of the correct mapping between the input and output ports and also that the two ports *eth1* and *eth2* are properly listed in the:

```
ovs-ofctl dump-ports-desc [vnf_br]
```

After this, confirm that the ping goes through again as shown in Figure 21.

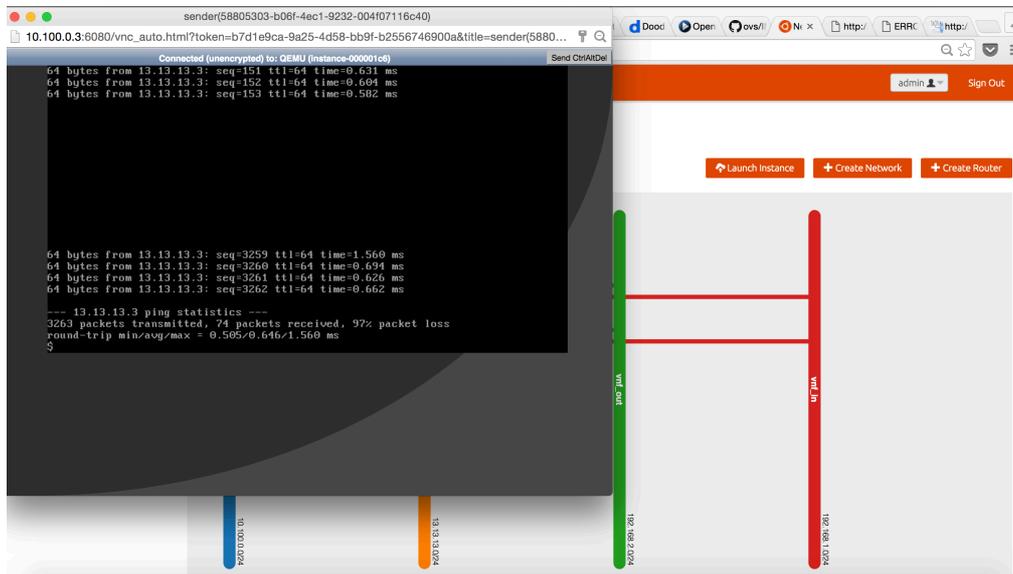


Figure 21 Sender VM console as ping to receiver resumes after the flow has been installed

It can be also checked by running: `tcpdump -i eth1 /eth2` on the node where the VNF is running and checking for ICMP messages, as the left terminal on the figure below shows:

```

root@vnf: /home/ubuntu
, length 64
07:56:50.610997 IP 13.13.13.2 > 13.13.13.3: ICMP echo request, id 24321, seq 48475
, length 64
07:56:51.611078 IP 13.13.13.2 > 13.13.13.3: ICMP echo request, id 24321, seq 48476
, length 64
07:56:52.611134 IP 13.13.13.2 > 13.13.13.3: ICMP echo request, id 24321, seq 48477
, length 64
07:56:53.611231 IP 13.13.13.2 > 13.13.13.3: ICMP echo request, id 24321, seq 48478
, length 64
07:56:54.363554 LLD, length 99: openFlow:233845178364757
07:56:54.611431 IP 13.13.13.2 > 13.13.13.3: ICMP echo request, id 24321, seq 48479
, length 64

root@network: /home/localadmin

root@Fvipop2-switch: /home/localadmin#
root@Fvipop2-switch:/home/localadmin# ovs-ofctl dump-flows br0 | grep priority=20
cookie=0x0, duration=72.473s, table=0, n_packets=34, n_bytes=3256, idle_age=1, priority=20,in_p
ort=3,dl_dst=04:01:ff:ff:ff:ff actions=output:1
root@Fvipop2-switch:/home/localadmin# ovs-ofctl dump-flows br0 | grep priority=20
cookie=0x0, duration=76.068s, table=0, n_packets=38, n_bytes=3648, idle_age=0, priority=20,in_p
ort=3,dl_dst=04:01:ff:ff:ff:ff actions=output:1
root@Fvipop2-switch:/home/localadmin# ovs-ofctl dump-flows br0 | grep priority=20
cookie=0x0, duration=77.561s, table=0, n_packets=39, n_bytes=3746, idle_age=1, priority=20,in_p
ort=3,dl_dst=04:01:ff:ff:ff:ff actions=output:1
root@Fvipop2-switch:/home/localadmin# ovs-ofctl dump-flows br0 | grep priority=20
cookie=0x0, duration=78.730s, table=0, n_packets=41, n_bytes=3942, idle_age=0, priority=20,in_p
ort=3,dl_dst=04:01:ff:ff:ff:ff actions=output:1
root@Fvipop2-switch:/home/localadmin# ovs-ofctl dump-flows br0 | grep priority=20

```

Figure 22 ICMP messages in the dummy VNF (left) and SFC flows on the switch (right)

This dummy VNF represents a simplistic case of the T-Nova vTC VNF used to steer traffic from one interface *eth1* and send it to the other *eth2* interface of the VNF VM.

Run on the switch terminal: `ovs-vsctl dump-flows [br_name]` to see if the *packets_in* counter increases as the pinging proceeds (for the flows with *priority=20*, i.e. the chain flows, as the right figure shows). Delete the flow in the VNF in order to see that the ping stops: `ovs-ofctl del-flows vnf`.

To DELETE the chain, a POST call is made using the *service-chain-id* as input parameter in the JSON string:

```
curl -H 'Content-Type: application/json' -X POST -d
'{"input": {"service-chain-id": "1"}}' --verbose -u
admin:admin http://127.0.0.1:8181/restconf/operations/netfloc:delete-service-chain
```

The design and the development details of the SDK4SDN can be found in the T-NOVA Deliverable [D4.31].

3.2.4. Monitoring Framework

With regard to the installation and deployment of the VIM monitoring back-end, for the convenience of the end-users, an official Docker image is provided. The official Docker image is based on a minimal Node.js Docker image. It was selected due to its small size: it does not exceed 40 MB. This image is built on Alpine Linux and contains additionally just the node as a static binary with no npm.

Thus, in order to deploy the VIM monitoring back-end, it is essential to deploy the necessary Docker containers and to configure accordingly the monitoring agents.

3.2.4.1. InfluxDB

The monitoring back-end requires an InfluxDB instance to host the monitoring data. The Docker container by Tutum¹ is used in the T-NOVA testing infrastructure and is highly recommended.

It is run with the following configuration:

```
docker run --name influxdb -d --restart=always \
  --env 'PRE_CREATE_DB=statsdb' \
  --env 'COLLECTD_DB=statsdb' --env 'COLLECTD_BINDING=:8096' \
  --volume /srv/docker/tnova_vim/tsdb:/data \
```

¹ <https://github.com/tutumcloud/influxdb>

```
--publish 8083:8083 --publish 8086:8086 --publish 8096:8096/udp \
tutum/influxdb:0.9
```

Listing 1: Configuration of InfluxDB docker container

The options to be set are:

--name influxdb: This is an identifier of the Docker container.

-d: This is to start the container in detached mode.

--restart=always: Always restart the container regardless of the exit status. This is to ensure starting the container during the Docker daemon start, in case the host restarts.

--env options: This is to ensure that a database (statsdb in the example) is created on the first time the container runs, that this database is used for storing collected data and that the InfluxDB is listening for collectd connections on the specified port.

--volume option: The only data volume is used here to persist the database files.

--publish options: The published port 8083 provides an HTTP user interface, port 8096 an HTTP API and port 8096 the collectd interface.

3.2.4.2. Monitoring Back-end

After setting up InfluxDB, the deployment of the back-end follows. The docker image is already available in dockerhub² and its deployment is achieved via:

```
docker run --name monitoring_backend -d --restart=always \
  --env 'CEILOMETER_HOST=localhost' --env 'CEILOMETER_PORT=8777' \
  --env 'NOVA_HOST=localhost' --env 'NOVA_PORT=8774' \
  --env 'IDENTITY_HOST=localhost' --env 'IDENTITY_PORT=5000' \
  --env 'IDENTITY_TENANT=tenant' \
  --env 'IDENTITY_USERNAME=username' --env 'IDENTITY_PASSWORD=pass' \
  --link influxdb:influxdb \
  --publish 8080:3000 \
  spacehellas/tnova-vim-backend:latest
```

Listing 2: Back-end deployment

The docker run command above had the following options:

--name monitoring_backend: This is an identifier of the Docker container.

-d: This is to start the container in detached mode.

--restart=always: Always restart the container regardless of the exit status. This is to ensure starting the container during the Docker daemon start, in case the host restarts.

--env options: The environment variables are explained in the next section.

--link influxdb:influxdb: This option links the back-end container with the InfluxDB one. Docker bridges this way the two containers automatically and the back-end container can detect which ports InfluxDB listens to.

² <https://hub.docker.com/r/spacehellas/tnova-vim-backend/>

`--publish 8080:3000`: The back-end application listens to port 3000 inside the container.

Table 5 below identifies the environment variables which need to be set for the proper communication with Openstack.

Table 5 Environment variables for the communication with Openstack

Name	Description
CEILOMETER_HOST	Defines the host of the OpenStack Ceilometer service
CEILOMETER_PORT	Defines the port of the OpenStack Ceilometer service
NOVA_HOST	Defines the host of the OpenStack Nova service
NOVA_PORT	Defines the port of the OpenStack Nova service
IDENTITY_HOST	Defines the host of the OpenStack Identity (Keystone) service
IDENTITY_PORT	Defines the port of the OpenStack Identity (Keystone) service
IDENTITY_TENANT	Defines the OpenStack tenant's name
IDENTITY_USERNAME	Defines the OpenStack username
IDENTITY_PASSWORD	Defines the OpenStack password

3.2.4.3. Grafana (optional)

Although not an essential component for the proper operation of the monitoring framework and the implementation of the use cases, Grafana can be installed as follows in order to visualize the monitoring data:

```
docker run --name grafana -d --restart=always \
  --publish 3000:3000 \
  grafana/grafana:latest
```

Listing 3: Grafana Deployment

3.2.4.4. Monitoring agents configuration

The VIM Monitoring Back-End uses *collectd* to collect the VNF instance monitoring data. Most importantly, it requires the *collectd* network plugin³ to be setup against the InfluxDB in order to send monitoring data:

```
<Plugin network>
  Server "<influxdb_host>" "<influxdb_port>"
  ReportStats false
</Plugin>
```

Listing 4: Configuration of collectd network plugin

`influxdb_host` and `influxdb_port` are the hostname and the port that are set previously for the InfluxDB Docker container.

³ <https://collectd.org/wiki/index.php/Plugin:Network>

3.3. WAN Infrastructure Connectivity Manager (WICM)

As the WICM component is not part of the integrated Pilot yet, this subsection presents a tutorial on the deployment and configuration of the WICM standalone component. The code for WICM is available as an opensource and available at the public github repository.⁴

The following tutorial is composed by two parts. The first presets the steps required to install WICM and the second explains how to setup the test environment for WICM. This tutorial has been tested on an Ubuntu 14.04 Linux machine. However, WICM may also work in other Linux distributions.

3.3.1. WICM Installation

- 1) Log in the machine where the WICM is being deployed.
- 2) Download the WICM code found in the T-NOVA repository and extract the code

```
tar -zxf compressed_file.tar.gz
```

- 3) Make sure the system is up to date:

```
apt-get update && apt-get upgrade -y
```

- 4) Install the following packages: *python-dev*, *python-mysqldb*, *python-pip*:

```
apt-get install python-dev -y  
apt-get install python-mysqldb -y  
apt-get install python-pip -y
```

- 5) Install the required python libraries:

```
pip install flask  
pip install Flask-SQLAlchemy  
pip install requests  
pip install MySQL-python  
pip install SQLAlchemy
```

- 6) Install mysql database:

```
apt-get install mysql-server -y
```

- 7) Create the WICM's database:

```
set the correct credential settings for the database in the file  
path/to/wicm/mysql/create_DB.sh
```

- a. DATABASE_NAME - Name of the database to be used by WICM
- b. DBUSER - WICM's database user
- c. DBUSERPASS - WICM's database password
- d. DBPASSWD - mysql root password

- 8) create the database by running:

```
./path/to/wicm/mysql/create_DB.sh
```

⁴ <https://github.com/T-NOVA/WICM>

- 9) Update the WICM file with the correct credentials
 - a. port - local port where WICM is listening
 - b. ip - local ip
 - c. odl_location - opendaylight's location
 - d. odl_auth - opendaylight's credentials (user, password)
 - e. mysql_connect - mysql connection string
- 10) Start WICM:

```
python ./path/to/wicm/main.py
```

- 11) Reset the database:

```
curl -X DELETE wicm_ip:wicm_port/reset_db
```

3.3.2. WICM demonstration

The WICM controls an OVS to redirect traffic to an external NFVI-POP and then receive the processed traffic and forward it to its original destination. This section shows how to build an environment to showcase the WICM.

The test environment is composed by 3 nodes: the first running an Openstack cluster (called Biker in this document) is used to simulate the NFVI-POP where the service chain is to be put in place. Traffic is going to be redirected into this machine in order to be processed. Next there is the node (called Alex) running OpenDaylight, the WICM, OVS and the traffic generators. WICM controls the OVS via OpenDaylight. The traffic generators produce the traffic to be redirected (customer traffic in a real scenario). Biker and Alex are two physical machines, connected over an Ethernet connection. Finally, a third machine is used in order to run the test script, which will act as a client and show WICM's functionality on a step by step fashion.

Figure 23 illustrates the demo setup. The blue and red dotted lines represent the traffic flows – while the red traffic flows directly between the aggregator switches br-ce and br-pe, the blue traffic is redirected to the NFVI-PoP and passes through the VNF located there before being sent to the intended destination.

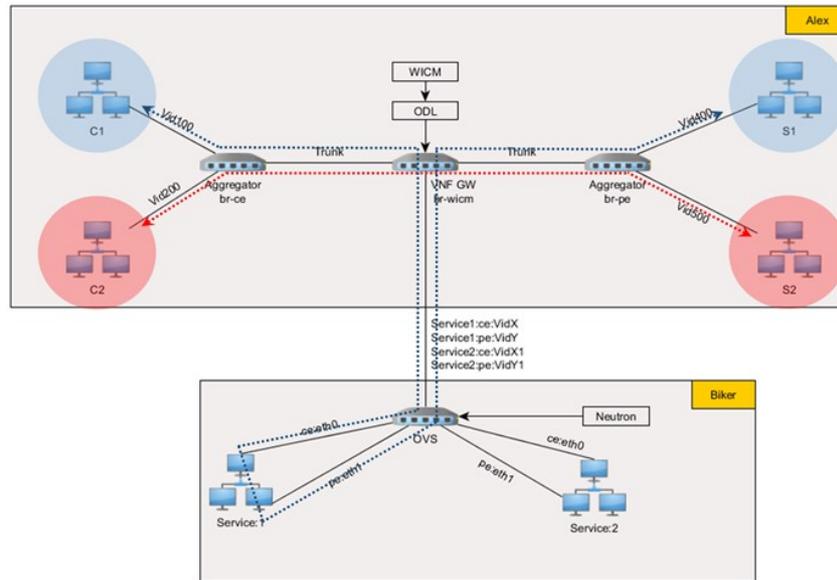


Figure 23 - WICM demo setup

3.3.2.1. Node 1 - Biker

Installation steps for OpenStack are out of the scope of this tutorial. The machine is expected to have a file called `admin-openrc.sh` containing OpenStack Keystone authentication credentials on the home directory.

3.3.2.2. Node 2 - Alex

Start by installing the OpenVirtual Switch. A version equal or better than 2.3.2 is required:

```
apt-get install ubuntu-cloud-keyring
echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" "trusty-
updates/kilo main" > /etc/apt/sources.list.d/cloudarchive-kilo.list
apt-get update && apt-get install openvswitch-switch -y
```

Then, download OpenDaylight:

```
Wget
https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/distribution-karaf/0.3.2-Lithium-SR2/distribution-karaf-0.3.2-Lithium-SR2.tar.gz
tar -zxvf distribution-karaf-0.3.2-Lithium-SR2.tar.gz
rm distribution-karaf-0.3.2-Lithium-SR2.tar.gz
```

1. Select the required ODL features for WICM by setting `featuresBoot` variable in file `org.apache.karaf.features.cfg`

```
cd path/to/odl/etc
sed -i \
's/featuresBoot=config,standard,region,package,kar,ssh,management/featuresBo
ot=config,standard,region,package,kar,ssh,management,odl-restconf-all,odl-
dlux-all, odl-mdsal-apidocs, odl-openflowplugin-all/g'
org.apache.karaf.features.cfg
```

2. ODL requires java7 to run. To install it issue the following commands:

```
apt-get update && apt-get upgrade -y
apt-get update && apt-get install openjdk-7-jre -y
```

3. Start ODL:

```
./path/to/odl/bin/start clean
```

4. Check if ODL is running:

```
./path/to/odl/bin/status
```

Setup the traffic generators:

1. Install Vagrant:

```
apt-get install dpkg-dev virtualbox-dkms
wget https://releases.hashicorp.com/vagrant/1.8.0/vagrant_1.8.0_x86_64.deb
dpkg -i vagrant_1.8.0_x86_64.deb
```

2. Prepare the network:

```
./path/to/wicm/test_env/init_network.sh
```

3. Start the vagrant machines

```
cd ./path/to/wicm/test_env
vagrant up
```

Finally, deploy the WICM, if not already deployed on node Alex.

3.3.2.3. Node 3 - Tester

- 1) Log in the machine
- 2) Extract the code

```
tar -zxf compressed_file.tar.gz
```

- 3) Make sure the system is up to date:

```
apt-get update && apt-get upgrade -y
```

- 4) Install the following packages: *python-dev*, *python-pip*

```
apt-get install python-dev -y
apt-get install python-pip -y
```

- 5) Install the required python libraries:

```
pip install prettytable
pip install pycrypto
pip install ecdsa
pip install fabric
```

- 6) Add the host names of the two other nodes (change the ips to fit your system):

```
echo "192.168.92.184 biker" >> /etc/hosts
echo "192.168.92.206 alex" >> /etc/hosts
```

- 7) Run the test script:

```
cd ./path/to/wicm/test_env/
python main_test.py
```

An extended description of the WICM component can be found in Deliverable [D4.21].

4. ORCHESTRATION LAYER DEPLOYMENT

4.1. TeNOR

TeNOR, T-NOVA's Orchestrator, has adopted micro-services based architecture, in which the overall features have been split into small and very simple modules, with well-defined interfaces. Furthermore, these modules communicate between them and with the remaining external modules through a REST API.

This section describes the script files that have to be executed to install TeNOR, with the exception of three of those sub-modules, which installation is described in its own sections:

- The Infrastructure Repository;
- The Service Mapping;
- The Gatekeeper.

4.1.1. Main Installation file

The main installation file is `install.sh`, shown in Listing 5.

```
#!/bin/bash
echo "Bundle install of each NS Module"
RAILS_ENV=development
for folder in $(find . -type d -name "orchestrator_ns*"); do
    echo $folder
    cd $folder
    bundle install
    cd ../
done
echo "\nConfigure NS modules"
for folder in $(find . -type d -name "orchestrator_ns*"); do
    echo $folder
    cd $folder
    cp config/config.yml.sample config/config.yml
    [ -f config/database.yml.sample ] && cp config/database.yml.sample
    config/database.yml
    [ -f config/mongoid.yml.sample ] && cp config/mongoid.yml.sample
    config/mongoid.yml
    cd ../
done
echo "Bundle install of each VNF Module"
for folder in $(find . -type d \( -name "orchestrator_vnf*" -o -name
"orchestrator_hot*" \) ); do
    echo $folder
    cd $folder
    bundle install
    cd ../
done
echo "\nConfigure VNF modules"
for folder in $(find . -type d \( -name "orchestrator_vnf*" -o -name
"orchestrator_hot*" \) ); do
```

```

echo $folder cd $folder cp config/config.yml.sample
config/config.yml [ -f config/mongoid.yml.sample ] && cp
config/mongoid.yml.sample config/mongoid.yml
cd ../
done

```

Listing 5: TeNOR's main modules installation (file install.sh).

This is a fairly simple 'install file', taking advantage of using a simple convention for naming the directories where the several micro-services could be found and the tools associated with the Ruby programming language used in the implementation (i.e., the `build` command in the file).

4.1.2. Cassandra Installation

Cassandra, the column-oriented NoSQL database used to store monitoring data, is installed by executing file `installation_cassandra.sh`, shown in Listing 6.

```

#!/bin/bash
#sudo apt-get install openjdk-7-jre
wget http://apache.forthnet.gr/cassandra/2.2.4/apache-cassandra-2.2.4-bin.tar.gz
tar -zxvf apache-cassandra-2.2.4-bin.tar.gz
#edit config file
#start_rpc: true => line 445
#rpc_address: 172.16.6.29 => line 475
nano apache-cassandra-2.2.4/conf/cassandra.yaml
#load schema
apache-cassandra-2.2.4/bin/cqlsh localhost 9042 -f ns_schema.txt
#start cassandra
apache-cassandra-2.2.4/bin/cassandra

```

Listing 6: Installing Cassandra (installation_cassandra.sh).

The first command in Listing 6 is required to install the Java Runtime Environment.

4.1.3. LogStash Installation

LogStash is used for logging all services, and is installed by executing file `installation_logstash.sh`, shown in Listing 7.

```

#!/bin/bash
#sudo apt-get install openjdk-7-jre
wget https://download.elastic.co/logstash/logstash/logstash-1.5.4.tar.gz
tar -zxvf logstash-1.5.4.tar.gz
logstash-1.5.4/bin/logstash agent -f logstash.conf
#elastic search
wget https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-1.7.2.tar.gz tar -zxvf elasticsearch-1.7.2.tar.gz
elasticsearch-1.7.2/bin/elasticsearch

```

Listing 7: Installing LogStash (file installation_logstash.sh).

4.1.4. MongoDB Installation

MongoDB, the document-oriented NoSQL database is used for storing VNF and NS Descriptors in JSON, and is installed by executing file `installation_mongodb.sh`, shown in Listing 8.

```

#!/bin/bash
# +-----+
# | Install MongoDB |
# +-----+
echo "Started installation of MongoDB"
# Import public key
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
# Create a list file
echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
# Reload local package database
sudo apt-get update
# Install the latest stable version
sudo apt-get install -y mongodb-org
# Change MongoDB configuration to accept external connections
sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mongod.conf
# +-----+
# | Disable Transparent Huge Pages (THP) |
# +-----+
echo "Disabling Transparent Huge Pages (THP)"
echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
## Create the init script to disable transparent hugepages (THP)
cat > /tmp/disable-transparent-hugepages << EOF
#!/bin/sh
### BEGIN INIT INFO
# Provides:          disable-transparent-hugepages
# Required-Start:    $local_fs
# Required-Stop:
# X-Start-Before:    mongod mongodb-mms-automation-agent
# Default-Start:     2 3 4 5 # Default-Stop:      0 1 6
# Short-Description: Disable Linux transparent huge pages
# Description:       Disable Linux transparent huge pages, to improve
#                   database performance.
### END INIT INFO
case $1 in
  start)
    thp_path=/sys/kernel/mm/transparent_hugepage
    echo 'never' > ${thp_path}/enabled
    echo 'never' > ${thp_path}/defrag
    unset thp_path
    ;;
esac
EOF
# Copy the init script to init.d folder
sudo mv /tmp/disable-transparent-hugepages /etc/init.d/disable-transparent-
hugepages
# Make it executable
sudo chmod 755 /etc/init.d/disable-transparent-hugepages
# Configure Ubuntu to run it on boot
sudo update-rc.d disable-transparent-hugepages defaults
echo "Restarting mongod service"
sudo service mongod restart
echo "Installation completed"

```

Listing 8: Installing MongoDB (file installation_mongodb.sh).

4.1.5. Micro-services registration

Executing the file `loadModules.sh`, shown in Listing 9, configures all TeNOR's internal micro-services.

```
#!/bin/bash
curl -XPOST http://localhost:4000/configs/registerService -H "Content-Type: application/json" -d '{"name": "nscatalogue", "host": "localhost", "port": 4011, "path": "/network-services"}' -H "X-Auth-Token: 504cec46-54e9-4ab1-8c72-ae9a72e5f36"
curl -XPOST http://localhost:4000/configs/registerService -H "Content-Type: application/json" -d '{"name": "nsdvalidator", "host": "localhost", "port": 4015, "path": "/nsds"}' -H "X-Auth-Token: 504cec46-54e9-4ab1-8c72-ae9a72e5f36"
curl -XPOST http://localhost:4000/configs/registerService -H "Content-Type: application/json" -d '{"name": "nsprovisioning", "host": "localhost", "port": 4012, "path": "/network-services"}' -H "X-Auth-Token: 504cec46-54e9-4ab1-8c72-ae9a72e5f36"
curl -XPOST http://localhost:4000/configs/registerService -H "Content-Type: application/json" -d '{"name": "nsmonitoring", "host": "localhost", "port": 4014, "path": "/network-services"}'
curl -XPOST http://localhost:4000/configs/registerService -H "Content-Type: application/json" -d '{"name": "vnfmanager", "host": "193.136.92.205", "port": 4567, "path": "/network-services"}'
#curl -X POST auth.piyush-harsh.info:8000/admin/service/ --header "Content-Type:application/json" --header "X-Auth-Token:79E9EEF8-FC4B-43FD-B1B2-4575D92864DC" -d '{"shortname":"vnfdpars","description":"this service parses the vnf descriptors"}'
```

Listing 9: Loading micro-services (file `loadModules.sh`).

In this we take advantage of having a uniform REST API for all micro-services.

4.1.6. Byobu Installation

Byobu, the text-based window manager and terminal multiplexer, used for executing the various instances of the component in their own terminal. The provided Listing 10. below allows to automatically spawn all the instances at once.

```
#!/bin/bash
SESSION='nsmanager'
SESSION2='vnfmanager'

# -2: forces 256 colors,
byobu-tmux -2 new-session -d -s $SESSION
# dev window
byobu-tmux rename-window -t $SESSION:0 'Mgt'
byobu-tmux send-keys "cd orchestrator_ns-manager" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:1 -n 'Catlg'
byobu-tmux send-keys "cd orchestrator_ns-catalogue" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:2 -n 'NSDV'
byobu-tmux send-keys "cd orchestrator_nsd-validator" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:3 -n 'Prov.'
byobu-tmux send-keys "cd orchestrator_ns-provisioner" C-m
```

```

byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:4 -n 'Ins.Repo'
byobu-tmux send-keys "cd orchestrator_ns-instance-repository" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:5 -n 'NSMon'
byobu-tmux send-keys "cd orchestrator_ns-monitoring" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:6 -n 'NSMon.Repo'
byobu-tmux send-keys "cd orchestrator_ns-monitoring-repository" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION:7 -n 'M-Mon'
byobu-tmux send-keys "cd orchestrator_ns-manager/default/monitoring" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux -2 new-session -d -s $SESSION2
byobu-tmux new-window -t $SESSION2:0 -n 'VNFMon'
byobu-tmux send-keys "cd orchestrator_vnf-monitoring" C-m
byobu-tmux send-keys "rake start" C-m
byobu-tmux new-window -t $SESSION2:0 -n 'VNFMon.Repo'
byobu-tmux send-keys "cd orchestrator_vnf-monitoring-repository" C-m
byobu-tmux send-keys "rake start" C-m
# Set default window as the dev split plane
byobu-tmux select-window -t $SESSION:0

```

Listing 10: Setting up all sessions using Byobu.

4.2. Infrastructure Repository

The section describes the initial integration of the Infrastructure Repository subsystem developed in task 3.2 and documented in deliverable D3.2 with other T-Nova components, deployed on the T-NOVA testbed hosted by NCSR D.

The Infrastructure Repository is designed to have an EPA Controller per NFVI PoP. This distributed architecture approach was adopted in order to provide scalability and performance in a multiple PoP scenario.

The Infrastructure Repository comprises of four primary components:

- **PoP Database** (1): a single graph database (DB) where the endpoints and the credentials for the services in each NFVI PoP are stored e.g. Keystone;
- **EPA Controller** (1-N): one controller per PoP;
- **Infrastructure Database** (1-N): one graph DB per PoP;
- **API Middleware Layer** (1-N): one or more API Middleware components to support scalability.

The flexibility of the infrastructure repository design supports different deployment scenarios i.e. singular or multiple NFVI-PoP configurations. To support early integration activities, a single PoP deployment has been realised on the NCSR D testbed. The deployment includes: one PoP DB, one infrastructure DB, an EPA Controller and an API Middleware.

To simplify the installation of the infrastructure repository a deployment script was developed based on a Python implementation:

```

from distutils.core import setup

setup(name='infrastructure_repo',

```

```

version='0.1.0',
description='Datacenter infrastructure repository',
license='Apache License v2',
keywords='EPA, Cloud Computing, Datacenter Software',
url='https://github.com/IntelLabsEurope/infrastructure-repository',
packages=['api', 'api.occi_epa',
          'api.occi_epa.backends',
          'api.occi_epa.extensions',
          'common', 'infrastructure_repository',
          'monitoring_service', 'monitoring_service.epa_database',
          'monitoring_service.epa_database.openstack'],
install_requires=['pika', 'py2neo==2.0.7', 'pyssf', 'networkx'],
scripts=['bin/infrastructure_repo', 'bin/infrastructure_repo_api'],
maintainer='Giuseppe Petralia',
maintainer_email='giuseppex.petralia@intel.com',
classifiers=["Development Status :: 3 - Alpha",
             "License :: OSI Approved :: Apache Software License",
             "Operating System :: OS Independent",
             "Programming Language :: Python",
             "Topic :: Internet",
             "Topic :: Scientific/Engineering",
             "Topic :: Software Development",
             "Topic :: System :: Distributed Computing",
             "Topic :: Utilities",
             "Topic :: System"
            ],
)

```

Listing 11 Infrastructure Repository deployment

All the infrastructure repository code is available from Intel Labs Europe's public github repository (<https://github.com/IntelLabsEurope/infrastructure-repository>) as shown in Figure 24.

A tool that can capture resource information from OpenStack or OpenDaylight deployments

Branch: master New pull request New file Find file HTTPS <https://github.com/IntelLabsEurope/infrastructure-repository> Download ZIP

peppetra Add support for different DBs credentials Latest commit 9cb298e an hour ago

File	Commit Message	Time Ago
api	Add infrastructure repository	6 hours ago
bin	Add infrastructure repository	6 hours ago
common	Add infrastructure repository	6 hours ago
config	Add check for enabled Openstack services	2 hours ago
epa_agent	Update agent.cfg	3 hours ago
infrastructure_repository	Add infrastructure repository	6 hours ago
monitoring_service	Add support for different DBs credentials	an hour ago
LICENSE	Initial commit	5 days ago
README.md	Update README.md	2 hours ago
requirements.txt	Add infrastructure repository	6 hours ago
setup.py	Add infrastructure repository	6 hours ago

README.md

Infrastructure Repository

Figure 24 Infrastructure Repository on ILE's Github

To install the repository the user needs to clone the repository locally and then execute the following steps:

4.2.1. Prerequisites

The following prerequisites must be installed and functioning before starting the installation of the infrastructure repository.

- OpenStack Liberty or Kilo release
- OpenDaylight Lithium Release
- Neo4j Database

4.2.2. EPA Controller

This component is responsible for collecting infrastructure related information and persisting it to a Neo4j DB. Infrastructure related information is collected by listening to OpenStack notifications, querying the OpenStack service DBs e.g. NOVA and listening to the agent's notifications queue (i.e. messages sent by EPA running on the compute nodes of NFVI PoP to indicate they have sent a data file for processing by the Controller).

4.2.2.1. EPA Controller Installation

To provide connectivity between the EPA Controller and OpenStack service database a Python MySQL Connector must be installed. Installation is as follows:

```
apt-get install python-mysqldb.connector
```

In the root directory:

```
pip install -r requirements.txt  
python setup.py install
```

4.2.2.2. Run EPA Controller

At installation time the EPA configuration file must be configured to set parameters such as credentials to access to OpenStack services DBs, to connect to RabbitMQ broker and to access the Neo4J DB as shown in the following extract of the configuration file:

```
[OpenstackDB]  
host=localhost  
nova_db_username=username  
nova_db_password=password  
...  
[RabbitMQ]  
rb_name=username  
rb_password=password  
rb_host=localhost  
rb_port=5672  
  
[EpaDB]  
epa_url = http://localhost:7474/db/data/  
epa_name = username  
epa_password = password
```

```
middleware_host_ip = localhost
```

Listing 12: Extract of the EPA Configuration File

A sample configuration is provided in `config/epa_controller.cfg`. The controller is run using the following command:

```
infrastructure_repo -c <path/to/configuration/file>
```

4.2.3. EPA Agent

An EPA Agent runs on each compute node in an NFVI PoP. The agent is responsible for collecting information about the compute node where it is running and for sending that information to the EPA controller. The agent should be launched after the Controller is up and running.

4.2.3.1. Installation

Copy the private key of the controller to machine compute where you want to run the agent. Install `hwloc`:

```
apt-get install hwloc
```

Provide the information required by the agent in the configuration file. A sample configuration can be found in `epa_agent/agent.cfg`. Install required packages:

```
pip install pika
```

4.2.3.2. Run EPA Agent:

```
python agent.py -c </path/to/the/configuration/file/>
```

4.2.4. API Middleware

The API middleware component exposes an OCCI compliant interface to consuming functions which require access to infrastructure related information stored in one or more the infrastructure repositories (each one called Point of Presence (PoP)).

4.2.4.1. API Middleware Installation

In the root directory:

```
pip install -r requirements.txt  
python setup.py install
```

4.2.4.2. Running API Middleware Component

Provide required information in a configuration file. A sample is provided in `config/middleware.cfg`.

Run the middleware with the following command:

```
infrastructure_repo_api -c <path/to/configuration/file>
```

4.2.4.3. Add a new PoP

To add a new PoP with the same name as the one used in the EPA Controller configuration file the following call is used

```
curl -X POST http://<MIDDLEWARE_IP>:<MIDDLEWARE_PORT>/pop/ --header
  "Accept: application/occi+json" --header "Content-Type: text/occi" --
  header 'Category: pop; scheme="http://schemas.ogf.org/occi/epa#";
  class="kind"'
-d 'X-OCCE-Attribute:
  occi.epa.pop.graph_db_url="http://usr:password@<NEO4J_IP>:7474/db/data/"
  X-OCCE-Attribute:
  occi.epa.pop.odl_url="http://<ODL_IP>:8181/restconf/operational/" X-
  OCCE-Attribute: occi.epa.pop.odl_name="admin" X-OCCE-Attribute:
  occi.epa.pop.odl_password="admin"' X-OCCE-Attribute:
  occi.epa.pop.name="GR-ATH-0001"
X-OCCE-Attribute: occi.epa.pop.lat=37.9997104 X-OCCE-Attribute:
  occi.epa.pop.lon=23.8168182'
```

To verify that the PoP has been correctly added to the repository make a GET call using the following url to see the available PoPs:

```
http://<MIDDLEWARE_IP>:<MIDDLEWARE_PORT>/pop/
```

If the installation is successful, the PoP should be included in the list. Copy the PoP ID and make a GET call to the following url to list the virtual machines currently running in the PoP and verify that the API Middleware is successfully connected to the infrastructure DB:

```
http://<MIDDLEWARE_IP>:<MIDDLEWARE_PORT>/pop/<POP_ID>/vm/
```

4.2.5. Resolved Issues

During the integration activities on the NCSR D testbed a number of issues were identified and resolved. The first one was related to integration with the new version of OpenStack, the Liberty release. The EPA controller queries the OpenStack service DBs to obtain the information about the virtual resources. The queries were designed to work with Kilo release. In the Liberty release the number of the DB schemas have changed. Therefore, queries related to retrieval of information with respect to Neutron ports, Nova Hypervisors and Heat Stacks were updated.

Another issue related to the fact that the Infrastructure Repository implementation assumed that the Keystone, Nova, Heat, Glance, Cinder and Neutron services were all active. In the current T-Nova testbed configuration the Cinder service is not used. For this reason new configuration parameters have been added to the Controller configuration file to permit to the PoP administrator to specify which services are enabled among those supported by the Controller.

4.3. Service Mapping

The Service Mapping package consists of a Rest service written in Ruby language and one or more solver applications. For the actual deployment in the NCSR D, the chosen solver is the one developed by Unimi and it consists of a custom C++ application.

This microservice is distributed as source code and, while the microservice code is interpreted on the target host by the Ruby interpreter, the C++ side of the application must be compiled on the target system.

The Service Mapping package has been deployed on a dedicated virtual machine into the testbed: the service does not need any special hardware requirement, so the basic m1.small flavor was enough to deploy and run the service. In details, the m1.small flavor consist of a virtual machine configured with a 1 VCPU, 2 Gigabytes of RAM memory and 20 Gigabytes of hard disk space.

Being quite a computing-intensive task, allocating more virtual CPU may increase the performance of the service.

The operating system installed on the virtual machine is Ubuntu 14.04 server.

Finally, the virtual machine has been configured so that it has both a local IP address and a floating IP address. The latter is not strictly mandatory: it is used for remote monitoring and updating during the test phase by logging in using tool such as PuTTY or a ssh shell.

Since different tasks are required to install the microservice on a vanilla environment, an install script has been developed to ease the installation process. The installer will be discussed later in this section while now the manual installation steps will be illustrated.

4.3.1. Manual Installation

After obtaining the code - whether by cloning the Service Mapper repository or by manually decompressing and copying the code archive - and copying it to a proper directory located into the home folder of the local user (from now we assume that the folder is `/home/servmapping/TeNOR-mapper`), the first step is to invoke the apt-get updater:

```
sudo apt-get update
```

This command synchronizes the local index of software packages from their sources.

Next step is to download and install the packages need by the microservice. Again, apt-get will automatically solve the dependencies by using the command:

```
sudo apt-get install -y make g++ ruby bundler zlib1g zlib1g-dev
```

The package "make" is a tool which controls the generation of executables and other non-source files of a program from the program's source files. It used during the compilation of the custom C++ application.

The package "g++" and all the associated tools and libraries (including GCC) is a C++ (and other languages as well) compiler and linker which builds the custom application C++ source code into object code and then links the latter into an executable program.

The package "ruby" and all the associated tools is a package which contains an interpreter, the development tools and all the related libraries/tools for the Ruby programming language. This software is used for the execution of the Rest service and the Thin web server.

The package "bundle" or "bundler" is a package whose function is to manage gem dependencies for the Ruby application so that eventual gems and all child dependencies specified in this manifest of Ruby application are automatically fetched, downloaded, and installed.

The packages zlib1g and zlib1g-devp are packages that contains zlib, a software library used for data compression. It solves a dependency during the compilation of the "nokogiri" Ruby gem.

After that, the next step is to solve all the Ruby gem dependencies for the Service Mapper Rest service. The package "Bundler" automatically solve this issue by downloading and configuring all the gems specified in the Service Mapper Rest service manifest file.

By moving to the Service Mapper application folder with:

```
cd /home/servmapping/TeNOR-mapper
```

and invoking Bundler with:

```
sudo bundle update
```

The following gems are downloaded, installed and configured:

```
Updating Ruby gems
Fetching gem metadata from https://rubygems.org/.....
Fetching gem metadata from https://rubygems.org/..
Resolving dependencies...
Installing rake (10.5.0)
Using addressable (2.3.8)
Using backports (3.6.7)
Using daemons (1.2.3)
Using unf_ext (0.0.7.1)
Using unf (0.1.4)
Using domain_name (0.5.25)
Installing eventmachine (1.0.9.1)
Using ffi (1.9.10)
Using http-cookie (1.0.2)
Using json (1.8.3)
Installing json-schema (2.6.0)
Installing rb-fsevent (0.9.7)
Using rb-inotify (0.9.5)
Using listen (3.0.5)
Using mime-types (2.99)
Using mini_portile2 (2.0.0)
Using multi_json (1.11.2)
Using netrc (0.11.0)
Installing nokogiri (1.6.7.1)
Using rack (1.6.4)
Using rack-protection (1.5.3)
Using rack-test (0.6.3)
Using rerun (0.11.0)
Using rest-client (1.8.0)
Installing tilt (2.0.2)
Using sinatra (1.4.6)
Using sinatra-contrib (1.4.6)
Using thin (1.6.4)
Using yard (0.8.7.6)
Using bundler (1.3.5)
Your bundle is updated!
```

Listing 13: Ruby gems required for the Service Mapper module installation

The following step is to install the GLPK package, and this operation may not be mandatory if the GLPK are already present on the host virtual machine.

Due to different licensing, this package cannot be embedded into the TeNOR Service Mapper code repository and its package source code is required to be manually downloaded from the official GLPK site. This can be done by using the wget utility:

```
cd ~
wget http://ftp.gnu.org/gnu/glpk/glpk-4.55.tar.gz
tar xvf glpk-4.55.tar.gz
```

and by using the tar utility to extract the GLPK source code file from the GLPK tar.gz archive.

Compiling and installing the GLPK library is straight-forward since the included configuration script and the makefile automatizes this process. By typing:

```
cd glpk-4.55
mkdir build
cd build
../configure
make
sudo make install
sudo ldconfig
```

the "libglpk.a" library is compiled and copied into the "/usr/local/lib" directory, and the include file "glpk.h" is copied in the "/usr/local/include" directory.

Finally, the command "ldconfig" updates the necessary links and cache to the most recent shared libraries found in the host virtual machine library directories.

Last step is to compile the custom C++ application package. A makefile has been provided as well, to ease this last operation.

By typing:

```
cd ~/TeNOR-Mapper/bin
make
```

the application will be compiled.

```
Compiling jsonConverter application
Building target: jsonConverter
Invoking: GCC C++ Compiler
g++ -I./include/ -O0 -g3 -Wall -c -fmessage-length=0 -std=c++11 -MMD -MP -o jsonConverter.o jsonConverter.cpp
Invoking: GCC C++ Linker
g++ -o "jsonConverter" jsonConverter.o
Finished building target: jsonConverter

Compiling solver application
Building target: solver
Invoking: GCC C++ Compiler
g++ -I./include/ -O0 -g3 -Wall -c -fmessage-length=0 -std=c++11 -MMD -MP -o solver.o solver.cpp
Invoking: GCC C++ Linker
g++ -o "solver" solver.o -lglpk
Finished building target: solver
```

Listing 14: Compiling the jsonConverter and solver applications

At the end of this step, the binary applications "jsonConverter" and "solver" will be created into the TeNOR-Mapper/bin folder.

The provided makefile will also clean the TeNOR-Mapper/bin directory from the compiled applications and intermediate object files with the command:

```
make clean
```

Also, the handy script `clear_workspace.sh` will delete old temporary work files from the TeNOR-Mapper/bin/workspace directory.

4.3.2. Automatic Installation

As mentioned earlier, an installer script is provided with the source code and automatizes the process hereby described, with the only exception of the GLPK libraries; the script does not download neither installs this package, but it only checks its availability on the host virtual machine. Therefore, this package must already be present (compiled and installed) on the host virtual machine before launching the installer script; however, if the installer fails to find the GLPK library on the system, the user can proceed with all the other installation steps (update of the apt-get cache, download of the other packages, creation of the "`~/TeNOR-Mapper`" directory, copy of the files, download and update of the Ruby gems) with the exception of the compilation of the C++ app, since it will most probably fail.

By default, the script installs the Service Mapper into the "`~/TeNOR-Mapper`" directory.

The installer script is started by typing:

```
./install.sh
```

Administration privileges may be necessary and queried as required.

At the end of the process, user will be greeted with the message:

```
Installation was successful.
Start the service by moving to /home/alessandro/TeNOR-Mapper
and typing:
rake start

By default, the service is listening to port 4042
and it can be changed by editing the config/config.yml file.
Exit the service with Control-C
```

Listing 15. Message on successful automated installation of the Service Mapper module

4.3.3. Service Mapper module configuration

The Service Mapper module does not require additional configuration, and it works out of the box. A few other optional steps may be required for custom configuration: as default, the service listens to the port 4042, but it can be changed by editing the "`~/TeNOR-Mapper/config/config.yml`" file.

Default route is "`http://localhost:4042/mapper`", and this can be changed by modifying the "`~/TeNOR-Mapper/routes/sm-unimi.rb`" file.

Optional parameters - such as multi-threading - can be passed to the Thin webserver by changing the "`~/TeNOR-Mapper/Rakefile`" configuration file.

4.3.4. Starting and stopping the Service Mapper module

The service is launched along with all the other T-NOVA services. However, it is possible to manually start the only Service Mapper by moving to the directory "~/TeNOR-Mapper" and typing:

```
rake start
```

The module can be stopped anytime, by pressing Control-C.

An extended description of the Service Mapper module can be found in Deliverable X [ref].

4.4. Gatekeeper

The Gatekeeper micro-service has been written in Go programming language. In order to deploy this service in the Athens T-NOVA Pilot, a VM of flavor *m1.small* was used in their OpenStack testbed. The service deployment is a straightforward process due to the fact that an installation script has been provided with the distribution. The steps that were followed were:

1. connect to the pilot environment using VPN
2. login to the gatekeeper VM
3. download the installation script from the T-NOVA code repositories⁵
4. execute the script

The script sets up the Go runtime, environment variables as well as recommended code dependencies in the VM.

```

configuring your environment for go projects ...
done.
testing new environment variables ...
GOPATH: /home/gatekeeper/go
Path: /home/gatekeeper/bin:/home/gatekeeper/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
Downloading auth-utils code now, please wait ...
Cloning into 'auth-utils'...
remote: Counting objects: 135, done.
remote: Total 135 (delta 0), reused 0 (delta 0), pack-reused 135
Receiving objects: 100% (135/135), 155.86 KiB | 53.00 KiB/s, done.
Resolving deltas: 100% (79/79), done.
Checking connectivity... done.
getting all code dependencies for auth-utils now, be patient ~ 1-2 minutes

```

Figure 25 Go runtime, environment and dependencies deployment

The configuration parameters that need to be configured before starting the service are shown below.

⁵ <https://github.com/T-NOVA>

```

19 ; Configuration file for gatekeeper auth-n/z service.
20
21 [gatekeeper]
22 port = 8000
23 dbfile = ./foo.db
24 logfile = ./auth-utils.log
25
26 [tnova]
27 defaultadmin = t-nova-admin
28 adminpassword = Eq7K8h9gpg

```

Listing 16 Gatekeeper Configuration

As shown in the configuration snippet above (Listing 16), the service runs on port 8000 and can be accessed by other T-NOVA services at this port using the floating IP assigned to this VM. Upon startup, the service performs a quick sanity checks and once DB integrity is verified it starts listening on the configured port for clients' requests.

```

gatekeeper@gatekeeper-2:~$ tail -f /var/log/tnova/gatekeeper/auth-utils.log
INFO: 2015/12/09 09:38:45 auth-utils.go:94: Table already exists in DB, nothing to do, proceeding normally.
INFO: 2015/12/09 09:38:45 auth-utils.go:124: Starting server on :8000
INFO: 2015/12/09 09:40:45 auth-utils.go:134: Received request on URI:/ GET
INFO: 2015/12/09 09:50:10 init.go:55: While performing DB sanity checks: found table user
INFO: 2015/12/09 09:50:10 init.go:55: While performing DB sanity checks: found table sqlite_sequence
INFO: 2015/12/09 09:50:10 init.go:55: While performing DB sanity checks: found table token
INFO: 2015/12/09 09:50:10 init.go:55: While performing DB sanity checks: found table service
INFO: 2015/12/09 09:50:10 auth-utils.go:94: Table already exists in DB, nothing to do, proceeding normally.
INFO: 2015/12/09 09:50:10 auth-utils.go:124: Starting server on :8000
INFO: 2015/12/09 09:52:19 auth-utils.go:134: Received request on URI:/ GET

```

Figure 26 Gatekeeper started and listening for requests

The details on how to use the service properly can be found in the T-NOVA Deliverable: Orchestrator Interfaces deliverable [D3.31].

4.4.1. Expression Solver (assurance formula evaluator)

T-NOVA orchestrator needs to continually evaluate the agreed SLA for any possible violations for all the deployed network services (NS). The agreed assurance formula is included in the NSD file which needs to be parsed, and evaluated with the actual component VNF's real-time monitored metrics. This micro-service provides REST APIs for the orchestrator to quickly evaluate any well formed assurance formula. The service is developed in Java using Jersey REST framework. The installation of this micro-service is straightforward as the installation and deployment scripts are included in the T-NOVA code repositories along with the source code. This service is co-located in the same VM as the Gatekeeper service. The installation steps involved were -

1. connect to the pilot environment through vpn
2. login to the gatekeeper VM (service was co-located with Gatekeeper)
3. download the installation scripts for exp-eval service from T-NOVA GIT repos
4. execute the script and follow the prompts

The configuration file parameters are self explanatory. Since the service has been collocated with Gatekeeper service, port 8888 was used while deploying.

```

22 dbfile=/Users/harh/Desktop/expeval.db
23 showexceptions=true
24 dbengine=sqlite
25 port=8000
    
```

Listing 17 Expression Solver configuration file

The service when started performs a quick sanity check and upon successful check starts to listen on the configured port for clients’ REST requests.

```

gatekeeper@gatekeeper-2:~$ java -jar /home/gatekeeper/bin/expressionsolver-0.1-jar-with-dependencies.jar
2015-12-09 08:46:09 DEBUG App:64 - Configuration file loaded properly.
2015-12-09 08:46:09 INFO Init:44 - Support variables have been initialized.
2015-12-09 08:46:09 INFO DBHelper:83 - Performing DB Sanity Checks now.
2015-12-09 08:46:09 INFO DBHelper:91 - Number of tables located: 0
2015-12-09 08:46:09 WARN DBHelper:101 - The following table: {template} was not found in the existing database!
2015-12-09 08:46:09 WARN DBHelper:101 - The following table: {expression} was not found in the existing database!
2015-12-09 08:46:09 WARN DBHelper:101 - The following table: {oplist} was not found in the existing database!
2015-12-09 08:46:09 WARN DBHelper:101 - The following table: {vnfinstance} was not found in the existing database!
2015-12-09 08:46:09 INFO DBHelper:108 - It is recommended to reinitialize the database before proceeding.
2015-12-09 08:46:09 INFO DBHelper:109 - I Will proceed to reinitialize the DB. Existing table contents will be lost.
2015-12-09 08:46:09 INFO DBHelper:131 - (Re)Created table: template
2015-12-09 08:46:09 INFO DBHelper:131 - (Re)Created table: expression
2015-12-09 08:46:09 INFO DBHelper:131 - (Re)Created table: oplist
2015-12-09 08:46:09 INFO DBHelper:131 - (Re)Created table: vnfinstance
2015-12-09 08:46:09 INFO DBHelper:136 - Database (re)initialized successfully!
Dec 09, 2015 8:46:10 AM org.glassfish.grizzly.http.server.NetworkListener start
INFO: Started listener bound to [0.0.0.0:8888]
Dec 09, 2015 8:46:10 AM org.glassfish.grizzly.http.server.HttpServer start
INFO: [HttpServer] Started.
2015-12-09 08:46:10 INFO App:85 - Jersey app started with WADL available at http://0.0.0.0:8888/exp-eval/application.wadl
Hit enter to stop it...
2015-12-09 08:46:10 INFO App:88 - System KPI parameters have been initialized.
2015-12-09 08:46:50 INFO Base:128 - URI:/ Method:GET Request processed.
    
```

Figure 27 Excerpt from Expression Solver log

Quick check using curl confirms that the service was deployed successfully.

```

CLT-MOB-T-6259:~ harh$ curl -X GET 10.10.1.63:8888/exp-eval/
{"api":[{"method":"GET","purpose":"capability discovery","uri":"/"},{"method":"GET","purpose":"list of registered expression te
mplates","uri":"/template/"}, {"method":"POST","purpose":"registration of a new expression template","uri":"/template/"}, {"me
thod":"GET","purpose":"get specific expression template details","uri":"/template/{id}"}, {"method":"DELETE","purpose":"delete
specific expression template, all expression instantiation will be deleted","uri":"/expression/{id}"}, {"method":"GET","purpose
":"list of registered expressions","uri":"/expression/"}, {"method":"POST","purpose":"register a new expression","uri":"/expre
sion/"}, {"method":"GET","purpose":"get specific expression details","uri":"/expression/{id}"}, {"method":"PUT","purpose":"upd
ate a specific expression","uri":"/expression/{id}"}, {"method":"POST","purpose":"execute a specific expression","uri":"/expre
sion/{id}"}, {"method":"DELETE","purpose":"delete a specific expression","uri":"/expression/{id}"}, {"method":"GET","purpose":
"get the service key runtime metrics","uri":"/kpi"}, {"op-supported":"lt, gt, add, max, min, avg","method":"POST","purpose":"on
the fly stateless expression evaluation","uri":"/otfly/"}], "src":"t-nova expression evaluation service", "msg":"api summary lis
    
```

Figure 28 Validation of the service deployment

The API guide for this service can be found in the T-NOVA internal wiki as well as in the upcoming WP3 Orchestrator deliverable D3.41.

As a result of the integration effort the following services deployment was achieved (Table 6):

Table 6 Gatekeeper Components

TNova Service	Floating IP	Port	Deployed?	Base-URI
Gatekeeper	10.10.1.63	8000	Yes	/

Expression Solver	10.10.1.63	8888	Yes	/exp-eval/
-------------------	------------	------	-----	------------

5. MARKETPLACE AND NF STORE DEPLOYMENT

5.1. NF Store

The Network Function Store has been written in Java programming language as a web application running on TomEE server, a tomcat server with java EE extensions.

In order to deploy the service on T-NOVA testbed hosted by NCSRD, a preliminary build of the project is needed.

The code is available into i2cat stash git repository <https://github.com/T-NOVA/NFS>.

The build produces an rpm that can be installed on server and then the NFSStore will be available as a standard Linux SysVinit service.

5.1.1. Build NF Store

- Operation System

The NFSStore build does not require a particular type of OS as the application and the server are realized in Java and the build will be done using Java Virtual Machine.

- Prerequisites

The following prerequisites must be installed and available before starting the build of the NF Store:

- [java virtual machine](#) (1.8 version)
- [ant](#)
- [git client](#)

1. Clone NFS repository

```
git clone http://\[i2cat\_username\]@stash.i2cat.net/scm/TNOV/wp5.git
```

2. Compile NF Store

```
$ cd wp5/WP5/NFS
$ ant
Buildfile: D:\WorkspaceGIT\wp5\WP5\NFS\build.xml
clean:
  [delete] Deleting directory D:\WorkspaceGIT\wp5\WP5\NFS\build
  [delete] Deleting directory D:\WorkspaceGIT\wp5\WP5\NFS\dist
init:
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\build\classes\META-INF
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\dist\var\db
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\dist\var\log
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\dist\var\run
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\dist\var\tmp
  [mkdir] Created dir: D:\WorkspaceGIT\wp5\WP5\NFS\dist\certs
server:
  [unzip] Expanding: D:\WorkspaceGIT\wp5\WP5\NFS\server\apache-tomee-1.7.1-
plus.zip into D:\WorkspaceGIT\wp5\WP5\NFS\dist
  [copy] Copying 8 files to D:\WorkspaceGIT\wp5\WP5\NFS\dist\apache-tomee-plus-
1.7.1\lib
  [copy] Copying 6 files to D:\WorkspaceGIT\wp5\WP5\NFS\dist\apache-tomee-plus-
1.7.1\conf
```

```

[copy] Copying 4 files to D:\WorkspaceGIT\wp5\WP5\NFS\dist\certs
[copy] Copied 1 empty directory to 1 empty directory under
D:\WorkspaceGIT\wp5\WP5\NFS\dist\certs
build:
[echo] NFS: D:\WorkspaceGIT\wp5\WP5\NFS\build.xml
[javac] Compiling 61 source files to D:\WorkspaceGIT\wp5\WP5\NFS\build\classes
[javac] warning: Supported source version 'RELEASE_6' from annotation processor
'org.apache.openjpa.persistence.meta.AnnotationProcessor6' less than -source
'1.7'
[javac] 1 warning
war:
[war] Building war: D:\WorkspaceGIT\wp5\WP5\NFS\prod\war\NFS.war
[copy] Copying 1 file to D:\WorkspaceGIT\wp5\WP5\NFS\dist\apache-tomee-plus-
1.7.1\webapps
rpm:
[build-rpm] Created rpm: nfs-1.0-0.noarch.rpm
BUILD SUCCESSFUL
Total time: 31 seconds

```

Listing 18 NF Store build command output

Now the rpm file to be deployed is available:

```

$ cd prod/rpms
$ ls *.rpm
nfs-1.0-0.noarch.rpm

```

5.1.2. Deploy NF Store

- Operation System

If the NF Store is deployed as a standard Linux SysVinit service then a host with Linux OS is necessary.

- Prerequisites

The following prerequisites must be installed and available before installation of the NFStore.

- [java virtual machine](#) (1.8 version)
 - [rpm](#)
1. Install built rpm (see previous steps)

```

$ ls *.rpm
nfs-1.0-0.noarch.rpm
$ rpm -i nfs-1.0-0.noarch.rpm
$ rpm -qa nfs
nfs-1.0-0.noarch

```

2. Verify installation - You can check that the rpm is installed by querying rpm and also checking that TomEE server is installed:

```

$ rpm -qa nfs
nfs-1.0-0.noarch
$ ls /usr/local/nfs
apache-tomee-plus-1.7.1 bin certs var

```

3. Customize NF Store configuration

The default configuration of NFStore is illustrated in Table 7:

Table 7 NF Store Configuration

Variable name	Default value	Description
NFS_STORE_PATH	/usr/local/store	local store directory
NFS_URL	https://api.t-nova.eu/NFS	NFStore URL used for set image links
ORCHESTRATOR_URL	https://api.t-nova.eu/orchestrator	orchestrator URL
TOMCAT_PROTOCOL	https	NFStore protocol
TOMCAT_IP	0.0.0.0	NFStore address
TOMCAT_HTTP_PORT	80	NFStore port when protocol is http
TOMCAT_HTTPS_PORT	443	NFStore port when protocol is https

The default values can be changed by setting the required values into the file **/usr/local/nfs/bin/nfs.conf** before starting the server.

This file already contains commented the default configuration for all listed variables so, if needed, uncomment and set the required values before starting the service, as shown in the following example:

```
#!/bin/bash
#####
# optional variable configuration to override default values
# uncomment and set required values before start nfs service
#####

#----- monitor log level
# LOG_LEVEL=notice

#----- store path
# NFS_STORE_PATH=/usr/local/store

#----- tomcat interface
TOMCAT_PROTOCOL=http
# TOMCAT_IP=0.0.0.0
TOMCAT_HTTP_PORT=8080
TOMCAT_HTTPS_PORT=8443

#----- url
ORCHESTRATOR_URL=http://193.136.92.205:4567/vnfs
NFS_URL=http://83.212.108.105:8080/NFS
```

Listing 19 NF Store Configuration File

5.1.3. Start NF Store

```
$ service nfs start
Starting nfs (via systemctl): [ OK ]
```

The NFStore status can be checked using service command:

```
$ service nfs status
nfsMonitor active: pid 29890
tomcat active: pid 29924
tomcat manager status: running
```

```
nfs app status: running
```

The service can be restarted or stopped using again service command:

```
$ service nfs restart
Restarting nfs (via systemctl): [ OK ]
$
$ service nfs stop
Stopping nfs (via systemctl): [ OK ]
$ service nfs status
nfsMonitor not active
tomcat not active
```

To configure the NFSStore service to restart automatically on system reboot, add these links into run level directory used by the server (run level 3 for the following example):

```
$ cd /etc/rc3.d
$ ln -s /etc/init.d/nfs S99nfs
$ ln -s /etc/init.d/nfs K99nfs
```

5.2. Marketplace

5.2.1. Prerequisites

- o Operation System

Marketplace supports multiple Linux distributions that support Docker. Currently Marketplace has been deployed and tested for Docker version in Ubuntu Trusty 14.04 Ubuntu Linux.

5.2.1.1. Installing Docker on Ubuntu

Docker requires the package apt-transport-https to be installed. This is included in the default Ubuntu-14.04 build.

Before you install Docker, you will need to add the Docker repository key to your keychain:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
36A1D7869245C8950F966E92D8576A8BA88D21E9
```

You should see a response similar to this:

```
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --
homedir /tmp/tmp.tNuh5zx96p --no-auto-check-trustdb --trust-model always -
-keyring /etc/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --
keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
36A1D7869245C8950F966E92D8576A8BA88D21E9
gpg: requesting key A88D21E9 from hkp server keyserver.ubuntu.com
gpg: key A88D21E9: public key "Docker Release Tool (releasedocker)
<docker@dotcloud.com>" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Listing 20 Apt repository GPG key installation

Next, update and install the lxc-docker package:

```
sudo sh -c "echo deb https://get.docker.com/ubuntu docker main >
/etc/apt/sources.list.d/docker.list"
sudo apt-get update
sudo apt-get install lxc-docker
```

Finally, test that Docker is working properly by using the following command:

```
sudo docker run -i -t ubuntu /bin/bash
```

5.2.1.2. Docker Compose

Docker Compose is an orchestration tool that automates the build of multi-container applications. With Compose, you define your application's components (containers, configurations, links, volumes) in a single file, then you can build your application with a single command that does everything that needs to be done to get your application running. Compose is great for development, testing and staging environments.

Using Compose is basically a three-step process:

- Define your applications environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up your application in docker-compose.yml so they can be run together in an isolated environment.
- Run docker-compose up and Compose will start and run your entire application.

The Marketplace's docker-compose.yml is illustrated in Listing 21:

```
1  dnsdock:
2  image: tonistiigi/dnsdock
3  hostname: dnsdock
4  restart: always
5  ports:
6  - 172.17.0.1:53:53/udp
7  volumes:
8  - /var/run/docker.sock:/var/run/docker.sock
9
10 mysql:
11 build: mysql/
12 environment:
13   MYSQL_USER: admin
14   MYSQL_PASS: ██████████
15   STARTUP_SQL: /tmp/init/*.sql
16 volumes:
17 - /data/mysql:/var/lib/mysql
18
19 mongodb:
20 build: mongodb/
21 volumes:
22 - /data/mongodb:/data/db
23 ports:
24 - 27017:27017
25 command: mongod --smallfiles
26
27 umaa:
28 build: umaa/
29 volumes:
30 - ./umaa/keys:/keys
31 links:
32 - mysql
33 dns:
34 - 172.17.0.1
35 environment:
36 - DNSDOCK_ALIAS=umaa.docker
```

Listing 21 Docker composition configuration file

In order to Install Docker Compose we follow the steps below:

- Install pip

Pip is a tool for installing Python packages.

```
sudo apt-get install python-pip python-dev build-essential
sudo pip install --upgrade pip
```

- Install Compose using pip

Compose can be installed from pypi using pip. If you install using pip it is highly recommended that you use a virtualenv because many operating systems have python system packages that conflict with docker-compose dependencies. Note: pip version 6.0 or greater is required.

```
sudo pip install docker-compose
```

5.2.2. Marketplace Deployment

Step 1 - Clone marketplace repository

Download the Marketplace source code by cloning it from the repository:

```
$ cd ~  
$ git clone http://[i2cat_username]@stash.i2cat.net/scm/TNOV/wp6.git  
$ cd wp6/marketplace
```

Step 2 - Build Cyclops base image

Because of the high number of dependencies need to build the cyclops-base separately. The cyclops image depends on ubuntu:14.04 standard docker image.

```
$ sudo docker build -t cyclops-base cyclops/docker-files/cyclops-base/.
```

Step 3 - Build Marketplace

Once the cyclops-base image is created, then proceed to build the application by running the docker-compose build command.

```
$ sudo docker-compose up
```

Step 4 - Add the marketplace domain to hosts file

The marketplace works only by visiting the domain marketplace.t-nova.eu, so you need to add this domain to hosts file in order to server the local instance of marketplace.

```
sudo echo "127.0.0.1 marketplace.t-nova.eu" >> /etc/hosts
```

Step 5 - Access Marketplace

Visit <http://marketplace.t-nova.eu> to ensure the deployment was successful.

6. OVERVIEW OF T-NOVA DEPLOYMENT

This section presents a visual overview of the T-NOVA deployed components as is currently available in Athens Pilot infrastructure. In addition it provides an overview of a T-NOVA tenant (customer of the SP) and a plausible deployment scenario of Network Services and VNFs.

In order to co-host the actual T-NOVA components in the same infrastructure with the actual NS components we choose to reuse the same infrastructure. Although in reality this might not be a proper selection especially in a production environment, it is imposed by the availability of IT resources of the Pilot and the scale of infrastructure hosted.

In order to isolate the components, a separate tenant is used for the deployment of T-NOVA Orchestration, Marketplace and VIM/WICM components. Figure 29 illustrates the deployment of these components as visualised by the Openstack Dashboard.

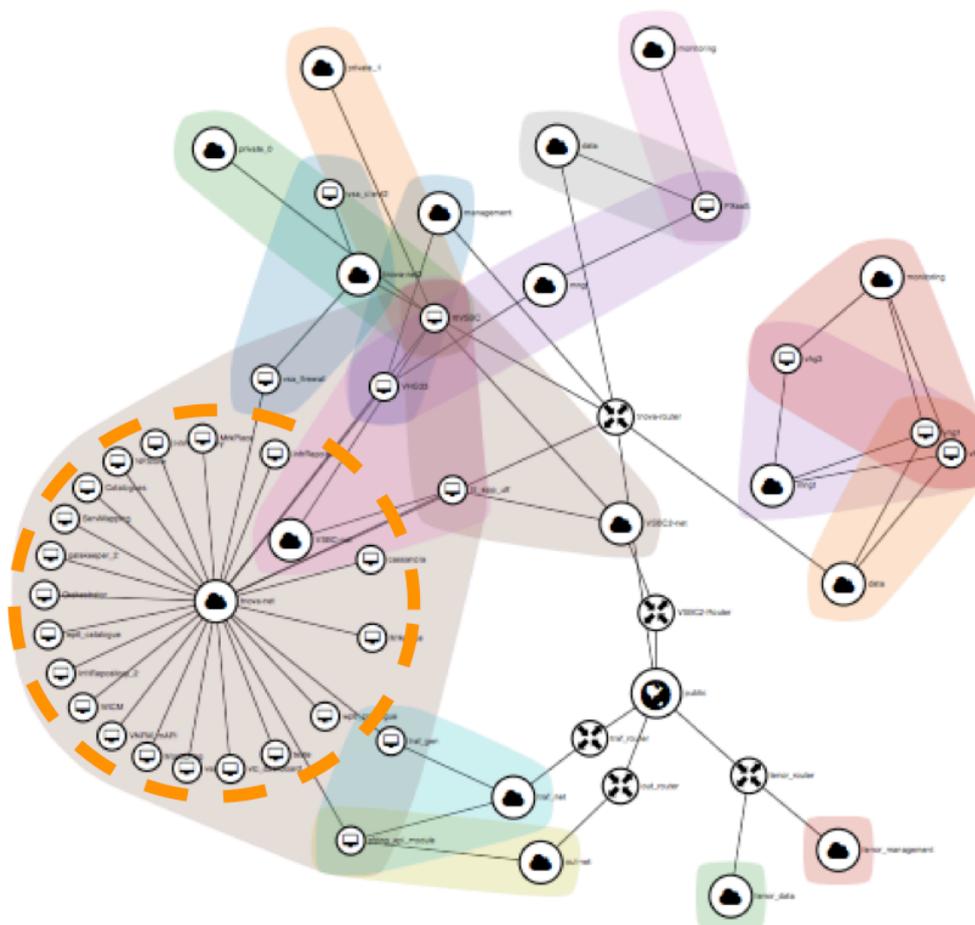


Figure 29 Component network topology

The orange dashed circle denotes the VMs that host the T-NOVA deployed components. The rest icons denote, example VNF deployments and a number of virtual networks used for their deployment.

A second, dedicated tenant account (corresponding to a T-NOVA customer tenant) is used by the VNF developers during the integration of T-NOVA in Athens pilot to deploy NFV service components.

7. CONCLUSIONS

The present deliverable served the purpose of a technical report accompanying the prototype deliverable i.e. the early version of the T-NOVA pilot. It can be also used as a technical guide for deploying all of part of the T-NOVA system.

At present, all components and subsystems of the T-NOVA architecture (in their current versions) have been deployed and integrated in the primary pilot testbed in NCSR premises in Athens. Currently, all partners are engaged in an iterative continuous integration procedure, in which updated versions of the modules are integrated into the testbed and validated.

As soon as the pilot reaches an acceptable maturity level, most of the components will also be deployed to the rest pilots and inter-pilot scenarios will be experimented upon. These advances will be included in the second edition of this deliverable (D2.52).

8. REFERENCES

- [cacti] Cacti: the complete rrdtool graphing solution. On-line: <http://www.cacti.net>
- [D2.51] G. Xilouris (ed.) et al, "Deliverable 2.51 - Planning of trials and evaluation – Interim", T-NOVA Project,
- [D2.52] G. Xilouris (ed.) et al, "Deliverable 2.52 - Planning of trials and evaluation – Final", T-NOVA Project,
- [D3.1] J. Bonnet (ed.) et al, "Deliverable 3.1 - Orchestrator Interfaces", T-NOVA Project, Sep. 2015
- [D3.2] M. McGrath (ed.) et al, "Deliverable 3.2 - Infrastructure Resource Repository", T-NOVA Project, Jul 2015.
- [D3.3] Deliverable 3.3-Service Mapping", T-NOVA Project, Dec 2015.
- [D3.41] Deliverable 3.41 - Service Provisioning, Management and Monitoring – Interim", T-NOVA Project, Dec 2015
- [D4.1] M. McGrath (ed.) et al, "Deliverable 4.1 - Resource Virtualisation", T-NOVA Project, September 2015.
- [D4.21] L. Zuccaro (ed.) et al, "Deliverable 4.21 - SDN Control Plane Interim", T-NOVA Project, November 2015.
- [D4.31] I. Trajkovska (ed.) et al, "Deliverable 4.31 - SDK for SDN Interim", T-NOVA Project, September 2015.
- [D4.41] G. Gardikis (ed.) et al, "Deliverable 4.41 - Monitoring and Maintenance – Interim", T-NOVA Project, November 2015.
- [D4.51] E. Trouva (ed.) et al, "Deliverable 4.51 - Infrastructure Integration and Deployment - Interim, T-NOVA Project, December 2015.
- [D5.1] N. Herbaut (ed.) et al, "Deliverable 5.1- Function Store", T-NOVA Project,
- [D5.31] P. Paglierani (ed.) et al, "Deliverable 5.31 - Network Functions Implementation and Testing", T-NOVA Project, Nov. 2015.
- [D6.1] A. Rammos (ed.) et al, "D6.1-Service Description Framework", T-NOVA Project, Dec 2015.
- [D6.2] E. Markakis (ed.) et al, "Deliverable 6.2-Brokerage Module", T-NOVA Project, Dec. 2015.
- [D6.3] E. Markakis (ed.) et al, "Deliverable 6.31-User Dashboard", T-NOVA Project, Dec. 2015.
- [D6.4] A. Ramos (ed.) et al, "Deliverable 6.41-SLAs and Billing", T-NOVA Project, Dec 2015.
- [Docker] Docker, on-line: <https://www.docker.com>
- [NFS] Network File System, on-line: <http://nfs.sourceforge.net>

[ODL] OpenDayLight, on-line: <http://opendaylight.org>

[Openstack] Openstack, on-line: <https://www.docker.com>

9. LIST OF ACRONYMS

Acronym	Explanation
CIFS	Common Internet File System
CPE	Customer Premises Equipment
COTS	Commercial of the shelf
FPGA	Field-programmable gate array
GRE	Generic Routing Encapsulation
IOMMU	I/O memory management unit
HDFS	Hadoop Distributed File System
LRDIMM	Load Reduced DIMM
LXC	Linux Containers
MD-SAL	Model-driven Service Abstraction Layer
ML2	Modular Layer 2
MM	Monitoring Manager
MVC	Model-View-Controller
NIC	Network Interface Controller
NFS	Network File System
NTP	Network Time Protocol
NUMA	Non-Uniform Memory Access
QPI	QuickPath Interconnect
ODL	OpenDaylight
OVS	Open vSwitch
PF	Physical Function
PCIe	PCI Express
POC	Proof of Concept
PXE	Preboot Execution Environment
RID	PCI Express Requestor ID
RDIMM	Registered Dual in-line Memory Module
RADOS	Reliable Autonomic Distributed Object Store
REST	Representational State Transfer
RPC	Remote Procedure Call

SDK4SDN	Software Development Kit for Software Defined Networking
SFC	Service Function Chaining
SMB	Server Message Block
SR-IOV	Single Root I/O Virtualisation
TFTP	Trivial File Transfer Protocol
UDIMM	Unregistered Dual in-line Memory Module
vHG	Virtual Home Gateway
vPaaS	Virtual Proxy as a Service
vSA	Virtual Security Appliance
vSBC	Virtual Session Border Controller
vTC	Virtual Transcoding Unit
VXLAN	Virtual Extensible LAN