SEVENTH FRAMEWORK PROGRAMME

TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D4.51

Infrastructure Integration and Deployment

Editor E. Trouva (NCSRD)

Contributors C. Sakkas, C. Xilouris, I. Angelopoulos (NCSRD), M. J. McGrath, V. Riccobene (Intel), L. Zuccaro , F. Cimorelli (CRAT), G. Gardikis, I. Koutras (SPH), I. Trajkovska, D. Baudinot (ZHAW), K. Karras (FINT)

Version 1.0

Date December 31st, 2015

Distribution PUBLIC (PU)

Executive Summary

This deliverable presents the procedures that were followed for the the deployment of the IVM components. Furthermore, lessons learnt from the deployment and integration efforts, as well as the interactions of the components are presented. A walkthrough is provided as a technical guide for the implementation of an NFVI testbed, aiming at those who intent to replicate a similar deployment.

First, we provide an overview of the T-NOVA NFVI and explain the rationale behind the technology choices selected in four different domains, hypervisors, computing storage and network. We describe the main roles that are available for the deployment of the IVM layer components and next, we list the components comprising the NFVI, including both infrastructure and T-NOVA specific components. A step-by step presentation of the procedures we used for the components deployment follows. We outline the interaction between the components and how their integration was accomplished.

Furthermore, we define the set of tools that were employed, to build a testing environment that validates the functionality and performance of a deployed T-NOVA IVM layer stack. We provide a set of tests that verify the function of the deployed components in order to support users who wish to successfully deploy the T-NOVA IVM testbed. A testing dashboard application was developed to ease the testing procedures, allowing automated installation of the testing environment, execution of the tests and presentation of the results in a graphical way.

The conclusions gathered from the activities within the work conducted in T4.5 and a summary of the next goals to be accomplished over the remaining duration of task are outlined. Finally, we provide a number of annexes with detailed guidelines and instructions on the steps involved in the deployment and integration of the T-NOVA IVM layer components and their required configuration to produce to a functional and performant testbed.

Table of Contents

Index of Figures

Index of Tables

1. INTRODUCTION

1.1. Motivation, Objectives and Scope

The primary focus of Task 4.5 is the implementation of a testbed comprised the implementations of the functional entities of the T-NOVA IVM layer namely the NFVI, VIM and WICM. 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. The integration of these assests into successful IVM layer is a key technical milestone in support WP7 which is tasked with the overall plan for the validation and assessment of the T-NOVA system from an end-to-end system wide use case prespective. The activities within the task are informed by the key outputs from tasks 4.1, 4.2, 4.3 and 4.4.

The scope of this task encompasses a number of key activities. Firstly, the Task 4.5 will leverage Tasks 4.1-4.4 to identify the technologies to be used in the implementation of the functional entities. The tasks provide the necessary hands on knowledge and best known methods (BKMs) for the deployment and configuration of the selected technology components to achieve a stable and reproducible deployment for the testbed. The technology innovations from the tasks will be deployed and validated on the testbed. The testbed will then be used to integrate the technology innovations from the tasks in order to provide an end-to-end solution to support the deployment of VNFs within the IVM. To validate both the components and overall testbed system validation and functional test cases will be defined and executed in this task. The functional tests cases will focus on determine if the IVM testbed has the correct functionality to support the associated requirements. The validation test cases will determine if the IVM fulfils its necessary goals within the context of the overall T-NOVA system. Key characteristics to be validated are reliability, availability, configurability and scalability. Validation will focus on the key technology comprising the NFVI and VIM such as OpenStack, OpenDaylight etc. Additionally, the validation activities will have a workload and service perspective. Therefore, tests associated with the deployment of VNF and networks services will be identified and executed. In order to support the test regime appropriate test tools will be identified and utilised. For example, OpenStack Rally and Tempest could be used to validate the deployment of OpenStack and its services. The selection of appropriate open source tools will ensure that testing can be implemented in a reproducible manner. The use of open source test tools where support easier testing of the T-NOVA system components by third party adopters.

The overall objective for this task is the integration and deployment of the components identified or developed in supporting tasks into a composite network/cloud programmable infrastructure. Supporting this overall objective are number of sub-objectives including:

- Validation of the SDN Control Plane
- Validation of VM interconnections and virtual networking functions.

- Validation of the NFVI resources (physical compute and network resources)
- Identification and execution of functional and validation test cases including:
  - VNF functional deployment
  - Network service deployment.
  - Infrastructure metrics capture and exposure
- Identification of IVM deployment patterns and mechanisms templates for VNF deployments.
- Generation of technical documentation that describes the IVM Layer implementation, components, supported stacks, installation procedures, user interfaces, APIs, and configuration files.

1.2. Relationship, Inter Task Dependencies and Relevant Documents

The implementation of a testbed comprising of the IVM layer components is the primary output of Task 4.5. Therefore, the task is dependent of the outcomes and learnings within the other WP4 tasks, namely Tasks 4.1, 4.2, 4.3 and 4.4. On-going close cooperation and coordination between the dependent tasks is required to ensure that the outputs are appropriate and meet the expectations of the dependent task. Table 1.1 provides a description of the tasks that Task 4.5 is dependent on and briefly explains the dependency.

Table 1.1: Task dependencies of T4.5

Task	Dependency
Task 4.1	Task 4.1 helps to define the technology components including both software and hardware and their most appropriate configuration required to implement a performant IVM in Task 4.5.
Task 4.2	The validation of the SDN Control plane that will be defined and developed in Task 4.2 is feeding the activities of Task 4.5 that handle the deployment and integration of the SDN Control Plane of T-NOVA infrastructure.
Task 4.3	Task 4.3 focuses on the design and implementation of an SDK for SDN within the T-NOVA platform, which is one of the IVM layer components to be integrated into Task 4.5. Test case scenarios and measurements are being produced in Task 4.3, validating the functionality and performance of the SDK under development.
Task 4.4	The IVM monitoring framework developed within Task 4.4 is another of the components to be integrated by Task 4.5. Moreover, a set of test cases will be produced in Task 4.4 and will be used to validate the deployment, functionality and performance of the monitoring framework.

Due to the dependencies of Task 4.5 to the other WP4 tasks we include a list of the relevant deliverables, in which detailed component descriptions, recommended configurations and individual per component test cases can be found.

Table 1.2: Deliverables relevant to Task 4.5

Deliverable	Relevance
D4.01 - Interim Report on Infrastructure Virtualisation and Management	D4.01 [12] presents an overview of all activities within WP4 tasks, focusing on the identification of appropriate virtualisation mechanisms and enablers; implementation and characterisation of a virtualised software defined networking (SDN) control plane; implementation of an SDN software development kit; infrastructure monitoring and maintenance subsystems for the IVM.
D4.1 - Resource Virtualisation	D4.1 [13] focuses on the identification, characterisation and optimisation of the hardware and software components that can be used in the implementation of the T- NOVA Infrastructure Virtualisation and Management (IVM) layer. This document includes testbed configuration and network optimisation, Storage Performance Characterisation test cases, VNF Workload Characterisation test cases and test cases for the vTC VNF.
D4.21 - SDN Control Plane - Interim	D4.21 [14] focuses on the design and development of the T-NOVA SDN control platform. Section 5 of this deliverable provides initial preliminary validation test cases for the SDN control plane. The first test case features a single controller instance and aims to validate the functionality of recovery from persistent data when using the clustering service. The second test case features multiple instances of controllers and its purpose is to validate the functionality of high availability of the control plane after an instance (specifically the leader) of the cluster fails.
D4.31 - SDK for SDN - Interim	D4.31 [15] focuses on the specification of the SDK for SDN architecture and its implementation. Section 5.4 of this deliverable provides initial test case scenarios, validating the functionality and benchmarking the developed SDK. The first scenario is related to service function chaining. It uses one of the T-NOVA VNFs to carry out traffic classification and branch the flow into two different chains. The second scenario provides basic performance characterisation of non-

	tunnelling traffic between two OpenStack VMs.
D4.41 - Monitoring and Maintenance – Interim	D4.41 [7] focuses on the design, implementation and integration of the monitoring framework elaborated within Task 4.4. Section 5 is dedicated to the validation and assessment of the developed framework. A set of functional tests are provided validating the functional capabilities of the monitoring framework and also, benchmarking tests are performed to test the scalability and performance of the monitoring framework when overloaded with an increasing number of requests for monitoring metrics.
D5.31 - Network Functions Implementation and Testing - Interim	D5.31 [16] focuses on the architecture and implementation details of the VNFs developed within T-NOVA. Preliminary testing scenarios have been considered for validating the expected behaviour, dimensioning and performance of the developed VNFs. Within Task 4.5 we are not interested in the test cases that are specific a particular VNF function, but to use a set of the developed T-NOVA VNFs and validate the correct function of the IVM layer stack that is related to VNFs such as typical operations on VNFs and their images (deploying, storing) and the ability to provide connectivity between the deployed VNFs.

2. REFERENCE T-NOVA NFVI OVERVIEW

2.1. Technology Choices and Selection Rationale

2.1.1. Hypervisors

There is a variety of open source and commercial hypervisors available such as KVM, QEMU, XenServer, VMware vCenter, Hyper-V, LXC and Ironic. OpenStack supports many hypervisors, each one offering a different set of features and capabilities. An effort has been made to collect all supported hypervisors and compare their features in a single table [8] so that users can select the most appropriate hypervisor for their needs.

In the deliverable D2.31 - Specification of the Infrastructure Virtualisation, Management and Orchestration - Interim [17] the key functionality of a hypervisor was described together with two approaches for classifying hypervisors, one based on the virtualisation method used by the hypervisor and a second one based on the type of the hypervisor. We also presented a short overview of the key hypervisor technologies, namely KVM, XenServer, VMware and Hyper-V, being the most popular choices amongst the available hypervisors. Section 7 of D4.01- Interim Report on Infrastructure Virtualisation and Management [12], the describes rational for the selection of the KVM hypervisor for used in T-NOVA.

2.1.2. Computing

The compute platform has a significant impact on the performance of the VNF. Typically, VNF workload characterisation is used to determine what type of compute platform is required to support the function in advance. For example, a VNF such as the vTC developed in the project run with 1 CPU and 2GRAM. However other VNF's many require much larger allocations of resources such as virtual content distribution network (vCDN). Also the number of a VNF instance or services to be run on the compute node has an influence on the sizing of the node e.g. a small Customer Premise Equipment (CPE) vs the main deployment for a large multinational company. Typically compute resources can be classified as small e.g. Supermicro Intel Atom based SOC's, medium e.g. HP DL360/DL390 and large e.g. Dell PowerEdge R930. For the purposes of the IVM testbed a medium scale node was the most appropriate choice given that nodes are required to support a limited number of VNFs and services. Another consideration for compute selection is availability. For a Telco grade NFVI deployment high-availability is a high priority requirement and therefore redundant controller and compute nodes are necessarily. However, for the IVM testbed high-availability was not required given the testing nature of the deployment.

2.1.2.1. Central Processing Units (CPU)

The selection of the CPU is key consideration for the compute node. ARM based options are available for small server options however the X86 s CPU offerings are available across the small to large categories. The current generation of CPU's targeting the medium to large category from Intel are the Haswell generation of processors (Xeon E5-26xx v3 series). Many server options are also available with the Ivy Bridge processor generation (Xeon E5-26xx v2). The testbed used in Task 4.1 used both generations. When considering the selection of a server there a number of factors which can affect overall system performance; however the most influential are as follows:

- CPU speed (1.6 GHz to 3.7 GHz, note some processors have offer a Turbo mode which is the maximum single core frequency at which the processor is capable of operating using Intel® Turbo Boost Technology.)
- CPU Features e.g. Integrated I/O, to reduce latency, Data Direct I/O Technology, to improve I/O performance through direct storage to cache communications. Advanced Programmable Interrupt Controller virtualization (APICv) to reduce virtualization overhead to improve performance and scalability.
- L3 Cache (measured in megabits, usually 10MB to 45MB)
- Number of cores per processors (4 to 18)
- Number of processors (1 to 8)
- QPI (provides high-speed, point-to-point links inside and outside of the processor. Speeds up data transfers by connecting distributed shared memory, the internal cores, the I/O hub, and other processors. With speeds of up to 9.6 GT/s supported.

Current Intel CPUs also have an integrated memory controller and an integrated PCIe controller which can significantly improve I/O handling an important capability for VNF applications.

2.1.2.2. Memory

VNFs typically require specific memory allocations; therefore, the amount and type of memory can have an important influence on performance. The number of dual in-line memory module (DIMM) slots and total memory affect how many VNFs can be supported on a given server. The speed of the DIMMs also has important influence on performance. While most servers support mixed speeds, it is common for them to operate at the slowest DIMM's common frequency. It is important to note that the various DIMM types such as unregistered DIMM (UDIMM), registered DIMM (RDIMM) or load reduction DIMM (LRDIMM) cannot be mixed.

2.1.2.3. CPU and NUMA Pinning

Provisioning VNFs requires the allocation of resources (memory and CPU). As described in deliverable 4.1 it is best practical to assign cores from the same CPU (i.e., the same socket) to a VNF for the best I/O performance and reduced latency. The process of assigning CPU cores is called "CPU pinning".

In multiprocessor systems NUMA pinning can have a significant influence on performance (up to 50% in packet processing throughput – see D4.1). NUMA is a computer memory access design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. In multiprocessor systems the processors can be grouped together with their own memory and their own I/O channels which includes direct access to the NIC. When a processor accesses memory that is not connected (i.e., remote memory), or I/O device e.g. NIC the data must be transferred over the NUMA connection (QPI) and this significantly impacts performance.

2.1.2.4. QuickPath Interconnect (QPI)

In a multi-processer server, the mechanism used to interconnect the processor and their access to memory slots and other I/O components is an important factor in system performance. Recent generations of servers based on Intel's multi-processor architecture use a high speed point-to-point technology called QuickPath Interconnect (QPI). A system with two processors will have two QPI connections providing speeds up to 9.6GT/s, which translates to 19.6 Gbps for unidirectional traffic. Traffic I/O from a PCIe to another PCIe slot on the same processor does not traverse the QPI, while traffic to a PCIe slot on the second processor will. It is therefore important to understand the architecture of the system and what the PCIe slot assignment is.

2.1.2.5. Huge Pages

When executing instructions in an x86 architecture both the CPU and OS mark the RAM as being used by a process. For efficiency, the CPU usually allocates RAM in blocks (the default value for Linux is 4KB) named pages. Since these pages can be swapped to the disk, the memory addresses are virtual and the operating system has to keep track of which page belongs to which process and where they are stored on disk. As the number of pages increases, more time is taken to find where the memory has been mapped too. Newer CPU architectures and operating systems support bigger pages (so less time spent on look-ups as is the number of pages required). This feature is called Huge Pages. Huge page support is processor dependent, for example only Xeon class processor support 1GB pages. As described in D4.1 huge pages can have had effect on VNF performance particular when multiple VNFs are deployed on the same compute node.

2.1.2.6. Hyper-threading Support

Hyper-threading on an x86 CPU is used to improve parallelisation of computations (doing multiple tasks at once). For each physical processor core available in a CPU the operating system addresses two virtual or logical cores, and shares the workload between them when possible thus doubling the amount of physical cores of a multi-core CPU. For example, if a server has two (dual socket) Intel E5-2690 v3 processors, each with 12 physical cores, enabling HT provides 48 logical cores (2x 12 cores x2). While hyper-threading is a commonly used feature due to potential improvements in the efficiency of CPU utilisation, it can also degrade performance due to resource contention in some scenarios. Ideally a VNF application should be written specifically to make use of hyper-threading. Hyper-threading needs to be enabled in the BIOS of the server.

2.1.2.7. Virtualisation Support

To support virtualisation on an X86 server, Intel Virtualisation Technology (Intel VT) must be enabled on the server platform. Intel VT is required to run virtualisation technologies such the KVM hypervisor. The feature is configured in the BIOS and is typically enabled by default.

2.1.2.8. Network Interface Card

Most servers have one or more network interface cards (NICS). The selection of the NICs is a significant decision and will affect what services can be deployed on the server.

When planning the number of NICs in a server, it is important to consider that:

- One interface is required for management/orchestration
- Typically, a second interface for management/orchestration for carrier grade applications.
- Two or more interfaces are required for network services

The NICs used for network services must support the required line rate e.g. 10Gbps, 40 Gbps etc. while the NIC for management and orchestration can have lower speeds such as 1Gbps or less. SPF+ or 10GBaseT are two common options for the physical interfaces of the NIC and come in come in single, dual and quad port configurations. For SPF+ NIC vendors frequently only support transceivers that have been tested and certified by them.

The NIC should support advanced features such as SR-IOV, DPDK, Virtual Machine Device Queues (VMDq), On-chip QoS and Traffic Management and others.

The version of PCIe the NIC supports is important with PCIe 3.0 being the latest version (Older versions are 2.0 and 1.1). It is important to know the version of the PCIe slots on the server and to ensure the NICs being ordered are compatible with PCIe version. If 40Gbps NICs are being considered, they will require PCIe 3.0. Also, the physical dimensions of the NIC may be important as the server may have either full height or low profile slots.

2.1.2.9. PCI Passthrough

PCI Passthrough is a capability allows a physical PCI device from a host machine to be assigned directly to a VM on the same host machine. The VM can use the device hardware such as NIC directly bypassing the hypervisor and vSwitch. This configuration can provide I/O performance improvement particularly for some workload types. When PCI Passthrough is enabled the CPUs is responsible for mapping PCI physical addresses to guest virtual addresses. The CPU manages device access (and protection), while the guest OS uses the device as if it were a non-virtualized system. Intel provides passthrough support through its Directed I/O (VT-d) technology while AMD provides in the form of I/O Memory Management Unit (IOMMU).

2.1.2.10. FPGA SoC-based Compute Node

T-NOVA also looked into the possibility of utilising heterogeneous compute nodes for accelerating specific VNFCs. As part of that investigation the integration of such a compute node is being pursued. As the vehicle of choice for this experiment an FPGA SoC-based node has been selected since a device offers ample programmable resources through which the acceleration is achieved and two ARM CPU cores on which the OpenStack agent software can be executed thus facilitating integration into existing cloud infrastructure. Since FPGAs differ fundamentally from standard x86 CPUs, several of the OpenStack components had to be adapted to accommodate this:

- A modified nova scheduler service communicates with the database to identify & manage an appropriate FPGA host.
- The Nova Conductor service was modified to interact with the Glance and to fetch VNF information such as host ID and flavour.
- The Nova Compute service was adapted to compensate for the absence of a hypervisor (libvirt, XenAPI, etc).
- The Glance service was modified to store the bitstream used to configure the FPGA.

This modified version of OpenStack is to be used when managing the FPGA SoC compute node, although all changes made have been performed as extensions of existing functionality, meaning the adapted OpenStack version can also manage standard x86 CPUs together with FPGA-based ones.

2.1.2.11. Testbed Compute Node Selection

For the implementation of the IVM layer testbed we have selected server hardware that satisfies the previously mentioned technology choices. Most models of the latest generation servers fulfil our requirements in CPU, memory, I/O Interconnect and network and virtualisation capabilities. In addition, latest hypervisors support hyper-threading and provide NUMA style division of PCIe I/O lanes between CPUs.

2.1.3. Storage

2.1.3.1. Instance Storage Options

Typically, there three approaches to provide storage on which the instantiated instances in a cloud will run:

On compute node storage - Non-shared

In this approach, each compute node is specified with enough disks to store the instances it hosts. The main advantages of this configuration is the increased performance due to the direct I/O access and also, the fact that heavy I/O usage on one compute node does not affect instances on other compute nodes. However, there are several disadvantages as well: In case of a compute node failure, the instances stored within that node are lost. Moreover, migrating instances from one node to another is more complicated and adding extra capacity is limited by the chassis size of the compute node.

On compute node storage - Shared

With this approach, each compute node is provisioned with a significant amount of disk space, but a distributed file system ties the disks from each compute node into a single mount. The main advantage of this option is that it scales to external storage when additional storage is required. On the downside, running a distributed file system removes data locality compared to non-shared storage where each host runs its instances. On a compute node failure, recovery of lost instances becomes complicated. Moreover, network access decreases performance, which may be not viable for applications with storage requirements sensitive to latency. Again, extensibility is limited by the chassis size of the compute nodes.

Off compute node storage - Shared

Following this approach, the disks storing the running instances are hosted in servers outside of the compute nodes. In this way, the instances are not dependent to the compute nodes normal operation. If a compute node fails, instances will be easily recovered. Moreover, running a dedicated storage system can be operationally simpler, supporting the temporary removal of compute nodes for scheduled maintenance activities. Moreover, the separation of compute to storage nodes is more efficient in terms of specification planning. Compute nodes have different requirements to storage nodes. Compute hosts typically require more CPU and RAM than storage hosts, while disk capacity is the most important factor for storage hosts. Separating compute nodes from storage nodes is a tactic typically followed by most operators as reliability and scalability are the most important factors in production-level cloud deployments. One of the disadvantages of this approach is the fact that network access can decrease performance. Off compute storage is not suited well for applications with low latency storage requirements.

2.1.3.2. Types of Storage

Storage might be **ephemeral** (temporary) or **persistent**. Ephemeral storage is tightly tied to the lifecycle of each virtual machine and it is the only option when the users do not have access to any form of persistent storage by default. Usually the local drives of the compute nodes that host the virtual machines are used for non-persistent storage. The disks associated with VMs are "ephemeral," meaning that (from the user's point of view) they effectively disappear when a virtual machine is terminated. Snapshots created from a running instance will remain, but any additional data added to the ephemeral storage since last snapshot will be lost. On the other hand, persistent storage means that the storage resource outlives any other resource and is always available, regardless of the state of a running instance. Today, clouds explicitly support three types of persistent storage: file system storage, block storage and object storage.

2.1.3.3. File System Storage

The most traditional service type is shared filesystem, or simply “file storage”, which as the name implies offers to multiple clients the ability to access a single shared folder. The two most popular shared filesystem protocols in use today are NFS and SMB/CIFS. Users interact with Shared File Systems service by mounting remote File Systems on their instances with the following usage of those systems for file storing and exchange. Shared File Systems service provides you with shares. A share is a remote, mountable file system. A user can mount a share to and access a share from several hosts by several users at a time.

Like Block Storage, the Shared File Systems service is persistent. It can be mounted to any number of client machines and detached from one instance and attached to another without data loss. During this process the data are safe unless the Shared File Systems service itself is changed or removed.

Shares are provided by the Shared File Systems service. In OpenStack, Shared File Systems service is implemented by Shared File System (manila) project, which supports multiple back-ends in the form of drivers. The Shared File Systems service can be configured to provision shares from one or more back-ends. Share servers are, mostly, virtual machines that export file shares via different protocols such as NFS, CIFS, GlusterFS, or HDFS.

2.1.3.4. Block Storage

Block storage provides network access to the equivalent of raw block devices. Users interact with block storage by attaching volumes to their running VM instances. A client machine connects to a specific volume on the storage service and formats it as if it were a local block device. These volumes are persistent: they can be detached from one instance and re-attached to another, and the data remains intact.

Most block storage drivers allow the instance to have direct access to the underlying storage hardware's block device. This helps increase the overall read/write IO. However, support for utilising files as volumes is also well established, with full support for NFS, GlusterFS and others. Multiple clients do not generally mount the same volume, but they may in master/slave high-availability configurations where the slave needs to be ready to take over the master. Block devices are usually exported over Fibre Channel, iSCSI or AoE (ATA over Ethernet).

Block storage is implemented in OpenStack by Cinder, the OpenStack Block Storage project, which supports multiple back ends in the form of drivers. The choice of a storage back end must be supported by a Block Storage driver. In addition, Ceph OpenStack project also offers block storage technology.

2.1.3.5. Object Storage

Object storage is a relatively new storage type, designed for unstructured data such as media, documents, logs, backups, application binaries and VM images. Conceptually they are like a persistent key/value store; objects are usually submitted via a REST API call, and an identifier returned. Most object stores allow attaching metadata to objects, and aggregating them into containers (or buckets). Both Ceph and Swift OpenStack projects offer object store interfaces, with Swift to be the recommended option for using only this type of storage. The most popular cloud object store is AWS S3 [18], and many object store implementations are compatible with it, including Ceph's S3 RADOS Gateway service [19].

2.1.3.6. NFVI Storage solution choices

As laid out in Deliverable 2.1 [21] with respect to the requirements of the storage subsystem and also taking into account the requirements imposed by the VNF developers [16], the selected implementation of storage for T-NOVA is summarised in the following table (Table 3):

Table 3: T-NOVA storage solution

Purpose	Technical Solution	Comments
Block storage for hosting of the VM instances	Separate NAS based disk array is used that serves as a common repository for hosting of the VM instances that run the VNFs. The access to the storage subsystem is achieved by NFSv4 protocol	This solution allows for live-migration between compute nodes, with close to zero latencies
Block storage for hosting of VNF/VM Images	Separate NAS based disk array may be used for this purpose or use Cloud Controller local disk. As this same path usually stores the snapshot, it would be more efficient for quicker access, backup and storage of snapshots. The access to the storage subsystem is achieved by NFSv4 protocol	Due to the limited span of the testbed environment discussed in this deliverable, it was selected to host the images and the snapshots on the Cloud Controller.
Block/Object Storage for hosting of persistent or ephemeral storage for disk volumes used by the VNFs	The additionally requested storage from the VMs is served either locally at each compute node or over the network at the disk array (over NFSv4 or iSCSI).	To avoid complicated storage models and focus on providing the minimum requirements for the VNF storage needs

The main driver of our choices related to storage was to retain a configuration as simple as possible. For this reason, we did not focus on an optimization based on the workloads or the types of VNFs.

2.1.4. Network

2.1.4.1. OpenStack Neutron Network Topologies

Neutron is a service which provides Networking-as-a-Service functionality in OpenStack. It has a rich tenant-facing API for defining network connectivity and addressing in the cloud, and gives operators the ability to leverage different networking technologies to power their cloud networking.

Each tenant has a virtual Neutron router with one or more private networks, which can communicate with the outside world. This allows full routing isolation for each tenant private network. Virtual networks (one or more) can be created for a single tenant, forming an isolated L2 network called a "private network". Each private network can support one or more IP subnets. Private networks can be segmented using one of three different topologies:

VLAN (IEEE 802.1Q tagging) segmentation

OVS will in the virtual switches allocate an internal VLAN for each tenant. These VLANs provide separation amongst the tenants (as VLANs are designed to do). Tenants can specify the same subnet and overlap in that subnet range (VM1 from tenant 1 can get assigned IP 10.4.128.3 and VM1 from tenant 2 can also get 10.4.128.3, without conflict).

Ideally, "Private network" traffic is located on a dedicated network adapter that is attached to an untagged network port. It is, however, possible for this network to share a network adapter with other networks. In this case, non-intersecting VLAN-ID ranges for "Private network" and other networks should be used.

A typical deployment for VLAN segmentation is shown in Figure 2-1. The network tagged as Management Network is a tagged or untagged isolated L2 network reserved for the communication between the different OpenStack components and supporting services (RabbitMQ, MySQl, etc.). The External Network is a tagged or untagged isolated L2 network that serves for external API access and providing the tenants' instances with connectivity to/from networking outside the cloud. The Storage Network connects the compute nodes to the server running the OpenStack storage services (Glance, Ceph, Swift or Cinder depending on the type of storage selected). Finally, the Private Network serves for 802.1Q (VLAN) tagged traffic of private network segments for tenants.

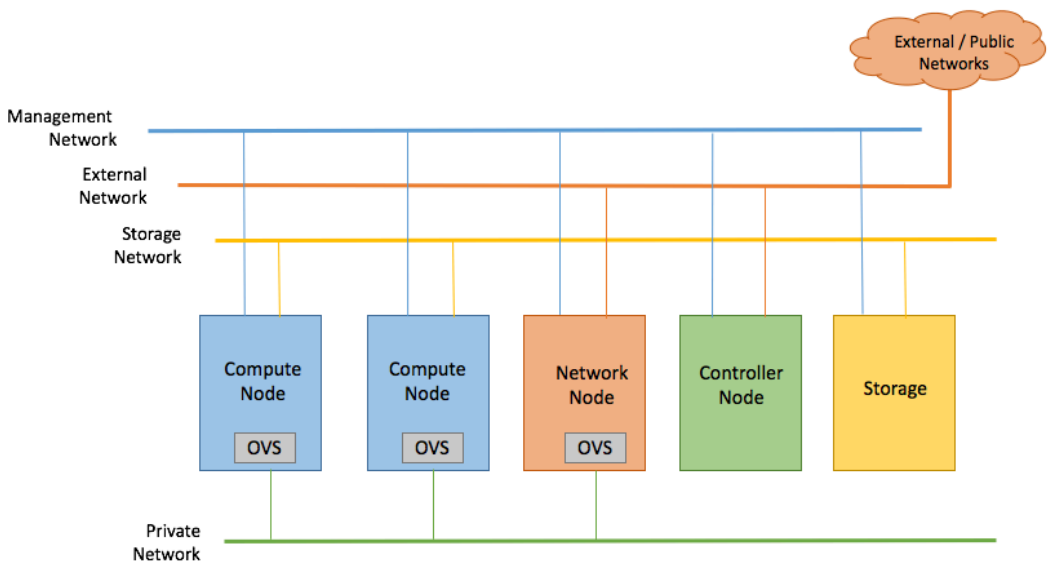


Figure 2-1: Neutron with VLAN segmentation

GRE segmentation

In this mode of operation, Neutron does not require a dedicated network adapter. Neutron builds a mesh of GRE tunnels from each compute node and controller nodes to every other node. Private networks for each tenant make use of this mesh for isolated traffic, encapsulating tenant traffic in the created tunnels. For example, a tenant might have VMs running on compute nodes A, B, and C. Neutron, along with OVS, will build a fully connected mesh of tunnels between all of these machines, and create a tunnel bridge on each of these nodes that is used to direct traffic from VMs into and out of these tunnels. If a VM on machine A wants to send packets to a VM on machine B, machine A will encapsulate the IP packets coming out of the VM using a segmentation ID that is generated for the tenant by OpenStack, and the receiving machine (B) will decapsulate the packets and route them to the destination VM using the addressing information in the Ethernet frame. Figure 2-2 depicts a typical deployment using GRE segmentation technology.

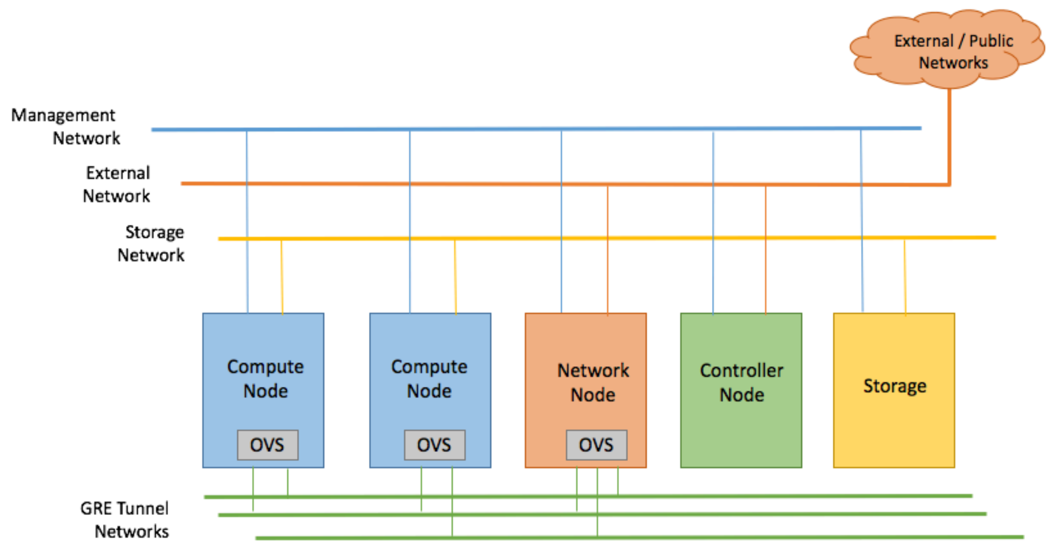


Figure 2-2: Neutron with GRE segmentation

VXLAN segmentation

VXLAN segmentation is similar to GRE segmentation. It also provides separation among tenants, and also allows overlapping subnets and IP ranges.

All of the Neutron topologies can use Open vSwitch (OVS) to isolate tenants from each other on L2 and L3 layers or the ML2 plugin with OVS driver.

The currently elaborated solution is a hybrid configuration that will effectively combine the native (default) network deployment model of Openstack environment along with the capability to have multi-homed compute nodes that are connected via Provider Network model. The later required manual configuration and interconnection with actual production equipment (e.g. Gateways, FW switches) in order to achieve connectivity.

2.1.4.2. Packet Acceleration

Intel's Data Plane Development Kit (Intel® DPDK) is a set of libraries and drivers for fast packet processing on x86 platforms that can improve packet-processing performance by up to ten times. DPDK support, which actually depends on the processor, is integrated in all recent Intel Atom and Xeon processors.

DPDK improves packet throughput performance by leveraging Poll Mode instead of interrupt based drivers. DPDK includes 1 Gigabit, 10 Gigabit and 40 Gigabit and para virtualized virtio Poll Mode Drivers. One side effect of this is that the CPU core assigned will run at 100%. It also maps software threads to hardware queues on the dedicated CPUs. DPDK also leverages batch packet processing (handling multiple packets at a time). While this can improve throughput, it may induce some latency, depending on the batch size used. It also uses huge memory pages, which greatly reduces TLB thrashing. DPDK can be enabled by Open vSwitch and can also be enabled directly from the VM. Use of DPDK requires a compatible NIC. Companies such Cisco, Intel, Mellanox and Broadcom provide NICs with DPDK support. A listed of supported NIC can be found at [20]. Apart from the necessity of the NIC supporting DPDK the VNF must also be built with DPDK support. A number of the T-NOVA VNF's use DPDK to improve packet throughput. The DPDK version of virtualised traffic classifier VNF for example achieved in excess of 8 Gbps throughput (see deliverable 4.1 for further details. Therefore, DPDK is a key packet acceleration technology for the testbed and will be fully supported.

2.1.4.3. SR-IOV

Single Root I/O Virtualisation (SR-IOV) is an extension to the PCI Express (PCIe) specification. It enables a single PCI Express (PCIe) device such as a network adapter to appear to the hypervisor as multiple special-purpose network adapters. These special-purpose network adapters, termed Virtual Functions (VF), are only available for direct presentation to VMs. By providing a VF directly to a VM, the hypervisor's virtual switch is no longer required to process network traffic. This hypervisor bypass increases network throughput, lowers latency, and reduces overall CPU utilisation.

Network adapters that feature SR-IOV are comprised of one Physical Function (PF) and multiple VFs per port. Each PF and VF is assigned a unique PCI Express Requestor ID (RID) that allows an I/O memory management unit (IOMMU) to differentiate between different traffic streams and apply memory and interrupt translations between the PF and VFs. This allows traffic streams to be delivered directly to the appropriate VM. As a result, data traffic flows from the PF to VF without affecting other VFs. It is important to note when a PCI device is directly assigned to a VM, migration will not be possible without first hot-unplugging the device from the guest which significantly impacts on the ability to migrate VMs (see deliverable D2.32 state of art review section for more details).

2.2. Infrastructure Components

2.2.1. Roles

In this section we define a number of roles, where each defined role performs a specific function, either related to the OpenStack environment, T-NOVA components and their testing or the provision of a special purpose function, such as ensuring service availability. It is not mandatory to assign a role per server; as we will see later in the Role Planning section the number of servers required for the deployment depends on the purposes this deployment was created for. It might well be the case that a server can be assigned to two or more roles at the same time.

2.2.1.1. OpenStack Nodes Roles

An OpenStack environment contains a set of specialised nodes and roles. While single-node configurations are acceptable for small environments, testing or POCs most production environments will require a multi-node configuration for a variety of reasons. As outlined previously multi-node configuration groups similar to OpenStack services provide scalability as well as support for high availability. The standards for deploying multi-node OpenStack are as a two-node, three-node or four-node configuration, separating compute, controller, network and possibly storage services. In the following paragraphs we briefly describe these roles and their function and the services they run.

Controller Node

A Controller node is a server which has most of the shared OpenStack services and other tools needed to orchestrate the deployment of virtual machines on Compute nodes. The Controller node supplies API, scheduling, and other shared services for the cloud. Usually the Controller node runs the following OpenStack services:

- the dashboard (Horizon),
- the image store (Glance),
- the telemetry service (Celiometer),
- the orchestration service (Heat)
- the identity service (Keystone).

Other services that may optionally run on the Controller include:

- Neutron,
- Swift,
- Ceph Monitor,
- Sahara,
- Murano.

To achieve high availability for the environment, a cluster of at least three Controller nodes is recommended. It is possible, although not recommended, to run both the Compute and Controller roles on a single server.

Compute Node

The Compute nodes are the servers on which the users will create their virtual machines and host their applications. Each compute node runs a hypervisor (KVM, ESX, Hyper-V, XenServer, etc.) program that allows the virtual machines to share the compute node's hardware. In a common setup, the compute node handles compute service (nova), telemetry (ceilometer compute agent) and network Open vSwitch agent service (neutron). Compute nodes need to talk to controller nodes and reach out to essential services such as RabbitMQ and MySQL.

Network Node

A Network node is a server that runs networking services. It runs the neutron services for L3, metadata, DHCP and Open vSwitch. The network node handles all networking between other nodes as well as tenant networking and routing. It provides virtual networking and networking services to Nova instances using the Neutron Layer 3 and services such as DHCP and floating IPs that allow instances to connect to public networks. Neutron sits on top of Open vSwitch using either the ml2 or openvswitch plugin. Using Open vSwitch, Neutron builds three network bridges: br-int, br-tun and br-ex. The br-int bridge connects all instances. The br-tun bridge connects instances to the physical NIC of the hypervisor. The br-ex bridge connects instances to external (public) networks using floating IPs. Both the br-tun and br-int bridges are visible on compute and network nodes. The br-ex bridge is only visible on network nodes.

Storage Node

The storage node runs storage services. These include the image service (Glance) and depending on the storage options we have selected for our environment block storage (Cinder or Ceph), object storage (Swift or Ceph) and shared file storage (Manila). Typically, a storage node would run one type of storage service: object, block or file system. Glance should run on nodes providing storage services for images (Cinder or Swift) or on a separate node.

2.2.1.2. Other Roles

Apart from the roles that are usually present in a standard OpenStack deployment, for the T-NOVA IVM layer deployment we expect nodes to be assigned to other roles such as a T-NOVA node, an SDN controller, a Build Server, a Testing Server and an Endpoint Node.

T-NOVA Node

A T-NOVA node is a server on which we have deployed T-NOVA IVM layer components. The components that need to be deployed are the SDN Control Platform, the SDK for SDN and the Monitoring Framework. Even in the simplest case of a deployment for study purposes multiple nodes are generally required.

SDN Controller Node

The SDN Controller node is a server running an SDN controller application to manage flow control. The controller is the logical control centre of the SDN network, communicating with switches via its "southbound" interface to provide networking instructions and communicating with applications via its "northbound" interface. Typically, the SDN controller contains a collection of pluggable modules that can perform different tasks such as inventorying what devices are within the network and the capabilities of each, gathering network statistics, etc. After reviewing a number of available SDN controllers, in T-NOVA we have decided to use the OpenDaylight SDN controller.

Build Server

A build server is a server assigned with the task to provision other nodes with their selected roles. It runs software that enables automated provisioning and configuration management and it usually runs and manages network services including DHCP, DNS, PXE, and TFTP. A build server is mainly responsible for the installation and configuration of software packages, as well as the configuration of new nodes (e.g. configuration of users and groups or network interfaces). Depending on the software it runs, it may be able to support bare-metal provisioning of new nodes through PXE-based on an unattended installation of the operating system. It may also monitor servers over the duration of their lifespan, reporting possible issues or failures. Well-known tools for automated provisioning and configuration management include Foreman, Fuel, Ansible, Puppet, Chef and Salt among others. Obviously, the existence of a build server for a T-NOVA IVM layer deployment is not mandatory. However, assigning this role to a server facilitates the deployment due to the use of the automated provisioning and configuration management tools.

Testing Server

A set of libraries and software required to set up a testing environment can be installed on this server to run tests validating the functionality and the performance of the services deployed in the other nodes of our configuration. Obviously, as with the build server, the existence of a testing server is not mandatory. By setting up the testing environment to a remote machine separate from the services under test, we diminish the possibility of mixing the different and often conflicting requirements (packages and libraries) between them.

Endpoint Node

The Endpoint node runs load balancing and high availability services. The Endpoint node is optional in a T-NOVA INM layer deployment and relates to production-level T-NOVA deployments.

2.2.2. Roles Planning

The following sections provide details related to deployment planning for different scenarios and purposes. In particular, three different deployments are described, one for demonstration and study purposes, one for testing purposes and one for production purposes.

The setups described do not make any allocation to T-NOVA nodes as the implementation of IVM layer T-NOVA components is currently in progress. Safe assumptions on the roles planning and hardware requirements for the T-NOVA components will be made in the final version of this deliverable.

2.2.2.1. Setup for Development Purposes

The absolute minimum requirement for a T-NOVA IVM layer deployment in a lab environment is the allocate of two nodes:

- 1 All-In-One OpenStack node that combines multiple roles, including the Controller, Compute, Network and Storage roles in one server
- 1 SDN Controller node

Such a deployment can serve development purposes, minimising the hardware requirements. The testing environment, along with tests can be installed and deployed in an All-In-One OpenStack node. Another possible option is the deployment in virtual machines for example using software such as VirtualBox.

2.2.2.2. Setup for Extensive Testing and Experimentation Purposes

When setting up a T-NOVA deployment for testing and experimentation purposes, it is recommended to spread the OpenStack roles (and, hence, the workload) over as many servers as possible in order to have a fully redundant, highly-available OpenStack environment and to avoid performance bottlenecks. It is highly preferable to separate compute from storage and network services, resulting in an allocation of 6 nodes:

- 1 Controller node
- 2 Compute nodes
- 1 Network node

- 1 Storage node
- 1 SDN Controller node

2.2.2.3. Setup for Pilot/Demonstration

The above minimum and recommended deployments do not foresee replication of services and servers to minimise downtime and data loss in the event of a hardware failure. The setups proposed are focused on testing purposes and not on high availability scenarios.

A setup for pilots or demonstration of T-NOVA requires high availability and redundancy of services, scalability and automation of operations. The OpenStack services are maintained on separate servers and also include a separate node for automated provisioning and testing. In addition, endpoint nodes offering load balancing to the OpenStack services are expected.

- 2 Controller nodes
- 2 Compute nodes
- 2 Endpoint nodes
- 1 Network node
- 2 Storage node
- 1 SDN Controller node
- 1 Build server (automated provisioning of nodes and testing environment)

2.2.3. Hardware Requirements

When choosing the hardware to deploy the T-NOVA IVM layer stack on, considerations have to be taken on the hardware requirements and specifically on the CPU, memory, storage and networking requirements for the servers:

CPU considerations - The number of required CPUs depends on the number of virtual machines we plan to deploy in the cloud environment and the CPU allocation per virtual machine.

Memory considerations - Depends on the amount of RAM assigned per virtual machine and the controller node.

Storage considerations - Depends on the local drive space per virtual machine, remote volumes that can be attached to a virtual machine, and object storage.

Networking considerations - Depends on the network topology chosen, the network bandwidth per virtual machine, and network storage.

2.3. T-NOVA Components

2.3.1. Virtualised SDN Control Plane

The Virtualised SDN Control Plane is composed by one or more synchronized instances of OpenDaylight controllers, each one deployed on a dedicated virtual machine. Such instances are configured to form a cluster of controllers, leveraging on the clustering service built in ODL based on the Akka [6] framework.

The clustered approach was adopted in order to develop a logically-centralized but physically-distributed SDN control plane aiming at providing data-persistence, reliability and high availability of the control plane. The state synchronization is obtained through a distributed datastore to ensure that each instance of ODL controllers works on the same data/state information. More specifically, the clustering service is provided by the OpenDaylight Model-driven Service Abstraction Layer (MD-SAL) Clustering feature and can be used in two different scenarios. In the single node cluster scenario, such feature takes advantage from the distributed data-store to provide data-persistence capabilities, i.e. the network state, being stored in the persistent memory, can be reconstituted after restarting the controller. Indeed, when multiple instances are deployed, such feature provides also horizontal scaling, fault-tolerant and high availability of the control plane. So, once a node becomes unavailable, it can be easily replaced by another one running in the cluster. Preliminary validation tests focused on the clustering service have been carried out by Task 4.2. Results are reported in D4.21.

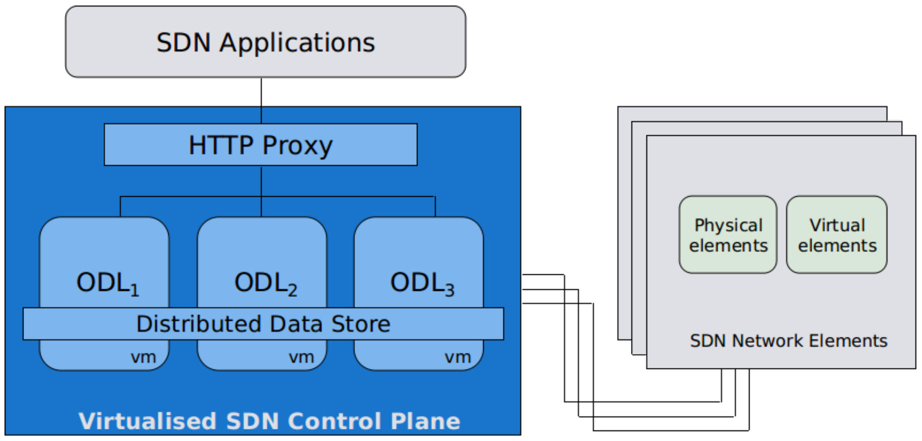


Figure 2-3 Virtualised SDN Control Plane

Leveraging the clustering service, the virtualised SDN Control Plane provides a common interface to the northbound applications, as shown in Figure 2-3. Since each RESTful request may be served by any controller in the cluster, a HTTP proxy is used to distribute the requests to one of the available instances. From the southbound side, the network nodes can be connected with one or more controllers simultaneously, allowing load balancing of the instances belonging to the cluster.

2.3.2. Monitoring Framework

The T-NOVA VIM Monitoring Framework is the subsystem of the VIM responsible for collecting, aggregating and communicating dynamic monitoring metrics to the upper layers (Orchestrator and, in turn, Marketplace). It comprises two main elements:

- A centralised VIM Monitoring Manager (VIMMM), which consists of:
    - OpenStack and OpenDaylight interface modules, to retrieve monitoring metrics from the network and cloud controllers
    - A VNF Application connector, which accepts data periodically dispatched by a VNF application. These metrics are specific to each VNF.

    - A time-series database (InfluxDB) for data persistence.
    - An alarming/anomaly detection engine for event processing and fault detection
    - A Graphical User Interface (GUI), based on Grafana, which visualises the stored metrics and presents them as live, time-series graphs.
    - A Northbound REST API, which communicates selected metrics and events to the Orchestrator and, in turn, to the VNF Manager(s).
- Distributed VNF monitoring agents, which are installed in the VNFCs to be monitored and collect a rich set of metrics from the guest OSs.

A detailed presentation of the VIM Monitoring Framework can be found in Deliverable D4.41 [7].

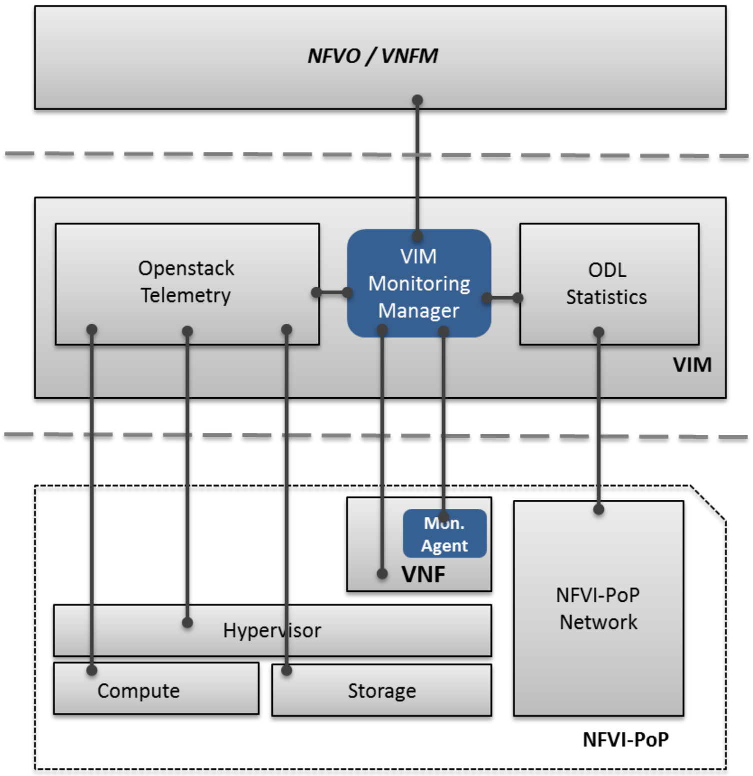


Figure 2-4. The components of the VIM Monitoring Framework

In order to integrate the VIM Monitoring Framework into the IVM infrastructure, the following steps are planned:

1. Installation of the VIM Monitoring Manager backend node.js application on either a dedicated physical server or a VM, accessible by the OpenStack public network.
    - Alternatively: The VIM MM is already built into a Docker container, so the container can be launched directly.
2. Installation of the database (InfluxDB) in either a physical server or VM (same with the VIM MM or another)

    - Alternatively: InfluxDB is already available as a Docker container, so it can be directly launched.

3. Configuration of the VIM MM (via configuration file) with regard to the following parameters:
    - OpenStack Keystone IP/port and credentials (in order to access Ceilometer and Nova)
    - Ceilometer service IP, port and polling interval
    - Nova service IP and port
    - InfluxDB IP address/port and credentials
4. Launching of the VIM MM backend service
5. Deployment of VNF agents with the following configuration (this step is meant to be automated via Heat)
    - VIM MM IP address and port to send metrics
    - Metrics to be dispatched
    - Pushing interval
6. Verification of proper operation via a GUI
7. Verification of proper operation via Northbound API (using the VIM MM swagger-based front-end)

2.3.3. SDK for SDN (SDK4SDN) and Service Function Chaining (SFC)

2.3.3.1. SDK4SDN Integration

A detailed description of the SDK4SDN can be found in [15]. The SDK4SDN integration has mainly two functions, one is to provide an API for Service Function Chaining and the other is to provide an API for connection based flow programming in a datacentre. The SDK4SDN is heavily integrated with the OpenStack Kilo release and depends on the Neutron ML2 plugin to get the necessarily information to provide SFC and connection based routing in that environment.

2.3.3.2. Service Function Chaining API

SFC in the SDK4SDN is realised by create, read, update and delete (CRUD) based remote procedure calls (RPCs). The complete API specification is under development at the time of writing. These are the minimal calls, which are necessarily to provide SFC routing:

- Create Service Chain: Provide an ordered list of Neutron Ports. The SDK4SDN will automatically calculate connection paths between them and push OpenFlow messages to establish the chain routing.
- Delete Service Chain: Chains can be referenced via an ID and deleted. Again the SDK4SDN handles this call by pushing the respective OpenFlow messages onto the network.

2.3.3.3. Installation Prerequisites

The current SDK4SDN is tested with the following dependencies:

- OpenStack Kilo release
- OpenDaylight Lithium release

In order to use SDK4SDN the network needs to be fully SDN enabled through OVS switches. SDK4SDN can only detect OVS topology. A physical switch which supports OVS is for example the Pica8 switch in OVS mode.

2.3.3.4. Disabling the firewall for Open Stack nova service and neutron service

The default SFC implementation requires that iptables rules are disabled as they would prevent non-standard forwarding to OpenStack instances.

On each controller and compute node, change in /etc/neutron/plugins/ml2/ml2_conf.ini:

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

To stop the nova-compute service from creating the iptables rules, it should be configured to use its Noop driver. In /etc/nova/nova.conf:

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

Restart all neutron-server, neutron-openvswitch-agent, nova-api and nova-compute services.

2.3.3.5. Installation

The repository is maintained on github.

```
git clone https://github.com/icclab/netfloc.git
cd netfloc
```

For compilation the following is needed:

```
export MAVEN_OPTS="-Xmx1028m -XX:MaxPermSize=256m"
```

Then the repository can be compiled as follows:

```
mvn clean install
```

2.3.3.6. Execution

After successful compilation the controller can be run using the following commands:

```
cd karaf/target/assembly/
./bin/karaf
```

2.3.3.7. Resolving Errors and Troubleshooting

The following are useful troubleshooting tips for developing on or extending the SDK4SDN.

If errors are experienced duirng the build processing due to missing bundles or features, the following should be run:

```
rm -rf ~/.m2/repository/org/opendaylight/
mvn clean install
```

If the problem happens when changes are introduced or new bundles features are installed in Netfloc, the following can be tried:

```
rm -rf journals snapshots
bin/karaf clean
```

It will clean the distribution data store from the previous executions.

Once a stable version of the installation with the required features and bundles has been created which should not be updated by Maven with new artifacts offline mode should be engaged. For that you will need to have all the artifacts available in your Maven local repo and use the dependency plugin's "go-offline".

```
mvn dependency:go-offline
mvn clean install -o
```

2.3.4. VNFs

Within WP5 of T-NOVA a set of VNFs are being developed. Specifically, a Virtual Security Appliance (vSA), a Virtual Session Border Controller (vSBC), a Virtual Transcoding Unit (vTU), a Traffic Classifier (vTC), a Virtual Home Gateway (vHG) and a Proxy as a Service (vPxaaS). An extensive description of the architecture and distinctive components of each VNF are provided in the deliverable D5.31 - Network Functions Implementation and Testing - Interim [16].

3. Infrastructure Deployment and Integration

3.1. Components Deployment

To deploy successfully an IVM layer stack the following steps are planned:

1. Determine the purpose of the deployment. As outlined in section 2.2.2 of this deliverable there are different infrastructure requirements depending on the objective of the deployment, whether it will be used for study/demonstration, testing or in a production environment.
2. Decide what is the appropriate infrastructure deployment topology: Document roles assigned to each node and note nodes' addresses that will be required during provisioning.
3. Networking configuration: Cable nodes and configure any required networking equipment.
4. Optionally, deploy a build server and configure the installation to the required nodes. In the final release of this deliverable we plan to use an available open source tool that automates the processes of deployment and provisioning. In this case, the allocation of an extra host should be considered under the role of build server.
5. Install OS to the target nodes. If a build server has been set up the OS to the target machines will be installed through bare metal provisioning of hosts. Also, required is control and management software, such as Puppet, which will be installed on the target nodes. In the scenario where a build server has not been provisioned, the OS should be manually installed on each target node.
6. Install the IVM Layer stack on nodes based on their specific roles. Installation will be done manually or through recipes configured on the build server in the case where a build server is planned.

Table 3.1: IVM layer stack templates

Component	Stack A	Stack B
OS	Ubuntu Trusty	Ubuntu Trusty
VIM	OpenStack Liberty	OpenStack Liberty
Network Controller	OpenDaylight Lithium	OpenDaylight Lithium
Virtualisation	KVM / QEMU	KVM / QEMU / Docker
Virtual Networking	Open vSwitch	Open vSwitch
VNFs	vTC, vHG	vTC, vPxaaS, vSA, vHG, vSBC, vTU

Table 3.1 provides two different templates for an IVM layer stack to be targeted. For this interim version of this deliverable the focus was on deploying Stack A with manual installation of the components. It specifically includes Ubuntu Trusty 14.04 as OS to all provisioned nodes, OpenStack Liberty as VIM, OpenDaylight Lithium for network controller, KVM/QEMU hypervisors, Open vSwitch for virtual switch software and vTC and vHG VNFs to be used for testing. Annexes A and B provide detailed instructions on the installation of OpenStack and OpenDaylight and describe the steps required for integration via the ML2 plugin. Stack B will be the target IVM layer stack for the final version of this deliverable. Within the second stack, Docker containers are included in the virtualisation enablers and all T-NOVA developed VNFs (vTC, vPxaaS, vSA, vHG, vSBC, vTU) are loaded as their development will be completed by the time of final release of this deliverable.

7. Pre-load VNF/VM images for the VNFs we want to deploy and test.
8. Install and configure the test environment to validate the deployment. The testing machine might be a compute node already assigned with a role or a different machine, separate from the actual deployment. Detailed instructions on the installation and use of the testing suites can be found in Annex C.

3.2. Components Integration

3.2.1. OpenDaylight - OpenStack Integration

OpenDaylight is integrated with OpenStack via the Neutron ML2 plugin. With the integration of OpenStack Neutron and OpenDaylight, changes to the network and network elements can also be triggered by an OpenStack user. The changes performed in OpenStack are translated into Neutron APIs, and handled by neutron plugins and corresponding agents running in OpenDaylight. For example, OpenDaylight interacts with Neutron by using the ML2 plugin present on the network node of Neutron via the REST API using northbound communication. When an OpenStack user performs any networking related operation (create/update/delete/read on network, subnet and port resources) the typical flow would be as follows:

- User operations on the OpenStack dashboard (Horizon) are translated into the corresponding networking API and sent to the Neutron server.
- The Neutron server receives the request and passes it to the configured plugin (assume ML2 is configured with an ODL mechanism driver and a VXLAN type driver).
- The Neutron server/plugin will make the appropriate change to the DB.
- The plugin will invoke the corresponding REST API to the SDN controller (assume an ODL).
- ODL, upon receiving the request, may perform necessary changes to the network elements using any of the southbound plugins/protocols, such as OpenFlow, OVSDB or OF-Config.

Annex B [6.2] provides detailed information on the steps needed for configuring OpenStack with Neutron ML2 networking to work with OpenDaylight.

4. INFRASTRUCTURE VALIDATION

The infrastructure validation of operation includes:

- Testing the basic VIM functionality that includes tenant/user CRUD operations (create, delete, list) and VNF images CRUD operations (create, delete, boot, list).
- Testing the VIM functionality to support VNF operations (deploy/create, modify, stop, destroy).
- Testing the VIM functionality to support basic VNF network connectivity (adding and deleting networks, subnets, routers and floating IPs).
- Testing the inter working between the VIM and the SDN controller (adding and deleting networks, subnets, ports and OpenFlow rules).
- Testing the NFVI functionality as a black box to ensure that it meets the VIM requirements.
- Testing the VNFs developed within T-NOVA.

Before defining the tests that will help us validate a correctly configured IVN layer stack, we describe the tools that were selected for this purpose, Rally and Tempest testing suites for OpenStack and Robot testing suite for OpenDaylight validation. Detailed instructions on the proper installation and configuration of these tools for testing the deployed infrastructure are provided in Annex C of this deliverable. A testing framework was developed that automates the installation of the testing environment and provides a GUI for the execution and reporting of results and statistics for the defined tests.

4.1. Testing Tools

4.1.1. OpenStack Tempest Test Suite

Tempest [1], [2] is the OpenStack official test suite. Its purpose is to run tests for OpenStack API validation, scenarios and other specific tests useful for validating an OpenStack deployment. It is also used as a gate for validating commits into the OpenStack core projects-it avoids breaking them while merging changes. Tempest is based on unittest2 framework and currently uses Nosetest runner to run test against OpenStack service endpoints by exercising API calls and validating the received response. There are several types of tests included in Tempest, such as smoke tests, positive tests, negative tests, stress tests and white box tests that allow functional, integration, load and performance testing. One can use the already provided Tempest tests or write its own customized tests. Overall, Tempest offers a single unified suite to test all OpenStack components, which is easily maintainable and expendable.

4.1.2. Rally Benchmarking Test Suite for OpenStack

Rally [3] is a community-based project that allows OpenStack developers, administrators and operators to get relevant and repeatable benchmarking data of how their cloud operates at under load at scale. It is intended to provide the OpenStack community with a benchmarking tool that is capable of performing specific, complicated and reproducible test cases on real deployment scenarios.

Rally automates and unifies multi-node OpenStack deployment, cloud verification, benchmarking & profiling using pluggable Rally benchmark scenarios. It offers different types of user-defined workloads useful for developers (synthetic tests, stress tests) and operators (real-life cloud usage patterns). It can be used as a basic tool for an OpenStack continuous integration / continuous development (CI/CD) system that would continuously improve its SLA, performance and stability. Rally comes with a dashboard that has an easy-to-read user interface, showing a graphical snapshot of a process's key performance indicators that enable the user to make instantaneous and informed decisions. Another valuable feature of Rally is that it can use Tempest in its testing. It automatically installs and configures Tempest, and automates running tests already created in Tempest. Moreover, Rally Benchmark can launch Tempest tests with a variable number of (simulated) active users, a feature which is not available when we are testing our deployment using Tempest alone.

Typical Rally use involves providing Rally with the OpenStack deployment to be benchmarked. This is done either through OpenRC files or through deployment configuration files in JSON format. After registering a deployment, a sequence of benchmarks has to be created which will be launched by Rally. The benchmarks are specified in a benchmark task configuration file, which is either in JSON or in YAML format. A single task might include one or multiple benchmarks. Parameters that can be set to simulate real-life cloud usage include the number of users, the number of tenants, concurrency, the type of workload and the duration of the test. There is an option to run the benchmark tasks already available in the Rally source or create our own tasks customized to our needs. The results of these tests and benchmarks are saved in Rally's database and we have the option to output these results in illustrative and comprehensive HTML reports based on the benchmarking data. The results include execution times, failure rates, graphics and charts and profiling data.

Typical Rally test cases are:

- Automated measurement & profiling focused on how new code changes affect OS performance;
- Using Rally profiler to detect scaling & performance issues;
- Investigating how different deployments affect the OS performance:
    - Finding the set of suitable OpenStack deployment architectures;
    - Creating deployment specifications for different loads (amount of controllers, swift nodes, etc.);
- Automating the search for hardware best suited for particular OpenStack cloud;
- Automate the production cloud specification generation:
    - Determine terminal loads for basic cloud operations: VM start & stop, Block Device create/destroy & various OpenStack API methods;

  - Check performance of basic cloud operations in case of different loads.

4.1.3. OpenDaylight Test Suite

For testing the OpenDaylight controller we will use the Robot Framework [5], inheriting the test cases already developed in the OpenDaylight project. Robot Framework is a Python-based, extensible keyword-driven test automation framework for end-to-end acceptance testing and acceptance-test-driven development (ATDD). Test cases are automated by writing steps using Robot framework keywords. It can be used for testing distributed, heterogeneous applications, where verification requires touching several technologies and interfaces.

The Robot Framework has a set of features that facilitate testing. The framework:

- Enables easy-to-use tabular syntax for creating test cases in a uniform way.
- Provides ability to create reusable higher-level keywords from the existing keywords.
- Provides easy-to-read result reports and logs in HTML format.
- Platform and application independent.
- Provides a simple library API for creating customised test libraries which can be implemented with either Python or Java.
- Provides a command line interface, XML and HTML based output files for integration into existing build infrastructure (continuous integration systems).
- Provides support for Selenium for web testing, Java GUI testing, running processes, Telnet, SSH, and so on.
- Supports creating data-driven test cases.
- Built-in support for variables, practical particularly for testing in different environments.
- Provides tagging to categorise and select test cases to be executed.
- Enables easy integration with source control: test suites are just files and directories whose version can be designated with the production code.
- Provides test-case and test-suite -level setup and tear down.
- The modular architecture supports creating tests even for applications with several diverse interfaces.

4.1.4. Installation of the Testing Environment

The testing environment for the T-NOVA IVM layer consists of the OpenStack Tempest test suite, the Rally Benchmarking test suite for testing OpenStack and Robot Framework for testing OpenDaylight. Annex C – Testing environment installation and validation provides installation instructions of the testing environment in two ways; either automated installation of the three tools through a script or specific steps on how to install each tool separately.

4.2. Validation Tests

4.2.1. OpenStack Tests

We use Rally OpenStack Bench test suite to run a set of functional tests that validate the OpenStack deployment and its functionality. The scenarios selected are based on the existing Rally samples scenarios [4] and test the OpenStack components individually.

We have chosen to test nova, neutron, cinder, glance, heat, ceilometer, keystone projects of OpenStack and validate their main functionalities as these are used during the interactions of the T-NOVA systems. The following table summarises the OpenStack tests we have chosen per project.

Table 4.1: OpenStack tests per service

OpenStack Component		
Nova	**Neutron**	**Cinder**
List hypervisors List images List servers Suspend and resume Boot Boot and list Boot and associate floating IP Boot and delete Suspend and resume Create and list networks	Create floating IPs and delete Create and delete networks Create and list networks Create and delete pools Create and delete routers Create and delete subnets Create and list floating IPs Create and list subnets Create and update networks Create and update pools Create and update routers Create and update subnets	List volumes Create volume Create and attach volume Create and delete volume Create and list volumes Create and delete snapshot Create and list snapshots
Glance	**Heat**	**Celiometer**

Create and delete image Create and list image Create image and boot instances List images	Create and delete stack Create, check and delete stack Create, update and delete stack Create and list stack Create, suspend, resume and delete stack Create and delete stack with neutron List stack and resources	List all meters List meters List all resources List resources List samples Create and query samples Create meter and get stats
Keystone	**Authenticate**	
Create user Create and delete user Create and list users Create tenant Create and list tenants	Keystone Validate cinder Validate glance Validate heat Validate neutron Validate nova	

Using Rally, we are also running Tempest. Tempest scenario tests are "through path" tests of OpenStack function, with complicated setups where one part might depend on completion of a previous part. They ideally involve the integration between multiple OpenStack services to exercise the touch points between them. For this first version we have chosen to run the Tempest smoke tests, a quick, lightweight automated set of tests that validate the basic functionality of OpenStack.

4.2.2. OpenDaylight Tests

For OpenDaylight testing we have decided to validate the OpenStack - OpenDaylight integration when the user preforms operations on the three core resources - networks, subnets and ports. Specifically, we are running tests for the creation and the deletion of networks, ports and subnets. The following describes the key steps for each test run when using the Robot Framework:

Create Network:

1. Check OpenStack Neutron for known networks
    a. Authenticate OpenStack to create session
    b. Perform GET request to retrieve current Neutron networks
    c. Verify that return code was 200 (GET request was successful)

   d. Log the result
2. Check OpenDaylight Networks
   a. Authenticate OpenDaylight to create session
   b. Perform GET request to retrieve current Neutron networks
   c. Verify that return code was 200 (GET request was successful)
   d. Log the result
3. Create Network in OpenStack
   a. Perform POST request to create a new Neutron network
   b. Verify that return code was 201 (POST request was successful and created element id was returned)
   c. Get created network id and description
   d. Log created network id and description
   e. Save created network id in variable for later usage
4. Check Network in OpenDaylight
   a. Perform GET request using saved network id to verify that the network is visible in OpenDaylight
   b. Verify that return code was 200 (GET request was successful)

Create Subnet:

1. Check OpenStack Subnets
   a. Authenticate OpenStack to create session
   b. Perform GET request to retrieve current Neutron subnets
   c. Verify that return code was 200 (GET request was successful)
   d. Log the result
2. Check OpenDaylight subnets
   a. Authenticate OpenDaylight to create session
   b. Perform GET request to retrieve current Neutron subnets
   c. Verify that return code was 200 (GET request was successful)
   d. Log the result
3. Create New subnet in OpenStack
   a. Perform POST request to create a new Neutron subnet
   b. Verify that return code was 201 (POST request was successful and created element id was returned)
   c. Get created subnet id and description
   d. Log created subnet id and description
   e. Save created subnet id in variable for later usage
4. Check New subnet in OpenDaylight
   a. Perform GET request using saved subnet id to verify that the subnet is visible in OpenDaylight
   b. Verify that return code was 200 (GET request was successful)

Create Port:

1. Check OpenStack ports
   a. Authenticate OpenStack to create session

   b. Perform GET request to retrieve current Neutron ports
   c. Verify that return code was 200 (GET request was successful)
   d. Log the result
2. Check OpenDaylight ports
   a. Authenticate OpenDaylight to create session
   b. Perform GET request to retrieve current Neutron ports
   c. Verify that return code was 200 (GET request was successful)
   d. Log the result
3. Create New Port in OpenStack
   a. Perform POST request to create a new Neutron ports
   b. Verify that return code was 201 (POST request was successful and created element id was returned)
   c. Get created port id and description
   d. Log created port id and description
   e. Save created port id in variable for later usage
4. Check New Port in OpenDaylight
   a. Perform GET request using saved port id to verify that the port is visible in OpenDaylight
   b. Verify that return code was 200 (GET request was successful)

Delete Network:

1. Delete a new Network in OpenStack
   a. Authenticate OpenStack to create session
   b. Perform DELETE request to delete saved network id
   c. Verify that return code was 204 (DELETE request was successful and no entity-body was returned)
   d. Log response content
2. Check Network is Deleted in OpenDaylight
   a. Authenticate OpenDaylight to create session
   b. Perform GET request using saved port id to verify that the network was deleted in OpenDaylight
   c. Verify that return code was 404 (requested element was not found)

Delete Subnet:

1. Delete a new Subnet in OpenStack
   a. Authenticate OpenStack to create session
   b. Perform DELETE request to delete saved subnet id
   c. Verify that return code was 204 (DELETE request was successful and no entity-body was returned)
   d. Log response content
2. Check a new subnet is deleted in OpenDaylight
   a. Authenticate OpenDaylight to create session

   b. Perform GET request using saved port id to verify that the subnet was deleted in OpenDaylight
   c. Verify that return code was 404 (requested element was not found)

Delete Port:

1. To delete Port in OpenStack
   a. Authenticate OpenStack to create session
   b. Perform DELETE request to delete saved port id
   c. Verify that rerun code was 204 (DELETE request was successful and no entity-body was returned)
   d. Log response content
2. Check Port is deleted in OpenDaylight
   a. Authenticate OpenDaylight to create session
   b. Perform GET request using saved port id to verify that the port was deleted in OpenDaylight
   c. Verify that return code was 404 (requested element was not found)

In addition a set of tests are being run to verify that the insertion and deletion of flows is being performed correctly. Flow rules can be based on source IP, destination IP, destination MAC address, VLAN ID and other parameters. An example test procedure that validates adding and removing a flow is described below:

Test suite for pushing/verify/remove a flow through RESTCONF:

1. Push a flow through REST-API
   a. Open premade configuration file
   b. Save file content to variable
   c. Perform PUT request to restconf using the variable as request body
   d. Verify that return code was 200 (PUT request was successful)
2. Verify after adding flow config
   a. Perform GET request to the Restconf using table id and flow id
   b. Verify that return code was 200 (GET request was successful – the flow exists)
   c. Compare response content with file sent to verify that the correct file was send
3. Verify flows after adding flow config on OVS
   a. Wait 1 second
   b. Open ssh connection
   c. Login
   d. Execute "ovs-ofctl dump-flows s1 -O OpenFlow13" in order to get switch flows
   e. Read switch output and verify it contains the input flow elements

   f. Close ssh connection
4. Remove a flow
   a. Perform DELETE request to the Restconf using table id and flow id to delete the flow
   b. Verify that return code was 200 (DELETE request was successful)
5. Verify after deleting flow config
   a. Perform GET request to the Restconf using table id
   b. Verify that the response does not contain the flow id
6. Verify flows after deleting flow config on OVS
   a. Open ssh connection
   b. Login
   c. Execute "ovs-ofctl dump-flows s1 -O OpenFlow13" in order to get the switch flows
   d. Read the switch output and verify it does not contain the input flow elements
   e. Close ssh connection

4.2.3. VNF Testing

As described in Deliverable 4.1 [13], there are four key types of network workloads:

- **Data Plane** – These workload types perform packet handling that involves input/output and read/write memory operations.
- **Control Plane** –These workloads relate to protocol exchanges including setup, session management, and termination.
- **Signal Processing** –These workloads are responsible for digital processing and are typically highly CPU intensive and delay-sensitive.
- **Storage** – These workloads have significant read and write to disk storage operations.

The VNFs being developed by T-NOVA and their workload classification are outlined in Table 4-2.

Table 4.2: T-NOVA VNF workload classifications

VNF	Data Plane	Control Plane	Signal processing	Storage
Security Appliance (vSA)	X			
Traffic Classification (vDPI)	X			
Session Boarder Controller (vSBC)		X		

Home Gateway (vHG)	X			
Video Transcoding Unit (vVTU)				X
Proxy as a Service (vPxaaS)				X

When testing, there are many factors that impact performance. It is important to establish performance baselines before adding complexity and determine what may be creating a performance bottlenecks through isolation. A step-wise process would involve:

When testing a virtualized function, there are two primary approaches:

- Map the function to physical NIC interfaces and test with traditional hardware-based test systems
- Test the function in the virtualised system by inserting testing into the virtualised system (primarily through running on a VM). This can also include a combination of virtual and physical test ports

Table 4.3: Factors affecting VNF performance in bare and virtualised deployments [10]:

are Metal	Virtualised Deployment
• Processor architecture • Extended instruction set (req. for crypto) • Clock rate • Size of data caches • Memory access speed • Memory latency • Inter-processor bus bandwidth • Number of cores on a processor • Large page support • I/O page support • TLB cache with large page support • I/O TLB cache with large page support • Size of TLB caches • Interrupt affinity • Layered memory cache • Deterministic allocation of threads in CPU • Deterministic memory allocation • Independent memory structures per thread • Inter-thread communications through memory pipeline structures • CPU isolation	• Processor architecture • Extended instruction set (req. for crypto) • Clock rate • Size of data caches • Memory access speed • Memory latency • Inter-processor bus bandwidth • Number of cores on a processor • Large page support • I/O page support • TLB cache with large page support • I/O TLB cache with large page support • Size of TLB caches • Interrupt affinity • Layered memory cache • Deterministic allocation of threads in CPU • Deterministic memory allocation • Independent memory structures per thread • Inter-thread communications through memory pipeline structures • CPU isolation

• DMA • Direct I/O access to processor cache • Flow affinity/steering by I/O devices • NIC acceleration capabilities	• Polling mode drivers • DMA • Direct I/O access to processor cache • Flow affinity/steering by I/O devices • NIC acceleration capabilities • Instructions to reduce the number of VM exits under certain common • operations • Second-level address translation services • Second-level address translation services for large pages • Second-level address translation services for I/O • Second-level address translation services large I/O pages • I/O interrupt remapping • Extension of processor caches with new fields to avoid cache eviction with • VM exits • Extension of processor TLB caches with new fields to avoid TLB flushes • Independent TX/RX queues for virtual machines • SR-IOV

Within Task 4.5 we are not interested in the test cases that are specific to the VNF functions but to use a subset of the WP5 developed VNFs to validate the correct function of the IVM layer stack that is related to VNFs such as typical operations on VNFs and their images (deploy/create, modify, stop, destroy), ability to provide connectivity between the deployed VNFs and operations on the VNF images (create, delete, download, list and update). The following section provides the test cases developed to validate the vTC VNF.

4.2.3.1. vTC VNF Tests

As outlined in deliverable 4.1 [13], a VNF Characterisation Framework was developed for automating the execution of test cases and benchmarks of VNF workloads.

On top of that framework, different test cases have been implemented. Specifically, in the context of Task 4.1, two test cases have been implemented to investigate and measure affinities for the different quantities and types of resources allocations in an iterative manner, as shown in Figure 4-1. The parameters that can be investigated are those which can be specified in the Heat template for a given VNF deployment. For a given test scenario, parameters values that will utilised during an experiment are defined in a configuration file which is used by the framework to generate a test of Heat templates to deploy the configurations of interest. The framework provides orchestration of the full test case lifecycle. A typical test case will comprise of template deployment, followed by VNF load application (e.g. traffic generation sent to the deployed VNF), collection of real time data on performance (e.g. network throughput), and VNF deletion. This process is repeated until all templates have been deployed. Development of the framework was completed as part of the Task 4.1 activities.

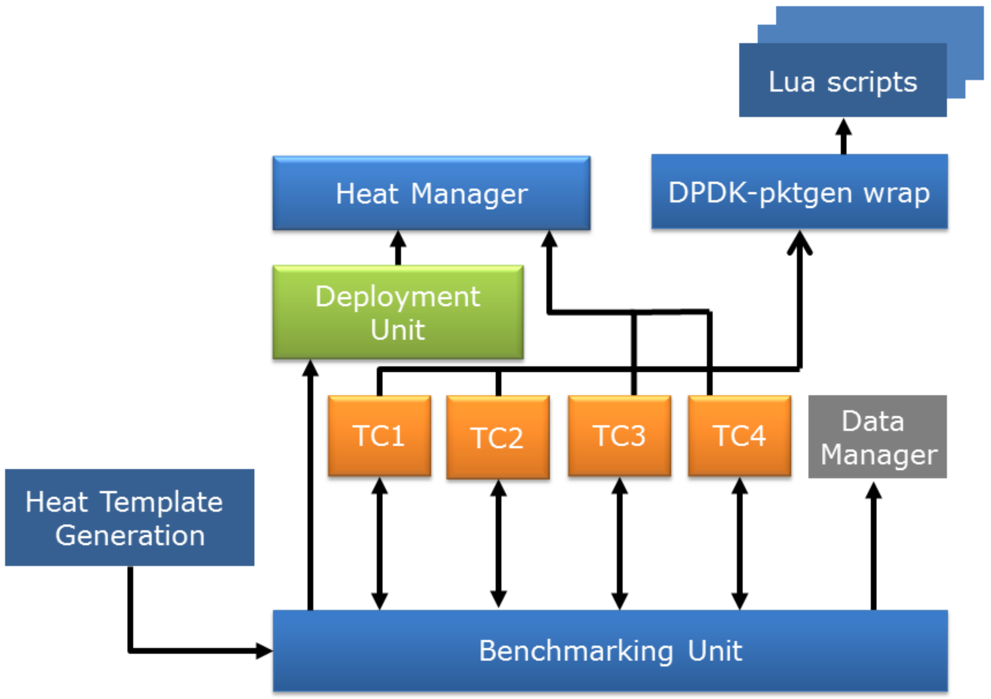


Figure 4-1: Architectural components of the VNF Characterisation framework

The four test cases that have been developed were defined in cooperation with OPNFV Yardstick project and form part of the contribution from T-NOVA to Yardstick.

The four cases are as follows:

1. Virtual Traffic Classifier Instantiation Test
2. Virtual Traffic Classifier Instantiation in the presence of Noisy Neighbours Test
3. Virtual Traffic Classifier Data Plane Throughput Benchmarking Test
4. Virtual Traffic Classifier Data Plane Throughput Benchmarking in presence of noisy neighbours Test

The definitions of these test cases are based on the ETSI specification ETSI GS NFV-TST001 - Validation of NFV Environments and Services.

Network throughput was measured using the RFC2544 benchmarking methodology for network interconnected devices (see Deliverable 4.1 section 8.2 for additional details).

The key focus from a Task 4.5 perspective has been on the development of these test cases, testing and deployment on the T-NOVA and Yardstick test beds. While the test case contributions to Task 4.5/Yardstick remain a work in progress, a set of key development activities have been completed in this task to date.

The initial version of the VNF characterisation framework was completed in D4.1 was further extended. A "BenchmarkBaseClass" class was developed to provide a reference for all benchmarks. This class provides the necessary definitions of the parameters required for a test case lifecycle.

Benchmark Feature Definition

The method that defines the features in a test case is as follows:

```
def get_features(self):
        features = dict()
        features['description'] =
              'Please implement the method "get_features" for
your benchmark'
        features['parameters'] = list()
        features['allowed_values'] = dict()
        features['default_values'] = dict()
        return features
```

The key aspects include:

- A description of the test case (a free text string that is user defined).
- A list of parameters, provided in the form of a list of strings containing the names of the parameters.
- The allowed values, which are represented as a dictionary; including all the permissible values for the specified parameters.
- The default values, which are represented as a dictionary and include the values that the parameters will be assigned in a test case where the user does not specify any value.

The values of the parameters are collected by the framework from the configuration file. The get_features method is used to over write by the benchmark class. An example is shown in following in the context of the "RFC2544ThroughputBenchmark" class.

```
PACKET_SIZE = 'packet_size'
VLAN_SENDER = 'vlan_sender'
VLAN_RECEIVER = 'vlan_receiver'
def get_features(self):
        """
        Returns the features associated to the benchmark
        :return:
        """
        features = dict()
        features['description']    =    'RFC    2544    Throughput
calculation'
        features['parameters']   =   [PACKET_SIZE,   VLAN_SENDER,
VLAN_RECEIVER]
        features['allowed_values'] = dict()
        features['allowed_values'][PACKET_SIZE] = ['64', '128',
'256', '512',
                                                   '1024',
'1280', '1514']
```

```
        features['allowed_values'][VLAN_SENDER]    =    map(str, 
range(-1, 4096))
        features['allowed_values'][VLAN_RECEIVER]   =   map(str, 
range(-1, 4096))
        features['default_values'] = dict()
        features['default_values'][PACKET_SIZE] = '1280'
        return features
```

This implementation requires the test case parameter values to be specified in the configuration file as follows:

```
[Testcase-parameters]
packet_size = 1280
vlan_sender = 1000
vlan_receiver = 1001
```

The initialisation method of the class is as follows:

```
def __init__(self, name, params):
    if not params:
        params = dict()
    if not isinstance(params, dict):
        raise ValueError("Parameters need to be provided in a 
dict")
    for param in self.get_features()['parameters']:
        if param not in params.keys():
            params[param]                                    = 
self.get_features()['default_values'][param]
    for param in self.get_features()['parameters']:
        if params[param] not in \
                (self.get_features())['allowed_values'][param]:
            raise ValueError('Value of parameter "' + param +
                             '" is not allowed')
    self.name = name
    self.params = params
```

This method is mainly dedicated to initialise the parameters given by the user and is based on the implementation of the get_features shown before.

Running a Test Case

Execution of a test case is performed by the framework (the benchmarking unit, specifically) calling the method “run”. The benchmarking class needs to implement the method according to the signature included in the BenchmarkBaseClass as shown in the following:

```
@abc.abstractmethod
def run(self):
    """
    This method executes the specific benchmark on the VNF 
already instantiated
    :return: list of dictionaries (every dictionary contains 
the results of a data point
    """
    raise NotImplementedError("Subclass must implement abstract 
method")
```

An example of test case execution relating to RFC2544 is shown in the following:

```
def run(self):
    """
    Sends and receive traffic according to the RFC methodology
in order to
    measure the throughput of the workload
    :return: Results of the testcase (type: dict)
    """
    ret_val = dict()
    packet_size = self._extract_packet_size_from_params()
    ret_val[PACKET_SIZE] = packet_size
    # Packetgen management
    packetgen = dpdk.DpdkPacketGenerator()
    self._configure_lua_file()
    packetgen.init_dpdk_pktgen(dpdk_interfaces=2,
                               pcap_file_0='packet_' +
                               packet_size + '.pcap',
                               pcap_file_1='igmp.pcap',
                               lua_script='rfc2544.lua',
                               vlan_0=self.params[VLAN_SENDER],

vlan_1=self.params[VLAN_RECEIVER])
    common.LOG.debug('Start the packet generator - packet size:
' +
                     str(packet_size))
    packetgen.send_traffic()
    common.LOG.debug('Stop the packet generator')
    # Result Collection
    results = self._get_results()
    for metric_name in results.keys():
        ret_val[metric_name] = results[metric_name]
    self._reset_lua_file()
    return ret_val
```

Once the size of the packets to be used is assigned, the method interacts with the DPDK packet generator library, in order to initialise the packet generator with the required parameters and then triggers execution of the traffic generator. This phase of the test case lifecycle is managed by LUA script indicated in the parameter listings. Once the execution of the traffic generation is completed the method collects the results and returns them. The results are automatically managed and stored by the Benchmarking Unit via the Data Manager.

Initialisation and Finalisation of a Test Case

The benchmarking class defined by the user needs to implement an abstract method called init. The signature of this method in the BenchmarkBaseClass is as follows:

```
@abc.abstractmethod
def init(self):
    """
    Initializes the benchmark
    :return:
    """
    raise NotImplementedError("Subclass must implement abstract
method")
```

A useful example on how to extend this is in the “MultitenancyThroughputBenchmark” test case, where the deployment of noisy neighbors is implemented.

```
def init(self):
        """
        Initialize the benchmark
        return: None
        """
        common.replace_in_file(self.lua_file, 'local out_file =
""', 'local out_file = "' + self.results_file + '"')
        heat_param = dict()
        heat_param['cores'] = self.params['number_of_cores']
        heat_param['memory'] = self.params['amount_of_ram']
        for              i              in              range(0,
int(self.params['num_of_neighbours'])):
            stack_name = self.stack_name + str(i)

common.DEPLOYMENT_UNIT.deploy_heat_template(self.template_file,
stack_name, heat_param)
            self.neighbor_stack_names.append(stack_name)
```

This method triggers Heat in order to deploy a given number of neighbours (user defined through configuration file) with a given amount of stress on the CPU and the RAM (user defined as well). For this test case finalisation has also been implemented, in order to terminate the virtual machines deployed during the initialisation phase. The base class finalisation method is as follows:

```
@abc.abstractmethod
    def finalize(self):
        """
        Finalizes the benchmark
        :return:
        """
        raise   NotImplementedError("Subclass   must   implement
abstract method")
```

This method has been overwritten in the MultitenancyThroughputBenchmark class as follows:

```
def finalize(self):
        """
        Finalizes the benchmark
        return: None
        """
        common.replace_in_file(self.lua_file, 'local out_file =
"' + self.results_file +
                               '"', 'local out_file = ""')
        # destroy neighbor stacks
        for stack_name in self.neighbor_stack_names:

common.DEPLOYMENT_UNIT.destroy_heat_template(stack_name)
        self.neighbor_stack_names = list()
```

Unit Testing

To interrogate and validate the functionality of the test cases an initial suite of unit tests has been developed and executed. The purpose of unit testing is to securitise the smallest testable parts of an application. Unit tests focus on the characteristics that are vital to expected behaviour of the unit under test. While unit testing in time-consuming and tedious it is necessary to ensure the quality of the application code.

OPNFV Test lab Deployment

To validate the functional characteristics of the VNF Framework and the four test cases developed they were deployed in an OPNFV test lab hosted by Ericson in Montreal, Canada. The OpenStack environment used to host the framework and test cases was deployed using Fuel as per OPNFV requirements for automated and reproducible deployment of components. Fuel is an open source deployment and management tool for OpenStack which provides an intuitive, GUI-driven experience for deployment and management of OpenStack. The key focus of Fuel is to bring consumer-grade simplicity to streamline and accelerate the complex, and error-prone process of deploying, testing and maintaining various configuration flavours of OpenStack at scale. The configuration of the testbed after the initial OpenStack deployment is shown in Figure 4-2. Configuration of the OPNFV testbed and setup of continuous integration testing using Jenkins remains a work in progress.

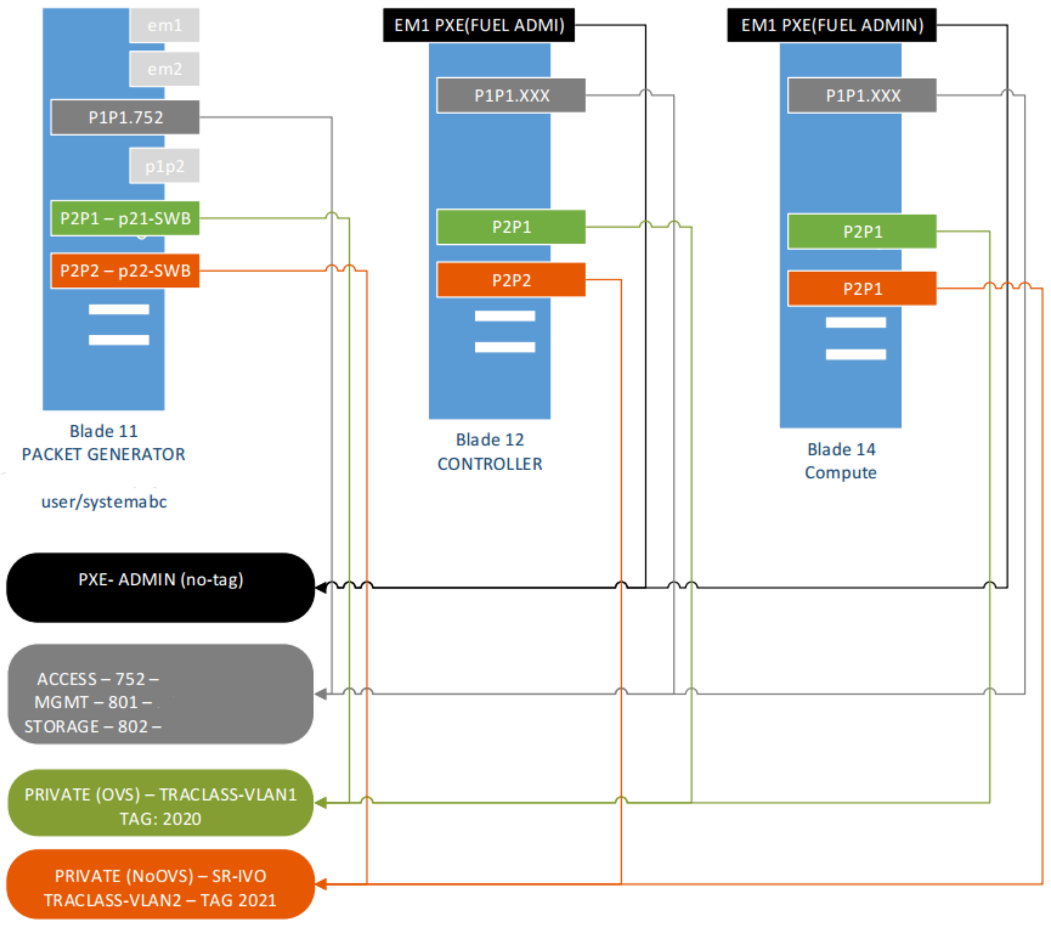


Figure 4-2: Yardstick testbed deployment configuration

Future Work

The step in the development of the VNF framework is the implementation of an API API interface. The purpose of the API is to allow another application or script to call it and execute test cases available within the framework. Initially the API will support integration with Yardstick framework allow the framework to call the VNF framework characterisation framework and request execution of one of the four test cases implemented.

In addition, the framework will need to provide support for OPNFV's results repository. This repository provides a centralised repository for all OPNFV implemented test cases and features a dashboard to show the results collected during the tests. Design and implementation of repository support is an on-going activity. An additional capability that potentially will be added is support for other packet generators. A leading candidate is Moongen [9], which is a packet generator based on DPDK that supports the generation of 10Gbps traffics streams, but also provide key network metrics such as latency and jitter, in addition to the throughput.

4.2.4. T-NOVA Testing Dashboard

Within the context of Task 4.5 a dashboard is being developed that enables the testing of an IVM layer deployment. The dashboard application can be installed in a remote machine, separate from the T-NOVA infrastructure and provide the means to validate a correctly installed T-NOVA IVM layer stack. The functions provided through the dashboard are in two main pages, Testing Environment and Tests. Within the Testing Environment, the user can setup the testing environment in their machine (install all the tools, libraries and frameworks required for testing), check if the environment is already installed and remove the testing environment. Within the test pages, the user can run tests that verify the installation and function of the T-NOVA IVM deployment by executing the OpenStack, OpenDaylight and VNF related tests described in the previous sections.

4.2.4.1. Implementation

The Testing Dashboard was developed as a web application using Python (Flask Framework) for the back-end and HTML, CSS (Bootstrap), Javascript (JQuery) for the front-end. The user interface was based on "SB Admin 2" jQuery – Bootstrap Theme. The main frameworks used in the development of the dashboard are:

Python - Flask

Flask is called a micro framework because it does not presume or force a developer to use a particular tool or library. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Javascript – JQuery

jQuery is a cross-platform Javascript library designed to simplify the client-side scripting of HTML. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.

HTML, CSS – Bootstrap

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML and CSS based design templates for typography, forms, buttons, navigation and other interface components, as well as optional Javascript extensions. It aims to ease the development of dynamic websites and web applications.

4.2.4.2. Application Architecture

The application follows a Model-View-Controller (MVC) architecture. The main components of a Model-View-Controller (MVC) application are described below:

Model - The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

View - A presentation of data in a particular format, triggered by a controller's decision to present the data.

Controller - The controller is responsible for responding to user input and handles interactions with the data model objects. The controller receives the input, validates it t and then performs the business operation that modifies the state of the data model.

The user, by visiting the URL, requests to view a page or load content. This request is sent to the server where it is handled by a specific **Controller.** The Controller makes use of the necessary **Models** and produces the result which is sent back to the user via the **View** which is the generated HTML page.

4.2.4.3. Using the Testing Dashboard

Testing Environment

The "Testing Environment" page, this is where the testing environment can be configured to run to the user's system. Figure 4-3 depicts a screenshot of the Test

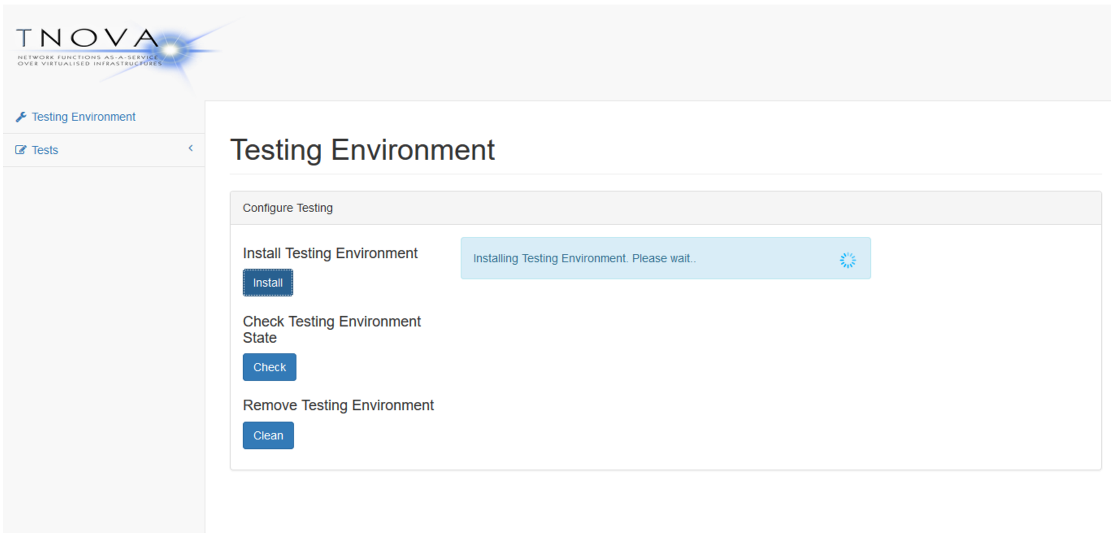

TNOVA
NETWORK FUNCTIONS AS-A-SERVICE OVER VIRTUALISED INFRASTRUCTURES
Testing Environment
Tests
Testing Environment
Configure Testing
Install Testing Environment
Install
Installing Testing Environment. Please wait..
Check Testing Environment State
Check
Remove Testing Environment
Clean

Figure 4-3: Installation of the testing environment within the dashboard

Environment page.

As shown, there are three options, “Install Testing Environment”, “Check Testing Environment State”, “Remove Testing Environment”. As their names dictate by pressing the corresponding buttons the application performs the following actions:

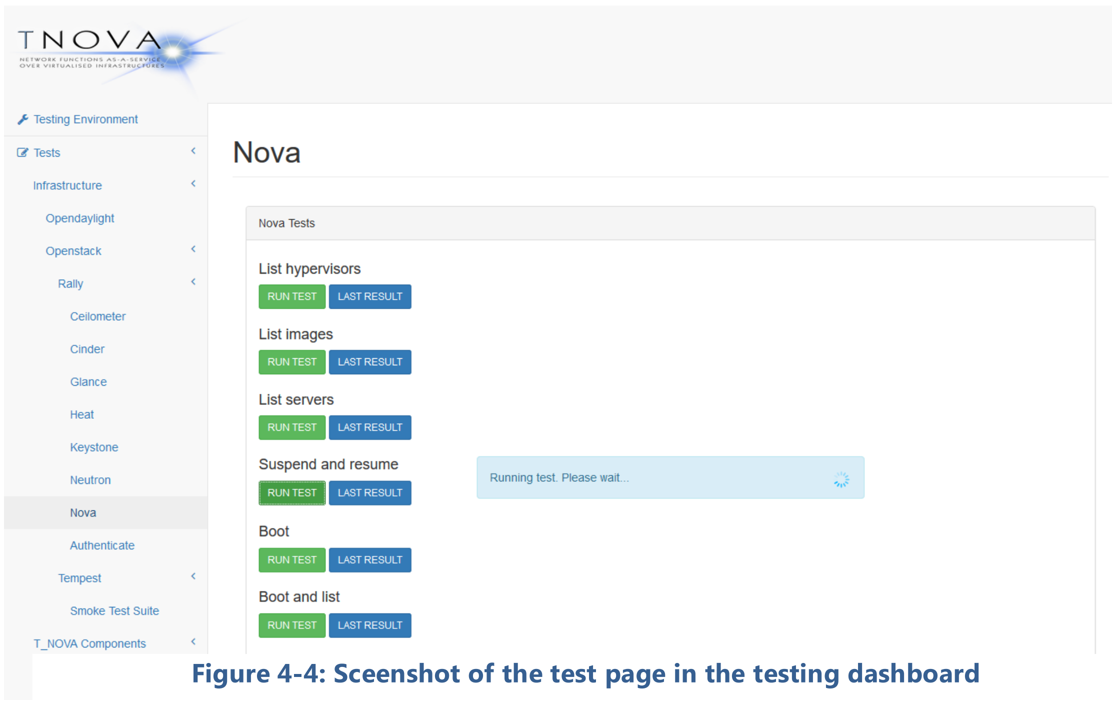
Install - Necessary dependencies and testing tools are downloaded and installed to the system. Specifically, OpenStack Rally, Tempest, Robot Framework, Open vSwitch and Mininet are installed. The installation can take up to 5 minutes.

Check - The application checks current system configuration to ensure that the necessary components are installed and configured properly to allow testing operations to take place. If the required tools and dependencies are not installed a relative message appears to notify the user.

Clean - The application removes saved files and folders created by the installation process.

Tests

As shown in Figure 4-4, the “Tests” page contains all the available tests categorized by the testing suite in which they belong, e.g. OpenStack Rally, Robot Framework, etc. By clicking on “Tests” option on the sidebar the user comes across a multi-level drop down list. By clicking on any of the list's elements the tests shown in the main content of the page are filtered accordingly.



Running test. Please wait...

Figure 4-4: Sceenshot of the test page in the testing dashboard

Each test comes with two buttons; “RUN TEST” and “LAST RESULT”. When the RUN TEST button is pressed, the application executes the test validating the remote T-NOVA IVM layer deployment. When the test is completed a message appears to notify the user. Figure 4-5 describes the workflow followed for executing a Rally test and getting back the result. We have configured a server (testing machine) which is separate from the T-NOVA IVM layer deployment. Within the testing machine we have installed the testing environment, including Rally, Tempest and Robot

Framework. The workflow depicted relates to the execution of the Rally tests. The application uses the Rally test suite to execute the task that corresponds to the desired test; the task is then saved to the Rally database. From there the task id is later retrieved to be used in the rally command responsible for exporting test results in html format. The exported html file is then parsed and saved to the application database.

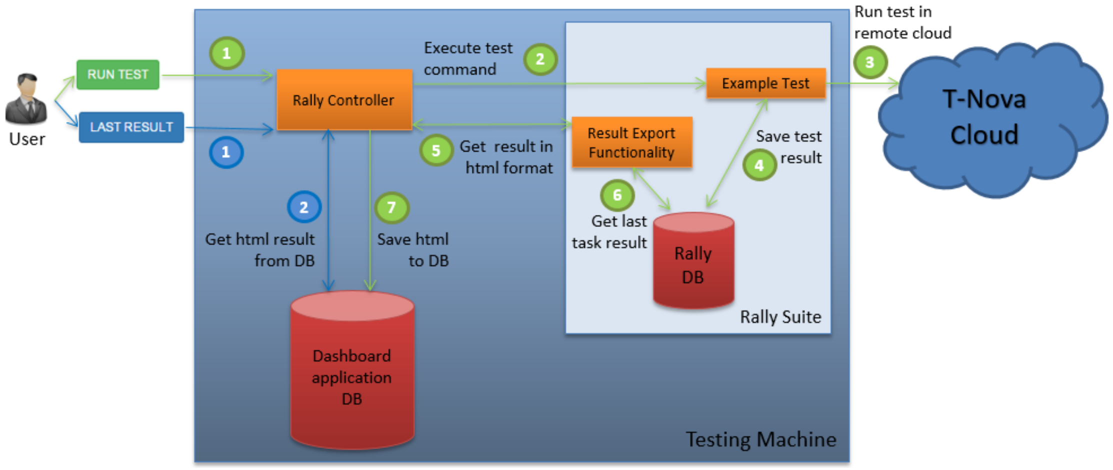
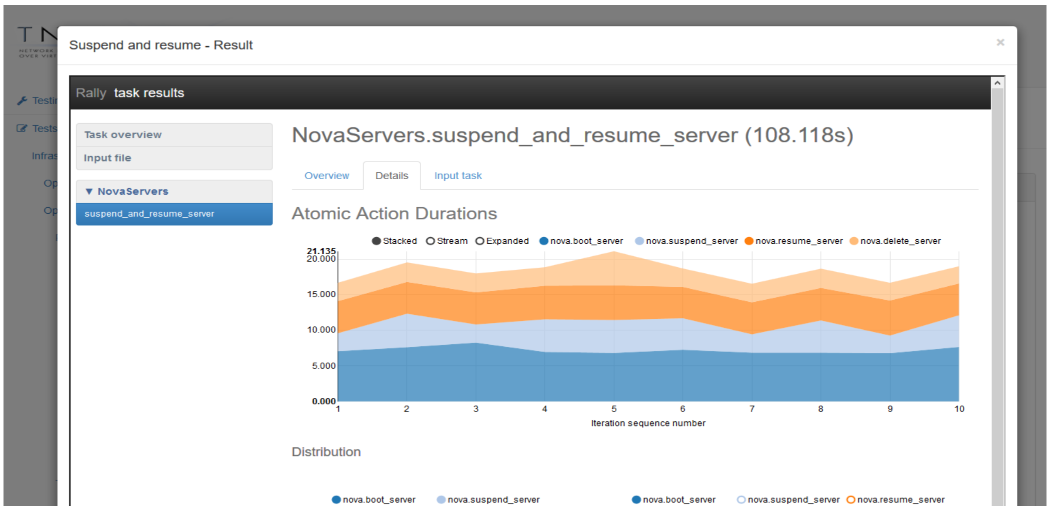


Figure 4-5: Testing Workflow

The user can obtain the last result from the execution of a particular test by pressing the button LAST RESULT. The application loads from the application database the last result for this test. The result is shown in a modal window which contains an HTML Inline Frame Element (iframe) where the test suite html output file is loaded. Figure 4-4 and Figure 4-6 show examples of the results returned to the user after executing Suspend and Resume Server and Remove Flow tests.


Suspend and resume - Result

Rally task results

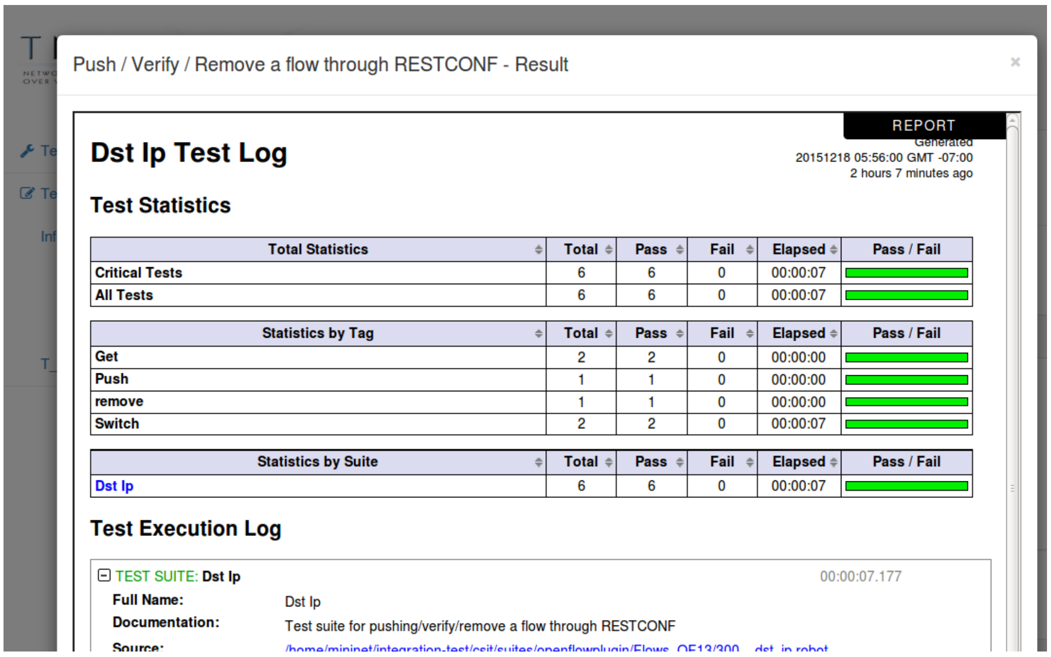
NovaServers.suspend_and_resume_server (108.118s)

Atomic Action Durations

Distribution

Task overview
Input file
▼ NovaServers
suspend_and_resume_server
Overview
Details
Input task
● Stacked ○ Stream ○ Expanded ● nova.boot_server ● nova.suspend_server ● nova.resume_server ● nova.delete_server
21.135
20.000
15.000
10.000
5.000
0.000
1 2 3 4 5 6 7 8 9 10
Iteration sequence number
● nova.boot_server ● nova.suspend_server
● nova.boot_server ○ nova.suspend_server ○ nova.resume_server

Figure 4-6: Sceenshot showing the result of executing the Suspend and Resume Servers test for OpenStack Nova



Push / Verify / Remove a flow through RESTCONF - Result

REPORT

Generated
20151218 05:56:00 GMT -07:00
2 hours 7 minutes ago

Dst Ip Test Log

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	6	6	0	00:00:07	🟩
All Tests	6	6	0	00:00:07	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
Get	2	2	0	00:00:00	🟩
Push	1	1	0	00:00:00	🟩
remove	1	1	0	00:00:00	🟩
Switch	2	2	0	00:00:07	🟩

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Dst Ip	6	6	0	00:00:07	🟩

Test Execution Log

TEST SUITE: **Dst Ip** 00:00:07.177

Full Name:	Dst Ip
Documentation:	Test suite for pushing/verify/remove a flow through RESTCONF
Source:	/home/mininet/integration-test/csit/suites/openflowplugin/Flows_OF13/300__dst_ip.robot

Figure 4-7: Screenshot depicting the result of the Remove Flow test

5. CONCLUSIONS AND FUTURE WORK

Task 4.5 is focused on the implementation of a testbed which provides an implementation of the functional entities that comprise the IVM layer of the T-NOVA system namely the NFVI, VIM and WICM. The task has a number of key primary dependencies within WP4 namely Tasks 4.1 to 4.4. In addition, there is also a dependency on WP5 in terms of providing VNFs which can be used to test and validate the outputs from the WP4 tasks and to interrogate the overall functional operation of the Task 4.5 testbed.

The key technologies selections from a hardware and software perspective have been described. Open source technologies have been used exclusively such as OpenStack for the cloud computing environment and OpenDaylight for the SDN control. From a hardware perspective commercial of the shelf (COTS) X86 servers have been used in the implementation of the testbed. However, the use of additional hardware platforms to provide specialised acceleration capabilities namely FPGA is being actively investigated and may form part of the final testbed implementation. Details on how the selected are being integrated together in the testbed are also provided together with a description of the various open source tools such as Rally, Tempest etc. that have selected and deployed to provide automated testing and validation of component functional and performance. Additionally, the approach that is being developed to create a robust and reproducible deployment process is described. This included the development of testing dashboard to provide coordination of IVM deployments and functional component testing through an intuitive user interface.

The key components of the cloud environment adopted in the testbed have been described. The implementation adopted is designed to adaptable in order to support various usage scenario such as experimental deployment, production deployment etc. In addition, the key components developed by T-NOVA have been described namely the virtualised SDN control plane, the monitoring framework, SDK for SDN with SFC and VNF Characterisation Framework. Details of how these components will be deployed and integrated into the testbed have been described.

The next steps for Task 4.5 Infrastructure integration and deployment involve the deployment of the T-NOVA components and the validation and testing of their function. Specifically, detailed documentation for the deployment of the Virtualised SDN Control Plane, the Monitoring Framework, the SDK for SDN, the SFC component will be produced. In addition, a set of test cases validating the proper deployment and function of the T-NOVA component will be implemented. Completing this step will help us to make safer and more realistic assumptions on the roles planning for the deployment of the T-NOVA IVM layer, affecting both the number of nodes required for the deployment and the hardware requirements needed. An additional step will be to include the T-NOVA components tests in the testing dashboard application. Moreover, effort will be made towards utilising open source tools that allow the automated deployment and integration of the T-NOVA IVM layer components, facilitating greatly the deployment procedures. We have already reviewed the available automation provisioning and configuration management tools and we lean towards the use of Foreman and Puppet for this purpose. All the aforementioned advances will be reflected in the final version of this deliverable.

6. ANNEXES

6.1. Annex A – OpenStack installation

In this Annex we describe the procedure for installing OpenStack Liberty.

Network Time Protocol (NTP)

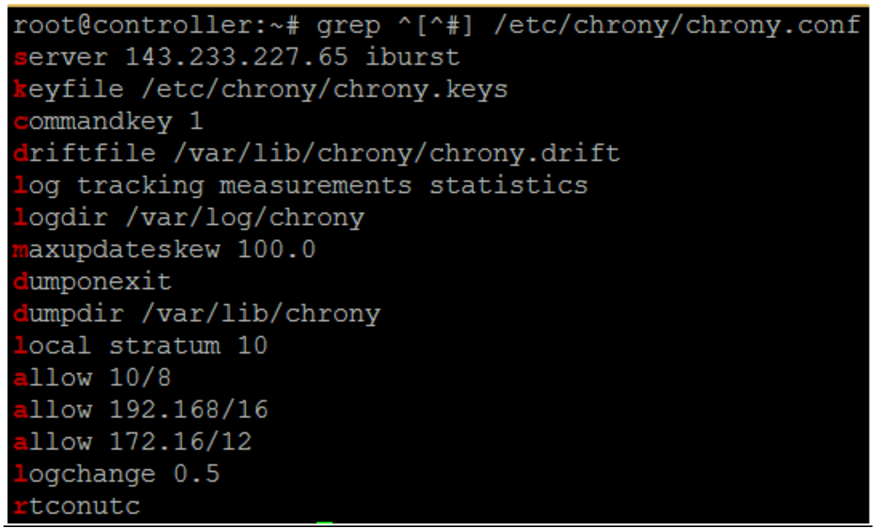
Chrony is an implementation of NTP to synchronize services among nodes. Controller Node is the reference of the compute nodes.

In all nodes:

```
# apt-get install chrony
```

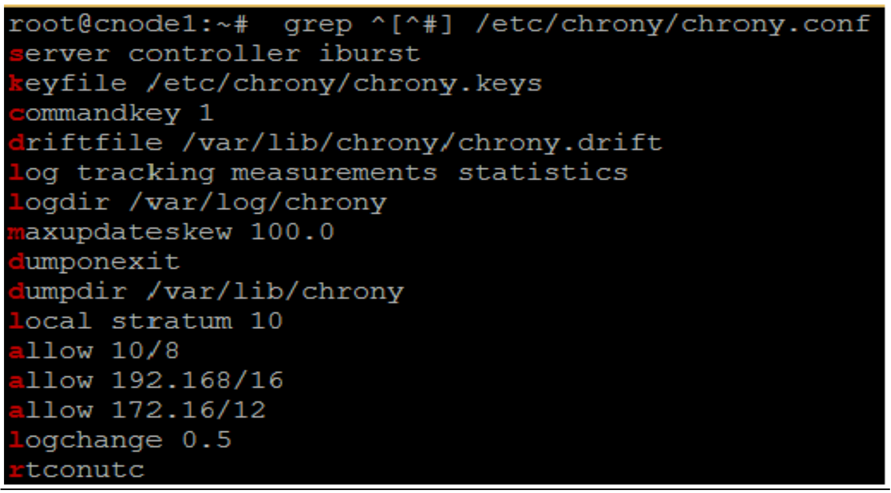
Edit and configure /etc/chrony/chrony.conf :

Controller



```
root@controller:~# grep ^[^#] /etc/chrony/chrony.conf
server 143.233.227.65 iburst
keyfile /etc/chrony/chrony.keys
commandkey 1
driftfile /var/lib/chrony/chrony.drift
log tracking measurements statistics
logdir /var/log/chrony
maxupdateskew 100.0
dumponexit
dumpdir /var/lib/chrony
local stratum 10
allow 10/8
allow 192.168/16
allow 172.16/12
logchange 0.5
rtconutc
```

Nodes



```
root@cnode1:~#  grep ^[^#] /etc/chrony/chrony.conf
server controller iburst
keyfile /etc/chrony/chrony.keys
commandkey 1
driftfile /var/lib/chrony/chrony.drift
log tracking measurements statistics
logdir /var/log/chrony
maxupdateskew 100.0
dumponexit
dumpdir /var/lib/chrony
local stratum 10
allow 10/8
allow 192.168/16
allow 172.16/12
logchange 0.5
rtconutc
```

Identity Service

The Identity service provides a single point of integration for managing authentication, authorization, and service catalog services.

The other OpenStack services need to collaborate with it. When an OpenStack service receives a request from a user, it checks with the Identity service whether the user is authorized to make the request.

Install and Configure

Before we configure the Identity service, we create the database and the administration token.

- Connect as root to the Database server

```
# mysql -u root -p
```

- Create the Database

```
CREATE DATABASE keystone;
```

- Grant proper access to the database

```
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
IDENTIFIED BY'KEYSTONE_DBPASS';

GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY \
'KEYSTONE_DBPASS';
```

Install and configure components

1. Disable the keystone service from starting automatically after installation:

```
# echo "manual" > /etc/init/keystone.override
```

2. Install the packages:

```
#apt-get install keystone apache2 libapache2-mod-wsgi memcached  \
python-memcache
```

3. Edit the /etc/keystone/keystone.conf file
   (??)
4. Populate the Identity service database:

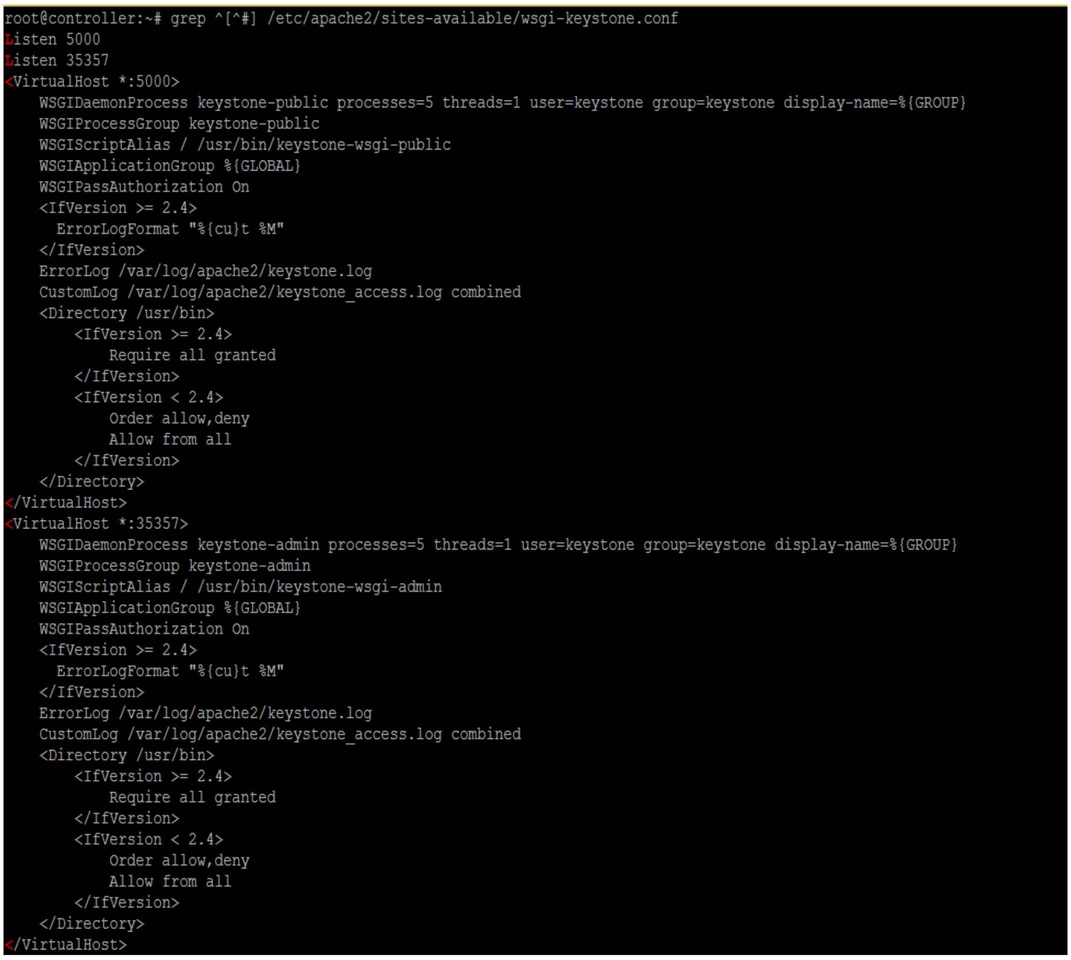
```
# su -s /bin/sh -c "keystone-manage db sync" keystone
```

Configure the Apache HTTP server

1. Edit the /etc/apache2/apache2.conf file and configure the ServerName option to reference the controller node:

```
ServerName controller
```

2. Create the /etc/apache2/sites-available/wsgi-keystone.conf file:



```
root@controller:~# grep ^[^#] /etc/apache2/sites-available/wsgi-keystone.conf
Listen 5000
Listen 35357
<VirtualHost *:5000>
    WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone group=keystone display-name=%{GROUP}
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /usr/bin/keystone-wsgi-public
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    <IfVersion >= 2.4>
      ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/keystone_access.log combined
    <Directory /usr/bin>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
    </Directory>
</VirtualHost>
<VirtualHost *:35357>
    WSGIDaemonProcess keystone-admin processes=5 threads=1 user=keystone group=keystone display-name=%{GROUP}
    WSGIProcessGroup keystone-admin
    WSGIScriptAlias / /usr/bin/keystone-wsgi-admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    <IfVersion >= 2.4>
      ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/keystone_access.log combined
    <Directory /usr/bin>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
    </Directory>
</VirtualHost>
```

3. Enable the Identity service virtual hosts:

```
# ln -s /etc/apache2/sites-available/wsgi-keystone.conf \
/etc/apache2/sites-enabled
```

Finalize the installation

1. Restart the Apache HTTP server:

```
# service apache2 restart
```

2. By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

```
# rm -f /var/lib/keystone/keystone.db
```

Create the service entity and API endpoints

The Identity service provides a catalog of services and their locations. Each service that you add to your OpenStack environment requires a service entity and several API endpoints in the catalog.

1. Configure the authentication token:

```
# export OS_TOKEN=ADMIN_TOKEN
```

2. Configure the endpoint URL:

```
# export OS_URL=http://controller:35357/v3
```

3. Configure the Identity API version:

```
# export OS_IDENTITY_API_VERSION=3
```

4. Create the service entity for the Identity service:

```
# openstack service create --name keystone --description "OpenStack \
identity" identity
```

5. Create the Identity service API endpoints:

```
# openstack endpoint create --region RegionOneidentity public \
http://$controller_public_ip:5000/v2.0
# openstack endpoint create --region RegionOne identity internal \
http://$controller_mgmt_ip:5000/v2.0
# openstack endpoint create --region RegionOneidentity admin \
http://$controller_mgmt_ip:5000/v2.0
```

Create projects, users, and roles

The Identity service provides authentication services for each OpenStack service. The authentication service uses a combination of domains, projects (tenants), users, and roles.

1. Create an administrative project, user, and role for administrative operations in your environment
   Create the admin project:

```
# openstack project create --domain default --description "Admin Project" \
admin
```

   Create the admin user:

```
# openstack user create --domain default--password-prompt admin
```

   Create the admin role:

```
# openstack role create admin
```

   Add the admin role to the admin project and user:

```
# openstack role add --project admin --user admin admin
```

2. Use a service project that contains a unique user for each service that you add to your environment.

   Create the service project:

```
# openstack project create --domain default --description "Service Project" \
service
```

3. Regular (non-admin) tasks should use an unprivileged project and user. In our environment we create project and user per VNF, for example the VHG project and user for the vHome Gateway VNF.

   Create the VHG project:

```
# openstack project create --domain default --description “VHG Project"  VHG
```

   Create the VHG user:

```
# openstack user create --domain default --password-prompt VHG
```

   Create the user role:

```
# openstack role create user
```

Add the user role to the VHG project and user:

```
# openstack role add --project VHG --user VHG user
```

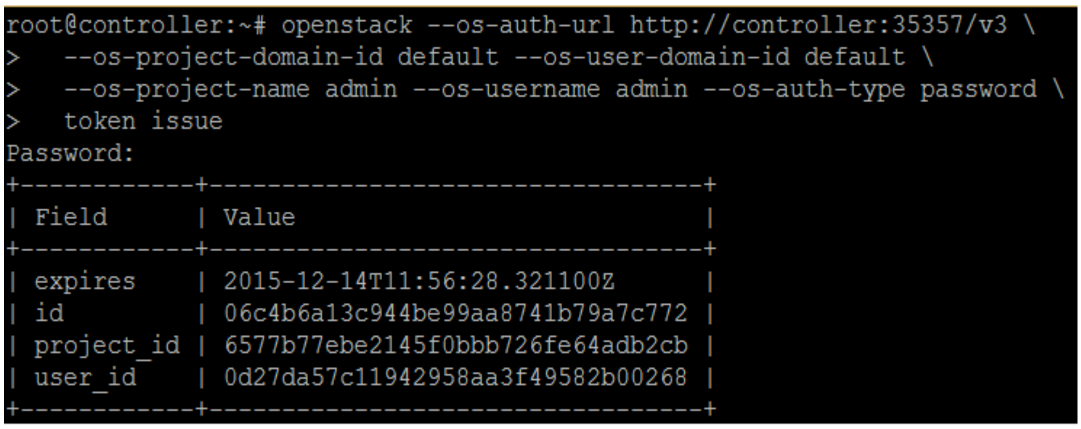
Verify operation

Disable, for security reasons, the temporary authentication token mechanism with removing admin_token_auth from the [pipeline:public_api], [pipeline:admin_api], and [pipeline:api_v3] sections in the /etc/keystone/keystone-paste.ini file.

1. Unset the temporary OS_TOKEN and OS_URL environment variables:

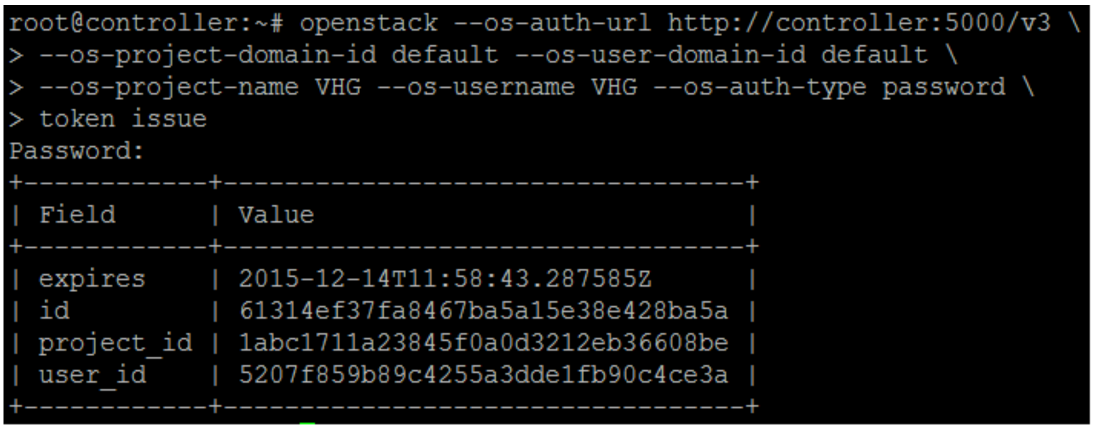
```
# unset OS_TOKEN OS_URL
```

2. As the admin user, request an authentication token:



```
root@controller:~# openstack --os-auth-url http://controller:35357/v3 \
>   --os-project-domain-id default --os-user-domain-id default \
>   --os-project-name admin --os-username admin --os-auth-type password \
>   token issue
Password:
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| expires    | 2015-12-14T11:56:28.321100Z      |
| id         | 06c4b6a13c944be99aa8741b79a7c772 |
| project_id | 6577b77ebe2145f0bbb726fe64adb2cb |
| user_id    | 0d27da57c11942958aa3f49582b00268 |
+------------+----------------------------------+
```

3. As VHG user, request an authentication toke:



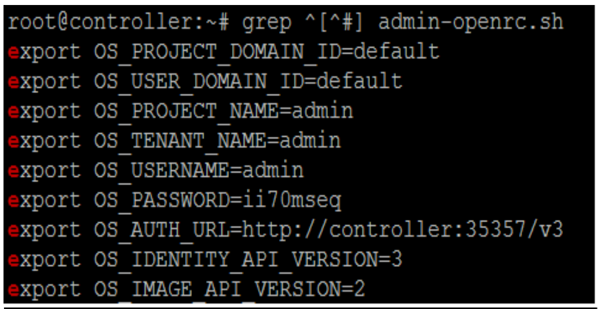
```
root@controller:~# openstack --os-auth-url http://controller:5000/v3 \
> --os-project-domain-id default --os-user-domain-id default \
> --os-project-name VHG --os-username VHG --os-auth-type password \
> token issue
Password:
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| expires    | 2015-12-14T11:58:43.287585Z      |
| id         | 61314ef37fa8467ba5a15e38e428ba5a |
| project_id | 1abc1711a23845f0a0d3212eb36608be |
| user_id    | 5207f859b89c4255a3dde1fb90c4ce3a |
+------------+----------------------------------+
```

Create OpenStack client environment scripts

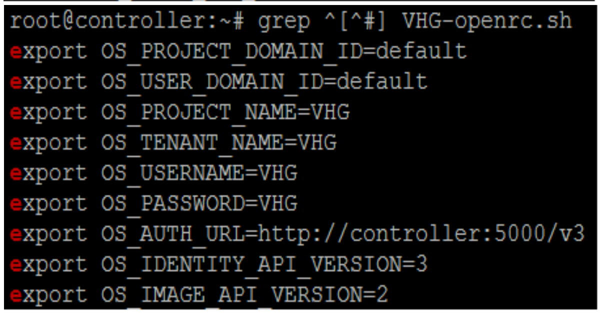
OpenStack supports simple client environment scripts also known as openrc files. These scripts typically contain common options for all clients, but also support unique options.

Creating the scripts

Create client environment scripts for the admin and VHG projects and users. Future portions of this guide reference these scripts to load appropriate credentials for client operations.



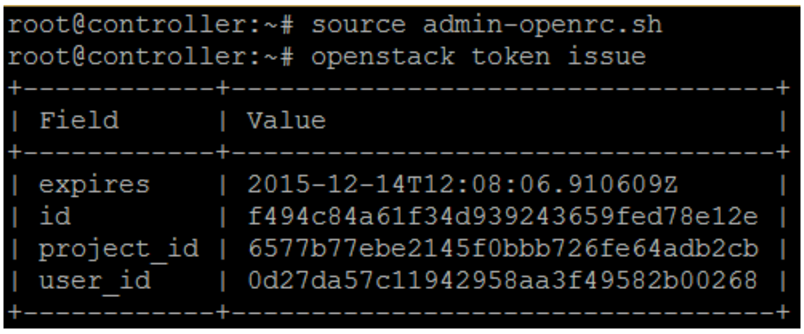
```
root@controller:~# grep ^[^#] admin-openrc.sh
export OS_PROJECT_DOMAIN_ID=default
export OS_USER_DOMAIN_ID=default
export OS_PROJECT_NAME=admin
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ii70mseq
export OS_AUTH_URL=http://controller:35357/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```



```
root@controller:~# grep ^[^#] VHG-openrc.sh
export OS_PROJECT_DOMAIN_ID=default
export OS_USER_DOMAIN_ID=default
export OS_PROJECT_NAME=VHG
export OS_TENANT_NAME=VHG
export OS_USERNAME=VHG
export OS_PASSWORD=VHG
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

To run clients as a specific project and user, you can simply load the associated client environment script prior to running them.

Load the admin-openrc.sh file to populate environment variables with the location of the Identity service and the admin project and user credentials:



```
root@controller:~# source admin-openrc.sh
root@controller:~# openstack token issue
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| expires    | 2015-12-14T12:08:06.910609Z      |
| id         | f494c84a61f34d939243659fed78e12e |
| project_id | 6577b77ebe2145f0bbb726fe64adb2cb |
| user_id    | 0d27da57c11942958aa3f49582b00268 |
+------------+----------------------------------+
```

<u>Add the Image service</u>

The OpenStack Image service (glance) enables users to discover, register, and retrieve virtual machine images. It offers a REST API that enables you to query virtual machine image metadata and retrieve an actual image.

<u>Install and configure</u>

1. Create glance database, service credentials, and API endpoints

   Create the database following the next steps

   - Connect as root to the Database server

```
# mysql -u root –p
```

   - Create the Database

```
CREATE DATABASE glance;
```

   - Grant proper access to the database

```
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' IDENTIFIED BY \
'GLANCE_DBPASS';
```

```
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY \
'GLANCE_DBPASS';
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. Create the service credentials

   Create the glance user:

```
# openstack user create --domain default --password-prompt glance
```

   Add the admin role to the glance user and service project:

```
# openstack role add --project service --user glance admin
```

   Create the glance service entity:

```
# openstack service create --name glance --description "OpenStack Image \
service" image
```

4. Create the Image service API endpoints:

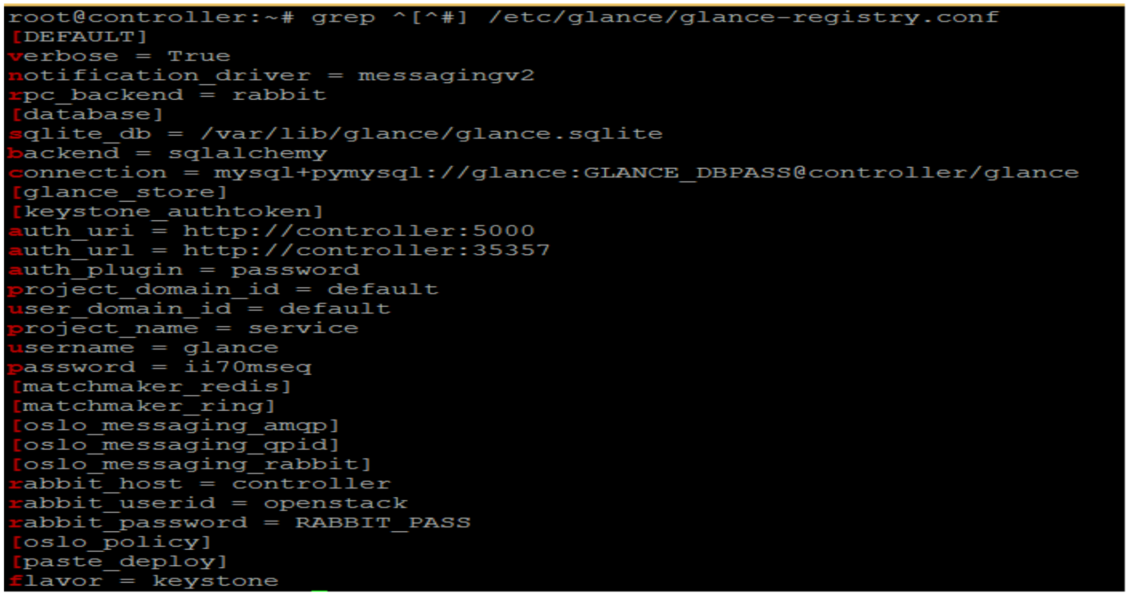
```
# openstack endpoint create --region RegionOne image public \
http://$controller_pub_ip:9292
# openstack endpoint create --region RegionOneimage internal \
http://$controller_mgmt_ip:9292
# openstack endpoint create --region RegionOneimage admin \
http://$controller_mgmt_ip:9292
```

Install and configure components

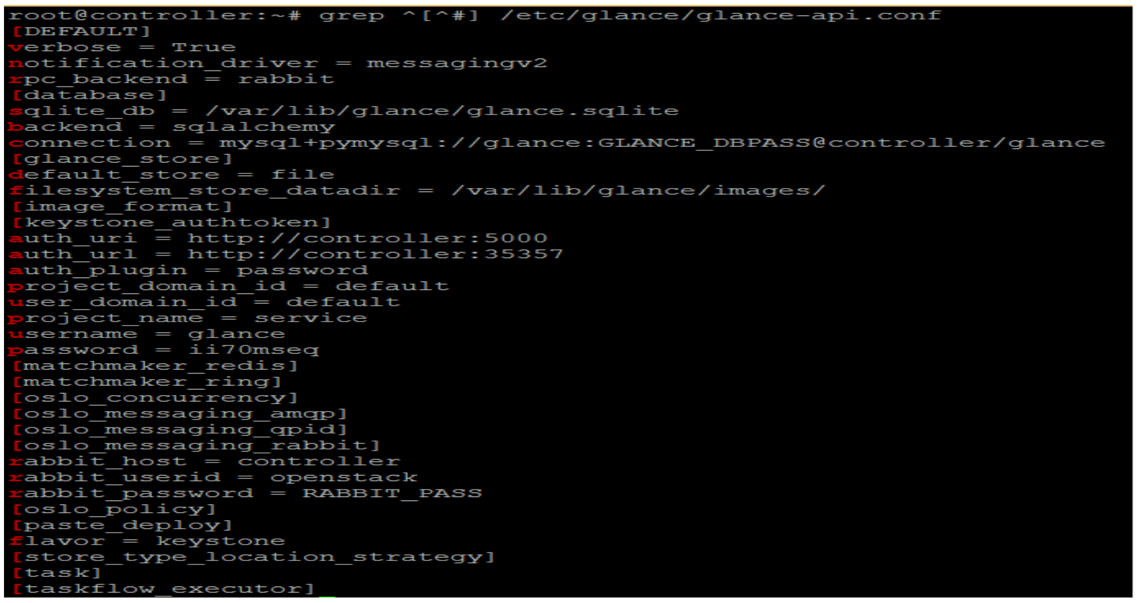
1. Install the packages:

```
# apt-get install glance python-glanceclient
```

2. Edit the /etc/glance/glance-api.confand etc/glance/glance-registry.conf files



```
root@controller:~# grep ^[^#] /etc/glance/glance-registry.conf
[DEFAULT]
verbose = True
notification_driver = messagingv2
rpc_backend = rabbit
[database]
sqlite_db = /var/lib/glance/glance.sqlite
backend = sqlalchemy
connection = mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
[glance_store]
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = glance
password = ii70mseq
[matchmaker_redis]
[matchmaker_ring]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
[paste_deploy]
flavor = keystone
```



```
root@controller:~# grep ^[^#] /etc/glance/glance-api.conf
[DEFAULT]
verbose = True
notification_driver = messagingv2
rpc_backend = rabbit
[database]
sqlite_db = /var/lib/glance/glance.sqlite
backend = sqlalchemy
connection = mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
[glance_store]
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
[image_format]
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = glance
password = ii70mseq
[matchmaker_redis]
[matchmaker_ring]
[oslo_concurrency]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
[paste_deploy]
flavor = keystone
[store_type_location_strategy]
[task]
[taskflow_executor]
```

3. Populate the Image service database:

```
# su -s /bin/sh -c "glance-manage db_sync" glance
```

Finalize installation

1. Restart the Image service services:

```
# service glance-registry restart
# service glance-api restart
```

2. By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

```
# rm -f /var/lib/glance/glance.sqlite
```

Verify operation

Run on the **Controller Node**

1. In each client environment script, configure the Image service client to use API version 2.0:

```
# echo "export OS_IMAGE_API_VERSION=2" | tee -a admin-openrc.sh \
VHG-openrc.sh
```

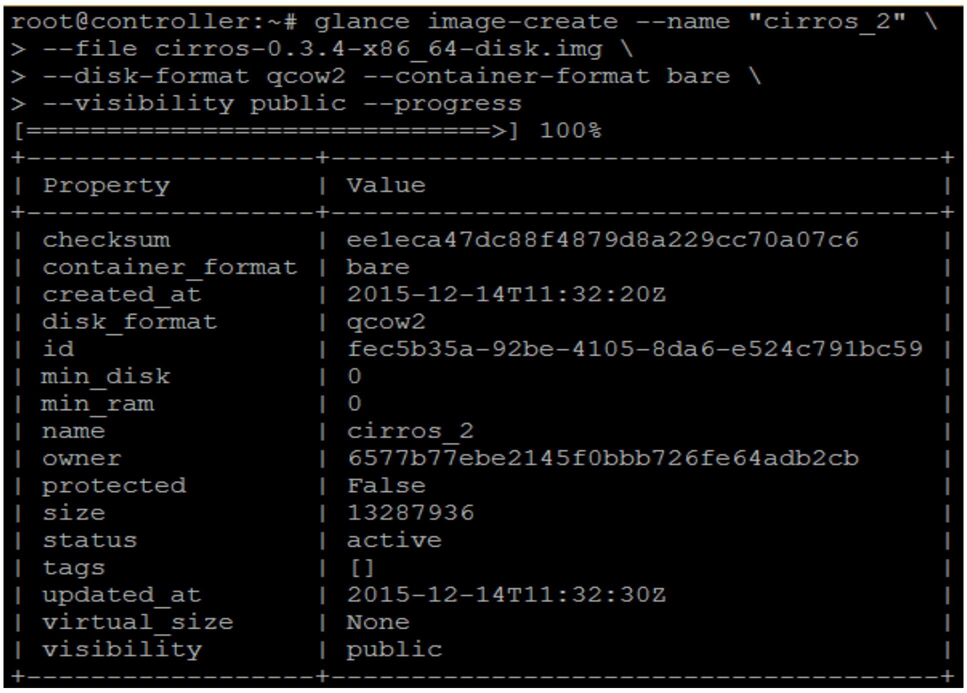
2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. Download the source image:

```
# wget http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img
```

4. Upload the image to the Image service using the qcow2 disk format, bare container format, and public visibility so all projects can access it:



```
root@controller:~# glance image-create --name "cirros_2" \
> --file cirros-0.3.4-x86_64-disk.img \
> --disk-format qcow2 --container-format bare \
> --visibility public --progress
[=============================>] 100%
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | ee1eca47dc88f4879d8a229cc70a07c6     |
| container_format | bare                                 |
| created_at       | 2015-12-14T11:32:20Z                 |
| disk_format      | qcow2                                |
| id               | fec5b35a-92be-4105-8da6-e524c791bc59 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros_2                             |
| owner            | 6577b77ebe2145f0bbb726fe64adb2cb     |
| protected        | False                                |
| size             | 13287936                             |
| status           | active                               |
| tags             | []                                   |
| updated_at       | 2015-12-14T11:32:30Z                 |
| virtual_size     | None                                 |
| visibility       | public                               |
+------------------+--------------------------------------+
```

Add the Compute Service

OpenStack Compute is a major part of an Infrastructure-as-a-Service (IaaS) system. The main modules are implemented in Python.

It interacts with OpenStack Identity for authentication, OpenStack Image service for disk and server images, and OpenStack dashboard for the user and administrative interface. Image access is limited by projects, and by users; quotas are limited per project. OpenStack Compute can scale horizontally on standard hardware, and download images to launch instances.

Install and configure **Controller Node**

1. Create nova database, service credentials, and API endpoints

   Create the database following the next steps

   - Connect as root to the Database server

```
# mysql -u root –p
```

   - Create the Database

```
CREATE DATABASE nova;
```

   - Grant proper access to the database

```
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' IDENTIFIED BY \
'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED BY \
'NOVA_DBPASS';
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. Create the service credentials

Create the nova user:

```
# openstack user create --domain default --password-prompt nova
```

Add the admin role to the nova user:

```
# openstack role add --project service --user nova admin
```

Create the nova service entity:

```
# openstack service create --name nova --description "OpenStack Compute" \
compute
```

4. Create the Image service API endpoints:

```
# openstack endpoint create --region RegionOne  image public \
http://$controller_pub_ip:8774/v2/%\(tenant_id\)s
# openstack endpoint create --region RegionOne image internal \
http://$controller_mgmt_ip:8774/v2/%\(tenant_id\)s
# openstack endpoint create --region RegionOne image admin \
http://$controller_mgmt_ip:8774/v2/%\(tenant_id\)s
```

Install and configure components

1. Install the packages:

```
# apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
nova-novncproxy nova-scheduler python-novaclient
```

2. Edit the /etc/nova/nova.conf file
3. Populate the Compute database:

```
# su -s /bin/sh -c "nova-manage db sync" nova
```

Finalize installation

Restart the Compute services:

```
# service nova-apinova-cert nova-consoleauth nova-scheduler \
nova-conductor nova-novncproxy restart
```

By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

```
# rm -f /var/lib/nova/nova.sqlite
```

Install and configure **Compute Nodes**

Install and configure components

1. Install the packages:

```
# apt-get install nova-compute sysfsutils
```

2. Edit the /etc/nova/nova.conf file

Finalize installation

1. Determine whether your compute node supports hardware acceleration for virtual machines:

```
# egrep -c '(vmx|svm)' /proc/cpuinfo
```

2. Restart the Compute service:

```
# service nova-compute restart
```

3. By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

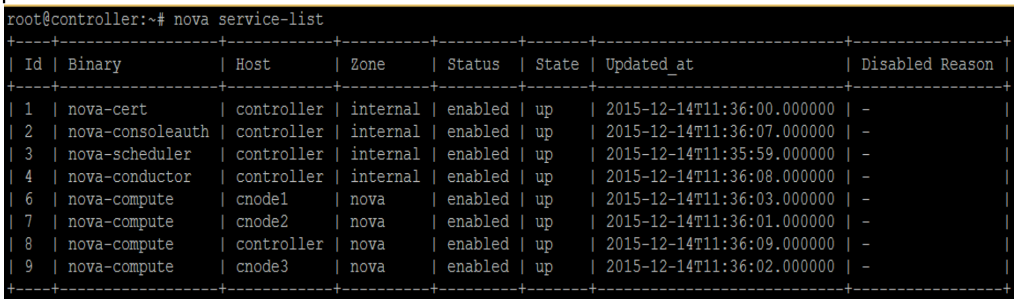
```
# rm -f /var/lib/nova/nova.sqlite
```

Verify operation

1. Source the admin credentials to gain access to admin-only CLI commands:

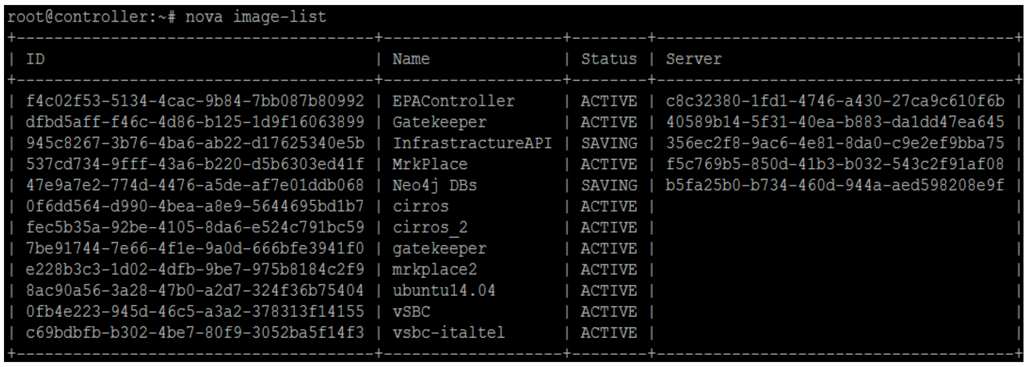
```
# source admin-openrc.sh
```

2. List service components to verify successful launch and registration of each process:



```
root@controller:~# nova service-list
+----+------------------+------------+----------+---------+-------+----------------------------+-----------------+
| Id | Binary           | Host       | Zone     | Status  | State | Updated_at                 | Disabled Reason |
+----+------------------+------------+----------+---------+-------+----------------------------+-----------------+
| 1  | nova-cert        | controller | internal | enabled | up    | 2015-12-14T11:36:00.000000 | -               |
| 2  | nova-consoleauth | controller | internal | enabled | up    | 2015-12-14T11:36:07.000000 | -               |
| 3  | nova-scheduler   | controller | internal | enabled | up    | 2015-12-14T11:35:59.000000 | -               |
| 4  | nova-conductor   | controller | internal | enabled | up    | 2015-12-14T11:36:08.000000 | -               |
| 6  | nova-compute     | cnode1     | nova     | enabled | up    | 2015-12-14T11:36:03.000000 | -               |
| 7  | nova-compute     | cnode2     | nova     | enabled | up    | 2015-12-14T11:36:01.000000 | -               |
| 8  | nova-compute     | controller | nova     | enabled | up    | 2015-12-14T11:36:09.000000 | -               |
| 9  | nova-compute     | cnode3     | nova     | enabled | up    | 2015-12-14T11:36:02.000000 | -               |
+----+------------------+------------+----------+---------+-------+----------------------------+-----------------+
```

3. List images in the Image service catalog to verify connectivity with the Image service:



```
root@controller:~# nova image-list
+--------------------------------------+-------------------+--------+--------------------------------------+
| ID                                   | Name              | Status | Server                               |
+--------------------------------------+-------------------+--------+--------------------------------------+
| f4c02f53-5134-4cac-9b84-7bb087b80992 | EPAController     | ACTIVE | c8c32380-1fd1-4746-a430-27ca9c610f6b |
| dfbd5aff-f46c-4d86-b125-1d9f16063899 | Gatekeeper        | ACTIVE | 40589b14-5f31-40ea-b883-da1dd47ea645 |
| 945c8267-3b76-4ba6-ab22-d17625340e5b | InfrastructureAPI | SAVING | 356ec2f8-9ac6-4e81-8da0-c9e2ef9bba75 |
| 537cd734-9fff-43a6-b220-d5b6303ed41f | MrkPlace          | ACTIVE | f5c769b5-850d-41b3-b032-543c2f91af08 |
| 47e9a7e2-774d-4476-a5de-af7e01ddb068 | Neo4j DBs         | SAVING | b5fa25b0-b734-460d-944a-aed598208e9f |
| 0f6dd564-d990-4bea-a8e9-5644695bd1b7 | cirros            | ACTIVE |                                      |
| fec5b35a-92be-4105-8da6-e524c791bc59 | cirros_2          | ACTIVE |                                      |
| 7be91744-7e66-4f1e-9a0d-666bfe3941f0 | gatekeeper        | ACTIVE |                                      |
| e228b3c3-1d02-4dfb-9be7-975b8184c2f9 | mrkplace2         | ACTIVE |                                      |
| 8ac90a56-3a28-47b0-a2d7-324f36b75404 | ubuntu14.04       | ACTIVE |                                      |
| 0fb4e223-945d-46c5-a3a2-378313f14155 | vSBC              | ACTIVE |                                      |
| c69bdbfb-b302-4be7-80f9-3052ba5f14f3 | vsbc-italtel      | ACTIVE |                                      |
+--------------------------------------+-------------------+--------+--------------------------------------+
```

Add the Networking service

OpenStack Networking (neutron) allows you to create and attach interface devices managed by other OpenStack services to networks. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

Install and configure **Controller Node**

1. Create nova database, service credentials, and API endpoints

   Create the database following the next steps

   - Connect as root to the Database server

```
# mysql -u root –p
```

   - Create the Database

```
CREATE DATABASE neutron;
```

   - Grant proper access to the database

```
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost' IDENTIFIED BY
'NEUTRON_DBPASS';
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' IDENTIFIED BY \
'NEUTRON_DBPASS';
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. Create the service credentials

   Create the neutron user:

```
# openstack user create --domain default --password-prompt neutron
```

   Add the admin role to the neutron user:

```
# openstack role add --project service --user neutron admin
```

   Create the neutron service entity:

```
# openstack service create --name neutron --description "OpenStack \
Networking" network
```

4. Create the Networking service API endpoints:

```
# openstack endpoint create --region RegionOne  image public \
http://$controller_pub_ip:9696
# openstack endpoint create --region RegionOne image internal \
http://$controller_mgmt_ip:9696
# openstack endpoint create --region RegionOne image admin \
http://$controller_mgmt_ip:9696
```

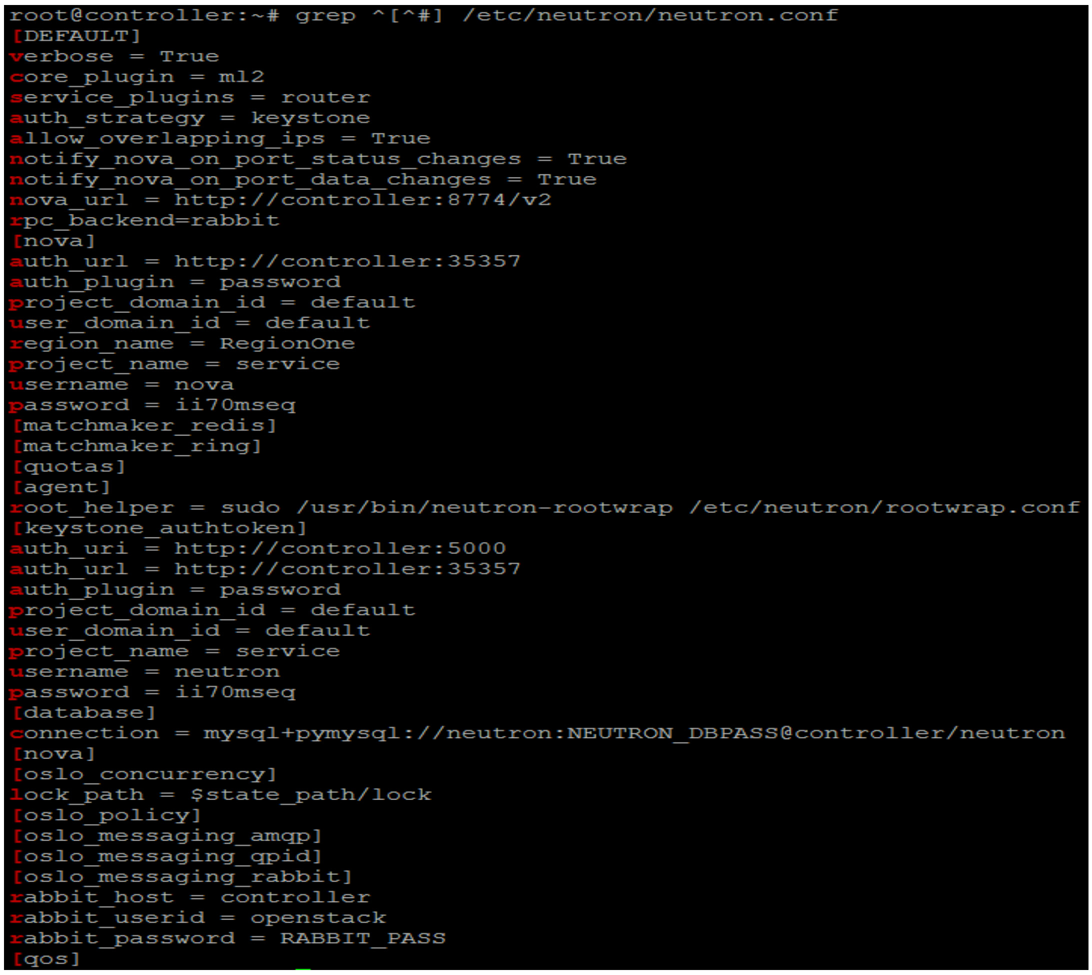
<u>Configure networking options</u>

Install the components:

```
# apt-get install neutron-server neutron-plugin-ml2 neutron-l3-agent \
  neutron-plugin-openvswitch-agent neutron-dhcp-agent \
  neutron-metadata-agent python-neutronclient
```

<u>Configure the server component</u>

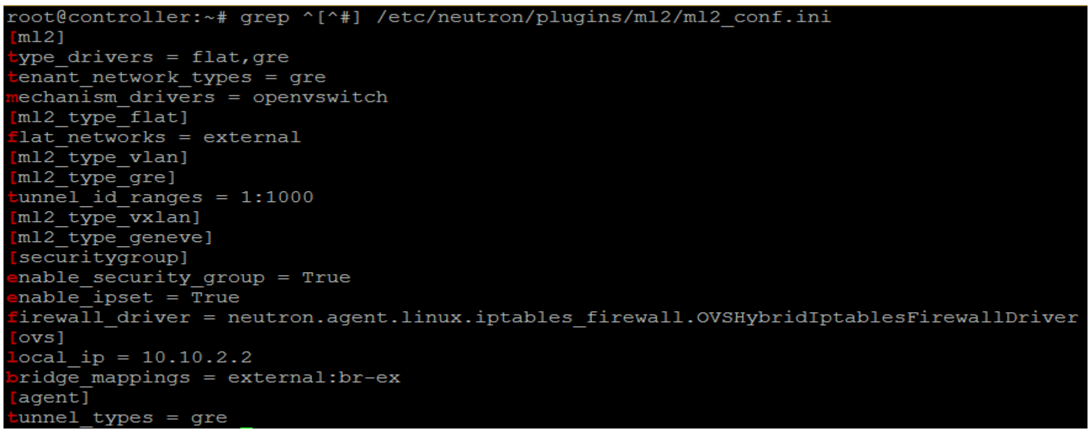
1. Edit the /etc/neutron/neutron.conf



```
root@controller:~# grep ^[^#] /etc/neutron/neutron.conf
[DEFAULT]
verbose = True
core_plugin = ml2
service_plugins = router
auth_strategy = keystone
allow_overlapping_ips = True
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://controller:8774/v2
rpc_backend=rabbit
[nova]
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
region_name = RegionOne
project_name = service
username = nova
password = ii70mseq
[matchmaker_redis]
[matchmaker_ring]
[quotas]
[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = ii70mseq
[database]
connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/neutron
[nova]
[oslo_concurrency]
lock_path = $state_path/lock
[oslo_policy]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[qos]
```

Configure the Modular Layer 2 (ML2) plug-in

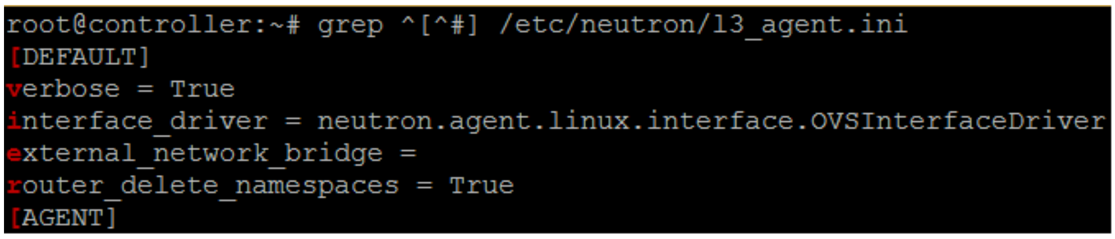
1. Edit the /etc/neutron/plugins/ml2/ml2_conf.ini



```
root@controller:~# grep ^[^#] /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
[ml2_type_flat]
flat_networks = external
[ml2_type_vlan]
[ml2_type_gre]
tunnel_id_ranges = 1:1000
[ml2_type_vxlan]
[ml2_type_geneve]
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
[ovs]
local_ip = 10.10.2.2
bridge_mappings = external:br-ex
[agent]
tunnel_types = gre
```

Configure the layer-3 agent

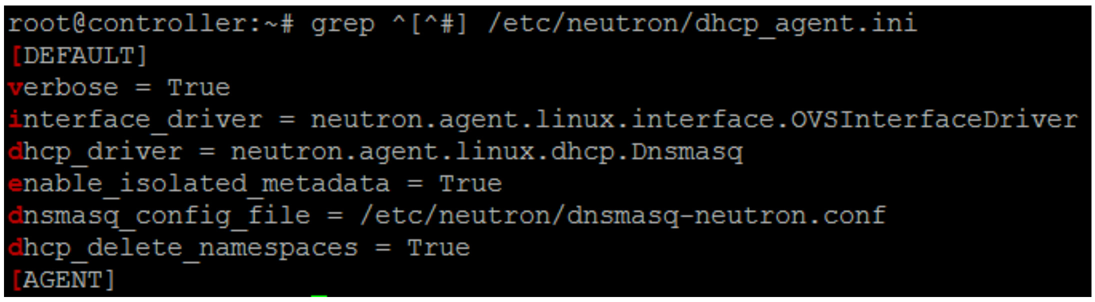
1. Edit the /etc/neutron/l3_agent.ini file



```
root@controller:~# grep ^[^#] /etc/neutron/l3_agent.ini
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
external_network_bridge =
router_delete_namespaces = True
[AGENT]
```

Configure the DHCP agent

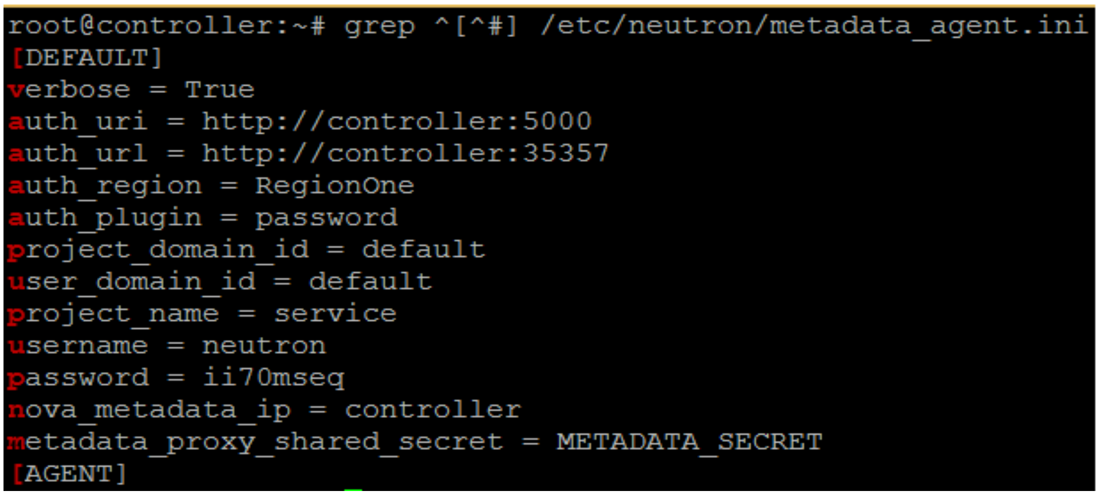
1. Edit the /etc/neutron/dhcp_agent.ini file



```
root@controller:~# grep ^[^#] /etc/neutron/dhcp_agent.ini
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = True
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
dhcp_delete_namespaces = True
[AGENT]
```

Configure the metadata agent

1. Edit the /etc/neutron/metadata_agent.ini file



```
root@controller:~# grep ^[^#] /etc/neutron/metadata_agent.ini
[DEFAULT]
verbose = True
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_region = RegionOne
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = ii70mseq
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_SECRET
[AGENT]
```

Configure Compute to use Networking

1. Edit the /etc/nova/nova.conf file

Finalize installation

1. Populate the database:

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf
 --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron
```

Restart the Compute API service:

```
# service nova-api restart
```

Configure the Open vSwitch (OVS) service

The OVS service provides the underlying virtual networking framework for instances. The integration bridge br-int handles internal instance network traffic within OVS. The external bridge br-ex handles external instance network traffic within OVS. The

external bridge requires a port on the physical external network interface to provide instances with external network access. In essence, this port connects the virtual and physical external networks in your environment.
Prepare following steps in **Controller Node**

1. Restart the OVS service:

```
# service openvswitch-switch restart
```

2. Add the external bridge:

```
# ovs-vsctl add-br br-ex
```

3. Add a port to the external bridge that connects to the physical external network interface:

```
# ovs-vsctl add-port br-ex $public_interface
```

4. Restart the Networking services:

```
# service neutron-server restart
# service neutron-plugin-openvswitch-agent restart
# service neutron-l3-agent-restart
# service neutron-dhcp-agent restart
# service neutron-metadata-agent restart
```

5. By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

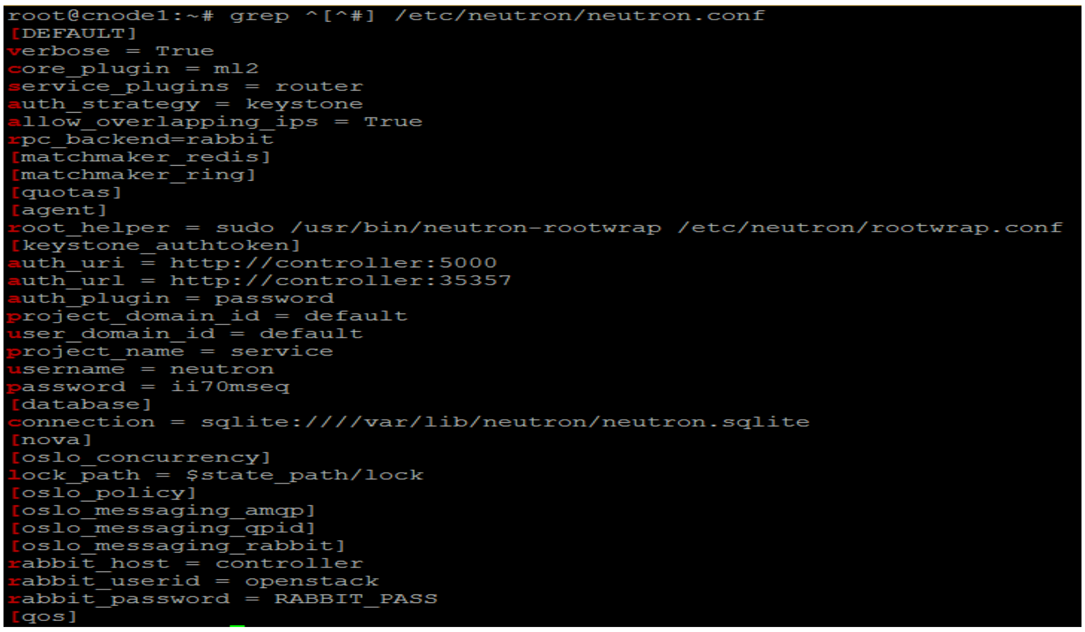
```
# rm -f /var/lib/neutron/neutron.sqlite
```

<u>Install and configure **Compute Nodes**</u>
Prepare these steps in **Controller Node**

1. Install the components

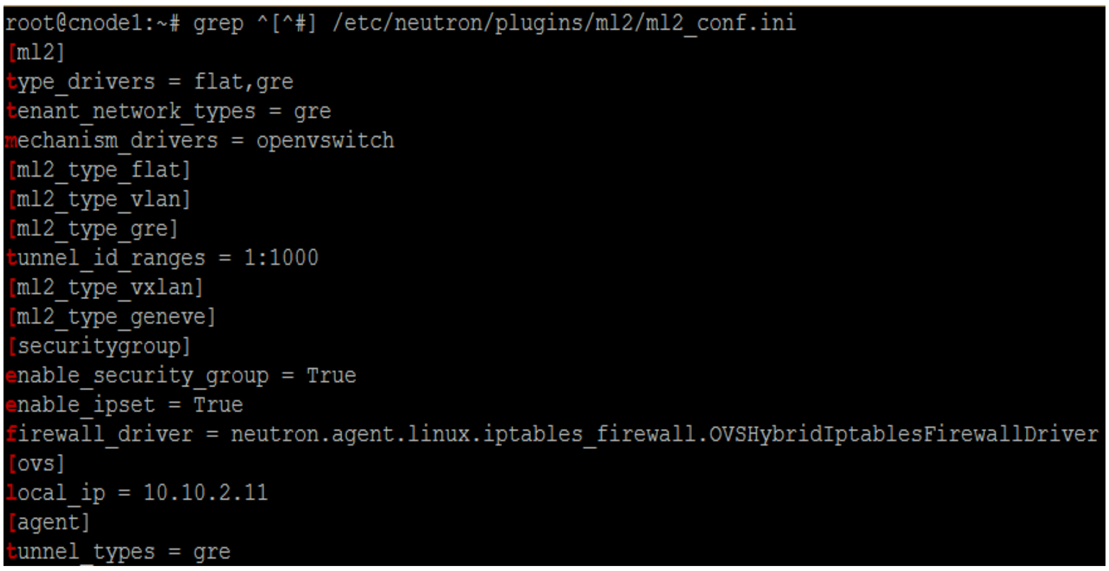
```
# apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent
```

Edit the /etc/neutron/neutron.conf file



```
root@cnode1:~# grep ^[^#] /etc/neutron/neutron.conf
[DEFAULT]
verbose = True
core_plugin = ml2
service_plugins = router
auth_strategy = keystone
allow_overlapping_ips = True
rpc_backend=rabbit
[matchmaker_redis]
[matchmaker_ring]
[quotas]
[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = ii70mseq
[database]
connection = sqlite:////var/lib/neutron/neutron.sqlite
[nova]
[oslo_concurrency]
lock_path = $state_path/lock
[oslo_policy]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[qos]
```

2. Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file



```
root@cnode1:~# grep ^[^#] /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
[ml2_type_flat]
[ml2_type_vlan]
[ml2_type_gre]
tunnel_id_ranges = 1:1000
[ml2_type_vxlan]
[ml2_type_geneve]
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
[ovs]
local_ip = 10.10.2.11
[agent]
tunnel_types = gre
```

Configure the Open vSwitch (OVS) service

1. Restart the OVS service:

```
# service openvswitch-switch restart
```

Configure Compute to use Networking

1. Edit /etc/nova/nova.conf file

Finalize the installation

1. Restart the Compute service:

```
# service nova-compute restart
```

2. Restart the Open vSwitch (OVS) agent:

```
# service neutron-plugin-openvswitch-agent restart
```

Verify operation

Run on **Controller Node**

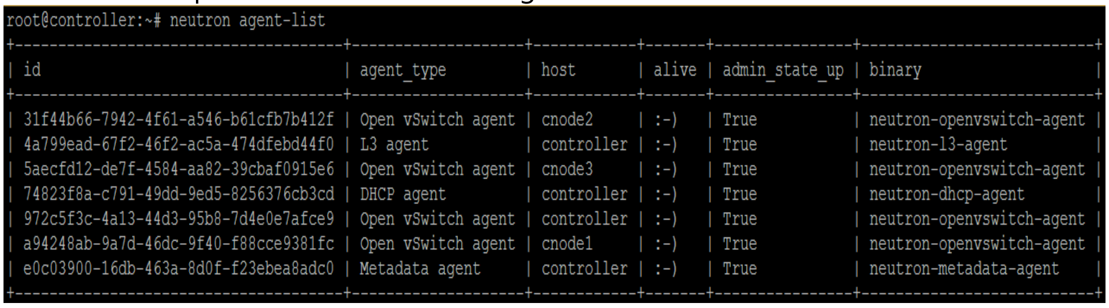
1. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

2. List agents to verify successful launch of the neutron agents:

```
# neutron agent-list
```

The output should be as following:



```
root@controller:~# neutron agent-list
+--------------------------------------+--------------------+------------+-------+----------------+---------------------------+
| id                                   | agent_type         | host       | alive | admin_state_up | binary                    |
+--------------------------------------+--------------------+------------+-------+----------------+---------------------------+
| 31f44b66-7942-4f61-a546-b61cfb7b412f | Open vSwitch agent | cnode2     | :-)   | True           | neutron-openvswitch-agent |
| 4a799ead-67f2-46f2-ac5a-474dfebd44f0 | L3 agent           | controller | :-)   | True           | neutron-l3-agent          |
| 5aecfd12-de7f-4584-aa82-39cbaf0915e6 | Open vSwitch agent | cnode3     | :-)   | True           | neutron-openvswitch-agent |
| 74823f8a-c791-49dd-9ed5-8256376cb3cd | DHCP agent         | controller | :-)   | True           | neutron-dhcp-agent        |
| 972c5f3c-4a13-44d3-95b8-7d4e0e7afce9 | Open vSwitch agent | controller | :-)   | True           | neutron-openvswitch-agent |
| a94248ab-9a7d-46dc-9f40-f88cce9381fc | Open vSwitch agent | cnode1     | :-)   | True           | neutron-openvswitch-agent |
| e0c03900-16db-463a-8d0f-f23ebea8adc0 | Metadata agent     | controller | :-)   | True           | neutron-metadata-agent    |
+--------------------------------------+--------------------+------------+-------+----------------+---------------------------+
```

Create Public network

1. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

2. Create Public network

```
# neutron net-create public --shared --provider:physical_network external \
--provider:network_type flat
```

3. Create Public subnet

```
# neutron subnet-create public 10.10.1.0/24 --name public --allocation-pool \
start=10.10.1.50,end=10.10.1.200 --dns-nameserver 8.8.8.8 \
--gateway 10.10.1.1
```

Create the private project network (one or more per tenant)

1. On the controller node, source the tenant's (for example VHG) credentials to gain access to user-only CLI commands:

```
# source VHG-openrc.sh
```

2. Create the network:

```
# neutron net-create private
```

3. Create a subnet on the network:

```
# neutron subnet-create private 172.16.1.0/24 --name private  \
--dns-nameserver 8.8.8.8 --gateway 172.16.1.1
```

Create a router

1. With admin credentials, add the *router:external* option to the public provider network:

```
# source admin-openrc.sh
# neutron net-update public --router:external
```

2. Load VHG tenant's credentials and create the router:

```
# source VHG-openrc.sh
# neutron router-create VHG-router
```

3. Add the private network subnet as an interface on the router:

```
# neutron router-interface-add router private
```

4. Set a gateway on the public network on the router:

```
# neutron router-gateway-set router public
```

Add the Openstack dashboard

The OpenStack Dashboard, also known as horizon is a web interface that enables cloud administrators and users to manage various OpenStack resources and services.

Install and configure components

1. Install the packages:

```
# apt-get install openstack-dashboard
```

2. Edit the /etc/openstack-dashboard/local_settings.py file and configure the following:

```
•     OPENSTACK_HOST = "controller"
•     ALLOWED_HOSTS = ['*', ]
•     CACHES = {
       'default': {
       'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
                  }
          }
•     OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
•     TIME_ZONE = "TIME_ZONE"
```

3. Reload the web server configuration:

```
# service apache2 reload
```

Add the Orchestration service

The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups and users. It also provides advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. This enables OpenStack core projects to receive a larger user base.

The service enables deployers to integrate with the Orchestration service directly or through custom plug-ins.

Prepare following steps only in **Controller Node**

Install and configure

1. Create glance database, service credentials, and API endpoints

   Create the database following the next steps

   • Connect as root to the Database server

```
# mysql -u root –p
```

   • Create the Database

```
CREATE DATABASE heat;
```

   • Grant proper access to the database

```
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' IDENTIFIED BY \
'HEAT_DBPASS';
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' IDENTIFIED BY \
'HEAT_DBPASS';
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. Create the service credentials

   Create the heat user:

```
# openstack user create --domain default --password-prompt heat
```

   Add the admin role to the heat user:

```
# openstack role add --project service --user heat admin
```

   Create the heat and heat-cfn service entities:

```
# openstack service create --name heat --description "Orchestration"  \
orchestration
# openstackservice create --name heat-cfn--description "Orchestration" \
cloudformation
```

4. Create the Orchestration service API endpoints:

```
# openstack endpoint create --region RegionOne orchestration public \
http://controller_public_ip:8004/v1/%\(tenant_id\)s
# openstack endpoint create --region RegionOne orchestration internal \
http://controller_mgmt_ip:8004/v1/%\(tenant_id\)s
# openstack endpoint create --region RegionOne orchestration admin \
http://controller_mgmt_ip:8004/v1/%\(tenant_id\)s
# openstack endpoint create --region RegionOnecloudformation public \
http://controller_public_ip:8000/v1
# openstack endpoint create --region RegionOnecloudformation internal \
http://controller_mgmt_ip:8000/v1
# openstack endpoint create --region RegionOnecloudformation admin \
http://controller_mgmt_ip:8000/v1
```

5. Orchestration requires additional information in the Identity service to manage stacks. To add this information, complete these steps:
   - Create the heat domain that contains projects and users for stacks:

```
# openstack domain create --description "Stack projects and users" heat
```

   - Create the heat_domain_admin user to manage projects and users in the heat domain:

```
# openstack user create --domain heat --password-prompt \
heat_domain_admin
```

   - Add the admin role to the heat_domain_admin user in the heat domain to enable administrative stack management privileges by the heat_domain_admin user:

```
# openstack role add --domain heat --user heat_domain_admin admin
```

   - Create the heat_stack_owner role:

```
# openstack role create heat_stack_owner
```

   - Add the heat_stack_owner role to the demo project and user to enable stack management by the demo user:

```
# openstack role add --project demo --user demo heat_stack_owner
```

   - Create the heat_stack_user role:

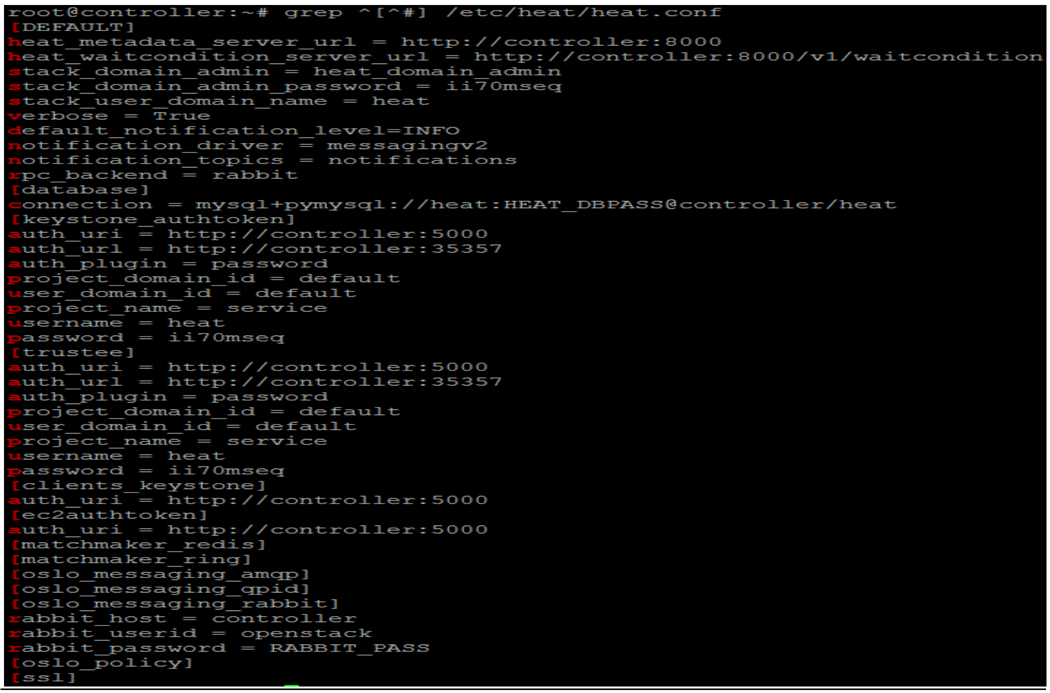
```
# openstack role create heat_stack_user
```

Install and configure components

1. Install the packages:

```
# apt-get install heat-api heat-api-cfn heat-engine python-heatclient
```

2. Edit the /etc/heat/heat.conf file



```
root@controller:~# grep ^[^#] /etc/heat/heat.conf
[DEFAULT]
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/waitcondition
stack_domain_admin = heat_domain_admin
stack_domain_admin_password = ii70mseq
stack_user_domain_name = heat
verbose = True
default_notification_level=INFO
notification_driver = messagingv2
notification_topics = notifications
rpc_backend = rabbit
[database]
connection = mysql+pymysql://heat:HEAT_DBPASS@controller/heat
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = heat
password = ii70mseq
[trustee]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = heat
password = ii70mseq
[clients_keystone]
auth_uri = http://controller:5000
[ec2authtoken]
auth_uri = http://controller:5000
[matchmaker_redis]
[matchmaker_ring]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
[ssl]
```

3. Populate the Orchestration database:

```
# su -s /bin/sh -c "heat-manage db_sync" heat
```

4. Restart the Orchestration services:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

5. By default, the Ubuntu packages create an SQLite database. We remove the SQLite database file:

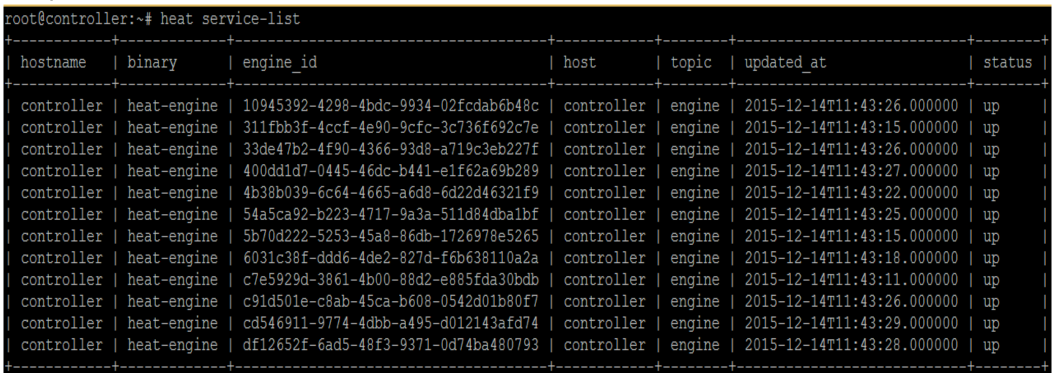
```
# rm -f /var/lib/heat/heat.sqlite
```

Verify operation

1. Source the admin tenant credentials:

```
# source admin-openrc.sh
```

2. List service components to verify successful launch and registration of each process:



```
root@controller:~# heat service-list
+------------+-------------+--------------------------------------+------------+--------+----------------------------+--------+
| hostname   | binary      | engine_id                            | host       | topic  | updated_at                 | status |
+------------+-------------+--------------------------------------+------------+--------+----------------------------+--------+
| controller | heat-engine | 10945392-4298-4bdc-9934-02fcdab6b48c | controller | engine | 2015-12-14T11:43:26.000000 | up     |
| controller | heat-engine | 311fbb3f-4ccf-4e90-9cfc-3c736f692c7e | controller | engine | 2015-12-14T11:43:15.000000 | up     |
| controller | heat-engine | 33de47b2-4f90-4366-93d8-a719c3eb227f | controller | engine | 2015-12-14T11:43:26.000000 | up     |
| controller | heat-engine | 400dd1d7-0445-46dc-b441-e1f62a69b289 | controller | engine | 2015-12-14T11:43:27.000000 | up     |
| controller | heat-engine | 4b38b039-6c64-4665-a6d8-6d22d46321f9 | controller | engine | 2015-12-14T11:43:22.000000 | up     |
| controller | heat-engine | 54a5ca92-b223-4717-9a3a-511d84dba1bf | controller | engine | 2015-12-14T11:43:25.000000 | up     |
| controller | heat-engine | 5b70d222-5253-45a8-86db-1726978e5265 | controller | engine | 2015-12-14T11:43:15.000000 | up     |
| controller | heat-engine | 6031c38f-ddd6-4de2-827d-f6b638110a2a | controller | engine | 2015-12-14T11:43:18.000000 | up     |
| controller | heat-engine | c7e5929d-3861-4b00-88d2-e885fda30bdb | controller | engine | 2015-12-14T11:43:11.000000 | up     |
| controller | heat-engine | c91d501e-c8ab-45ca-b608-0542d01b80f7 | controller | engine | 2015-12-14T11:43:26.000000 | up     |
| controller | heat-engine | cd546911-9774-4dbb-a495-d012143afd74 | controller | engine | 2015-12-14T11:43:29.000000 | up     |
| controller | heat-engine | df12652f-6ad5-48f3-9371-0d74ba480793 | controller | engine | 2015-12-14T11:43:28.000000 | up     |
+------------+-------------+--------------------------------------+------------+--------+----------------------------+--------+
```

Add the Telemetry service

- The Telemetry service performs the following functions:
- Efficiently polls metering data related to OpenStack services.

- Collects event and metering data by monitoring notifications sent from services.
- Publishes collected data to various targets including data stores and message queues.
- Creates alarms when collected data breaks defined rules.

Install and configure components

1. Create the ceilometer database:

```
# mongo --host controller --eval '
  db = db.getSiblingDB("ceilometer");
  db.addUser({user: "ceilometer",
  pwd: "CEILOMETER_DBPASS",
   roles: [ "readWrite", "dbAdmin" ]})'
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

3. To create the service credentials, complete these steps:
   Create the ceilometer user:

```
# openstack user create --domain default --password-prompt ceilometer
```

   Add the admin role to the ceilometer user.

```
# openstack role add --project service --user ceilometer admin
```

   Create the ceilometer service entity:

```
# openstack service create --name ceilometer--description "Telemetry" \
 metering
```

4. Create the Telemetry service API endpoints:

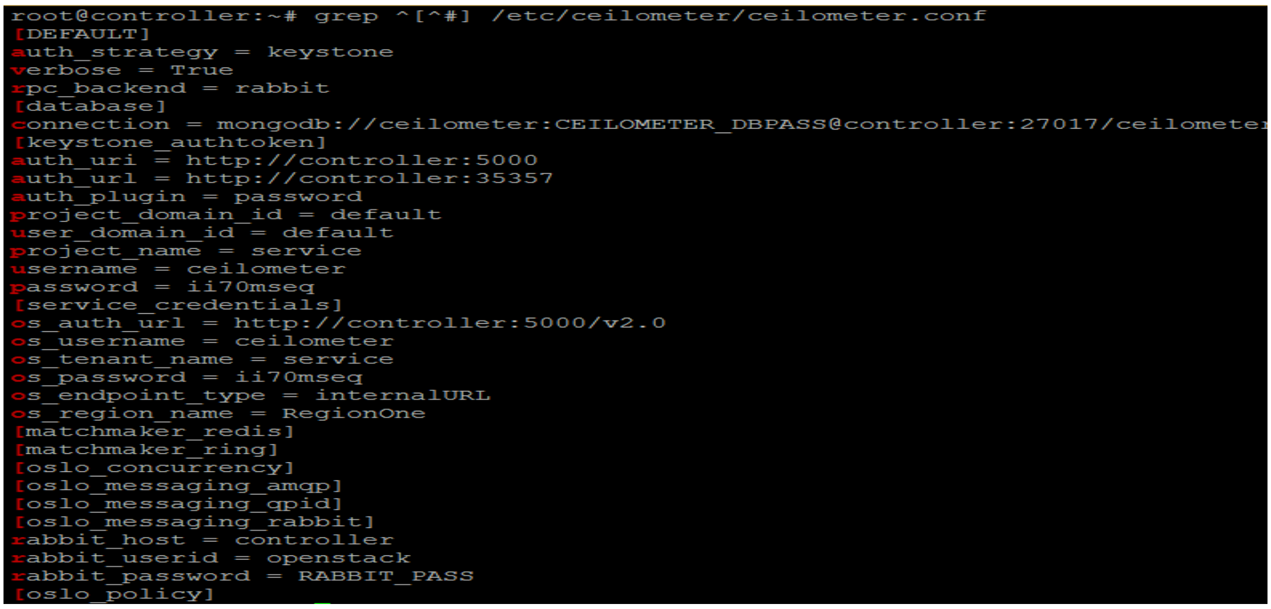
```
# openstack endpoint create --region RegionOnemetering   public \
 http://controller_public_ip:8777
# openstack endpoint create --region RegionOnemetering internal \
 http://controller_mgmt_ip:8777
# openstack endpoint create --region RegionOnemetering admin \
 http://controller_mgmt_ip:8777
```

Install and configure components

1. Install the packages:

```
# apt-get install ceilometer-api ceilometer-collector ceilometer-agent-central \
 ceilometer-agent-notification ceilometer-alarm-evaluator \
 ceilometer-alarm-notifier python-ceilometerclient
```

2. Edit the /etc/ceilometer/ceilometer.conf file



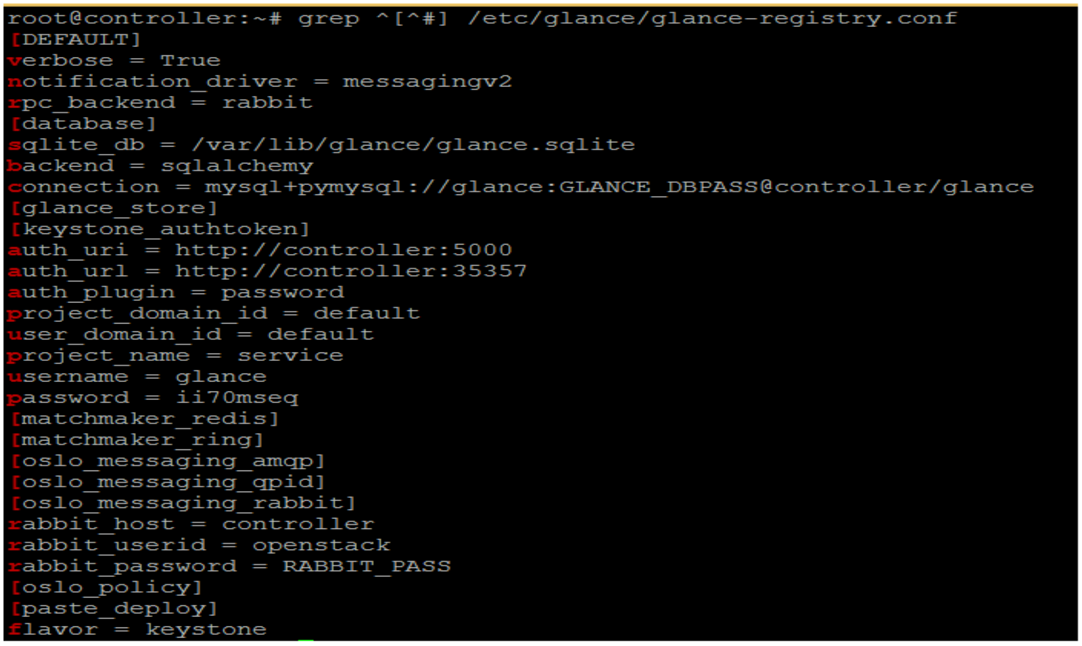
```
root@controller:~# grep ^[^#] /etc/ceilometer/ceilometer.conf
[DEFAULT]
auth_strategy = keystone
verbose = True
rpc_backend = rabbit
[database]
connection = mongodb://ceilometer:CEILOMETER_DBPASS@controller:27017/ceilomete
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = ceilometer
password = ii70mseq
[service_credentials]
os_auth_url = http://controller:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ii70mseq
os_endpoint_type = internalURL
os_region_name = RegionOne
[matchmaker_redis]
[matchmaker_ring]
[oslo_concurrency]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
```

3. Restart the Telemetry services:

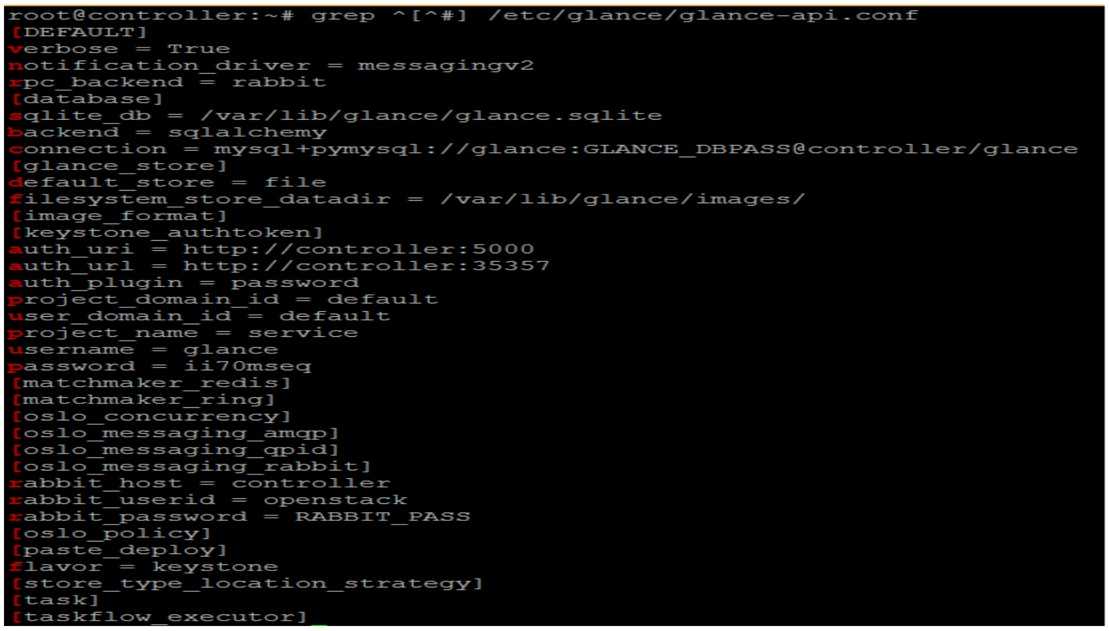
```
# service ceilometer-agent-central restart
# service ceilometer-agent-notification restart
# service ceilometer-api restart
# service ceilometer-collector restart
# service ceilometer-alarm-evaluator restart
# service ceilometer-alarm-notifier restart
```

<u>Configure the Image service to use Telemetry</u>

1. <u>Edit the /etc/glance/glance-api.conf and /etc/glance/glance-registry.conf files</u>



```
root@controller:~# grep ^[^#] /etc/glance/glance-registry.conf
[DEFAULT]
verbose = True
notification_driver = messagingv2
rpc_backend = rabbit
[database]
sqlite_db = /var/lib/glance/glance.sqlite
backend = sqlalchemy
connection = mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
[glance_store]
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = glance
password = ii70mseq
[matchmaker_redis]
[matchmaker_ring]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
[paste_deploy]
flavor = keystone
```



```
root@controller:~# grep ^[^#] /etc/glance/glance-api.conf
[DEFAULT]
verbose = True
notification_driver = messagingv2
rpc_backend = rabbit
[database]
sqlite_db = /var/lib/glance/glance.sqlite
backend = sqlalchemy
connection = mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
[glance_store]
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
[image_format]
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = glance
password = ii70mseq
[matchmaker_redis]
[matchmaker_ring]
[oslo_concurrency]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
[paste_deploy]
flavor = keystone
[store_type_location_strategy]
[task]
[taskflow_executor]
```

2. Restart the Image service:

```
# service glance-registry restart
# service glance-api restart
```

Enable Compute service meters (**Compute Nodes**)

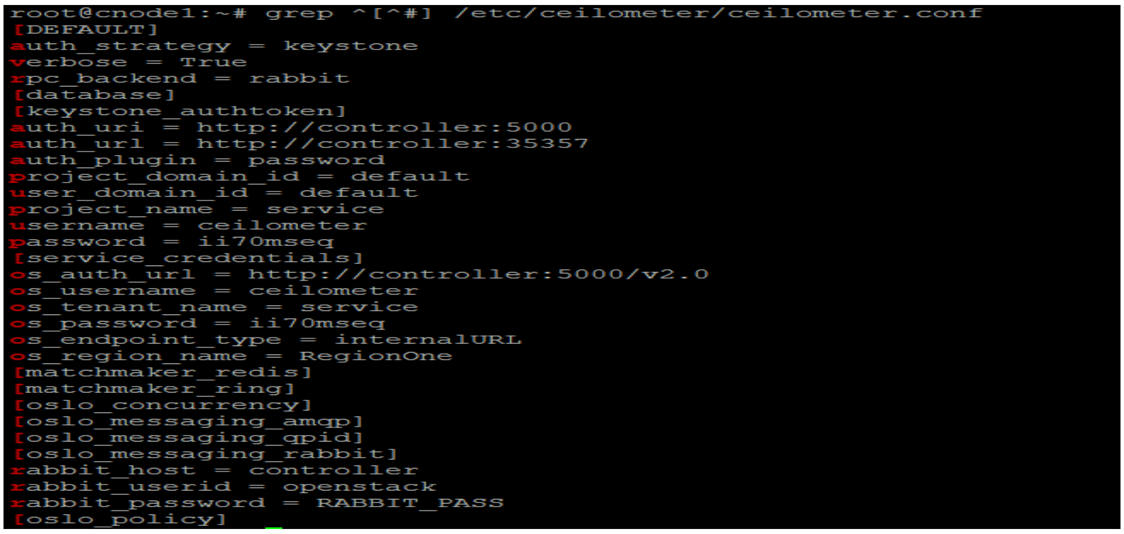
Telemetry uses a combination of notifications and an agent to collect Compute meters. Perform these steps on each compute node.

Install and configure components

1. Install the packages:

```
# apt-get install ceilometer-agent-compute
```

2. Edit the /etc/ceilometer/ceilometer.conf file



```
root@cnode1:~# grep ^[^#] /etc/ceilometer/ceilometer.conf
[DEFAULT]
auth_strategy = keystone
verbose = True
rpc_backend = rabbit
[database]
[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = ceilometer
password = ii70mseq
[service_credentials]
os_auth_url = http://controller:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ii70mseq
os_endpoint_type = internalURL
os_region_name = RegionOne
[matchmaker_redis]
[matchmaker_ring]
[oslo_concurrency]
[oslo_messaging_amqp]
[oslo_messaging_qpid]
[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
[oslo_policy]
```

Configure Compute to use Telemetry

Edit the /etc/nova/nova.conf file and add the following lines in the [Default] section:

```
[DEFAULT]
```

```
...
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = messagingv2
```

Finalize Installation

1. Restart the agent:

```
# service ceilometer-agent-compute restart
```

2. Restart the Compute service:

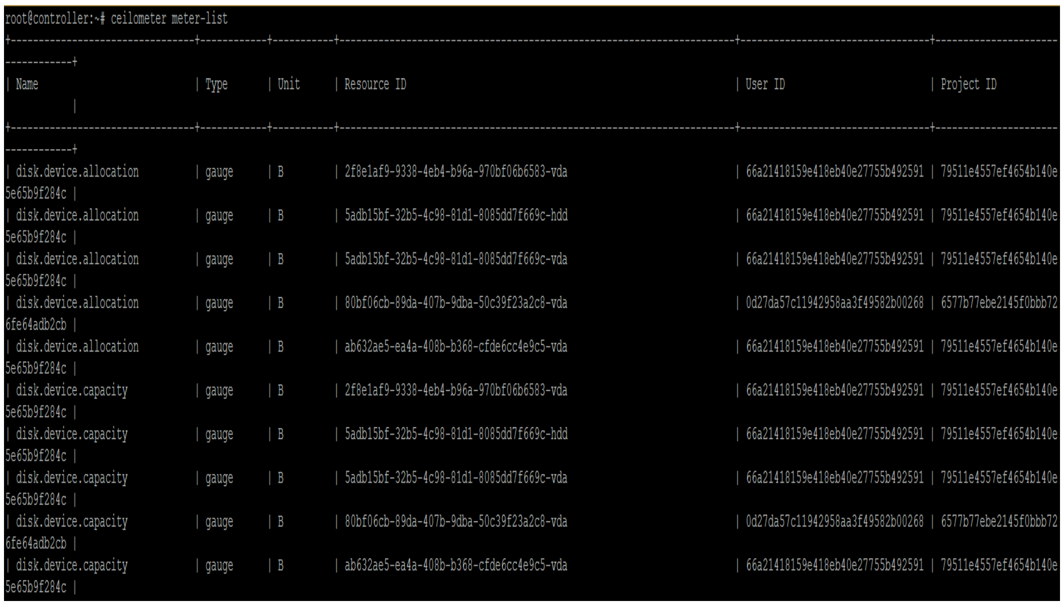
```
# service nova-compute restart
```

Verify operation

1. Source the admin credentials to gain access to admin-only CLI commands:

```
# source admin-openrc.sh
```

2. List available meters:



```
root@controller:~# ceilometer meter-list
+--------------------------------+------------+-----------+-------------------------------------------------------------------------+----------------------------------+----------------------
------------+
| Name                           | Type       | Unit      | Resource ID                                                             | User ID                          | Project ID
           |
+--------------------------------+------------+-----------+-------------------------------------------------------------------------+----------------------------------+----------------------
------------+
| disk.device.allocation         | gauge      | B         | 2f8e1af9-9338-4eb4-b96a-970bf06b6583-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.allocation         | gauge      | B         | 5adb15bf-32b5-4c98-81d1-8085dd7f669c-hdd                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.allocation         | gauge      | B         | 5adb15bf-32b5-4c98-81d1-8085dd7f669c-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.allocation         | gauge      | B         | 80bf06cb-89da-407b-9dba-50c39f23a2c8-vda                                | 0d27da57c11942958aa3f49582b00268 | 6577b77ebe2145f0bbb72
6fe64adb2cb |
| disk.device.allocation         | gauge      | B         | ab632ae5-ea4a-408b-b368-cfde6cc4e9c5-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.capacity           | gauge      | B         | 2f8e1af9-9338-4eb4-b96a-970bf06b6583-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.capacity           | gauge      | B         | 5adb15bf-32b5-4c98-81d1-8085dd7f669c-hdd                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.capacity           | gauge      | B         | 5adb15bf-32b5-4c98-81d1-8085dd7f669c-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
| disk.device.capacity           | gauge      | B         | 80bf06cb-89da-407b-9dba-50c39f23a2c8-vda                                | 0d27da57c11942958aa3f49582b00268 | 6577b77ebe2145f0bbb72
6fe64adb2cb |
| disk.device.capacity           | gauge      | B         | ab632ae5-ea4a-408b-b368-cfde6cc4e9c5-vda                                | 66a21418159e418eb40e27755b492591 | 79511e4557ef4654b140e
5e65b9f284c |
```

6.2. Annex B – OpenStack and OpenDaylight integration through ML2 plugin

This annex describes in detail the steps needed for configuring OpenStack Juno with Neutron ML2 networking to work with OpenDaylight Lithium and GRE Tunnels. Also it is important to know that one OpenDaylight manages only one OpenStack deployment.

Prerequisites

You must have a working OpenStack Juno deployment in Ubuntu 14.04 (LTS). To install it use the instructions provided in Annex A.

The networks required are:

- Management network 10.0.0.0/24
- Tunnel Network 10.0.1.0/24
- External Network 203.0.113.0/24

The OpenStack nodes required for this guide are:

- Controller node: Management Network, (External Network if you want public access to the controller)
- Network node: Management Network, External Network
- Compute node 1: Management Network, Tunnel Network
- Compute node 2: Management Network, Tunnel Network

Additionally, you must have OpenDaylight Helium SR2 installed in the Management Network. OpenDaylight must be installed in a different machine from your OpenStack nodes.

```
apt-get install openjdk-7-jdk
wget https://nexus.opendaylight.org/content/groups/public/org/open
daylight/integration/distribution-karaf/0.2.2-Helium-
SR2/distribution-karaf-0.2.2-Helium-SR2.zip
unzip distribution-karaf-0.2.2-Helium-SR2.zip
cd distribution-karaf-0.2.2-Helium-SR2
```

We want OpenDaylight to communicate with OpenFlow 1.3.

Edit etc/custom.properties and uncomment line ovsdb.of.version=1.3

```
./bin/start
```

Wait some seconds.

```
./bin/client
```

Now you are connected to OpenDaylight's console. Install all the required features:

```
feature:install odl-base-all odl-aaa-authn odl-restconf odl-nsf-all
odl-adsal-northbound odl-mdsal-apidocs odl-ovsdb-openstack odl-
ovsdb-northbound odl-dlux-core
```

Wait for the feature installation to finish.

Verify

```
curl -u:admin:admin:http://<OPENDAYLIGHT MANAGEMENT
IP>:8080/controller/nb/v2/neutron/networks
```

If everything is working fine, a list of networks will be returned and it will be empty.

Monitor

If you want to monitor OpenDaylight there are 2 log files.

```
tail -f data/log/karaf.log
tail -f logs/web_access_log_2015-11.txt
```

Controller Node: Erase all instances, networks, routers and ports

You must delete all existing instances, networks, routers and ports from all tenants. Default installation has admin and demo.

If you want do it from Horizon dashboards or use the following commands.

```
source admin-openrc
nova list
nova delete <INSTANCE ID>
neutron port-list
neutron port-delete <PORT ID>
neutron router-list
neutron router-gateway-clear <ROUTER ID>
neutron router-delete <ROUTER ID>
neutron net-list
neutron net-delete <NEWORK ID>
```

Do the same with demo-openrc.

If some ports cannot be deleted do the following:

```
mysql -uroot -p
use neutron;
delete from ports;
exit
```

Verify that everything is empty.

```
nova list
neutron port-list
neutron router-list
neutron net-list
```

Stop the neutron-server service for the duration of the configuration.

```
service neutron-server stop
```

A message saying that the neutron-server is stopped should appear. If not press it again to make sure it is stopped.

Network/Compute Nodes: Configure OpenvSwitches

The neutron plugin in every node must be removed (or stopped and disabled) because only OpenDaylight will be controlling openvswitches.

```
apt-get purge neutron-plugin-openvswitch-agent
service openvswitch-switch stop
rm -rf /var/log/openvswitch/*
rm -rf /etc/openvswitch/conf.db
service openvswitch-switch start
ovs-vsctl show
```

The last command must return an empty openvswitch. You should see only <OPENVSWITCH ID> and version.

```
ovs-vsctl set Open_vSwitch <OPENVSWITCH
ID> other_config={'local_ip'='<TUNNEL INTERFACE IP>'}
```

Nothing will appear if this command is entered correctly. To verify the configuration, you can use:

```
ovs-vsctl list Open vSwitch
```

ONLY NETWORK NODE SECTION START

Create the bridge br-ex that is needed for the external network for OpenStack.

```
ovs-vsctl add-br br-ex
ovs-vsctl add-port br-ex <INTERFACE NAME OF EXTERNAL NETWORK>
```

ONLY NETWORK NODE SECTION END

Connect every openvswitch with the OpenDaylight controller.

```
ovs-vsctl set-manager tcp:<OPENDAYLIGHT MANAGEMENT IP>:6640
```

If everything went ok you can see 4 switches in OpenDaylight. 3 br-int and 1 br-ex.

All Nodes: Configure ml2_conf.ini

Controller Node

Edit vi /etc/neutron/plugins/ml2/ml2_conf.ini and put the following configuration.

```
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = opendaylight
[ml2_type_gre]
tunnel_id_ranges = 1:1000
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver =
neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDrive
r
[ml2_odl]
password = admin
username = admin
url = http://<OPENDAYLIGHT MANAGEMENT
IP>:8080/controller/nb/v2/neutron
```

Network Node

Edit vi /etc/neutron/plugins/ml2/ml2_conf.ini and put the following configuration.

```
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = opendaylight
[ml2_type_flat]
flat_networks = external
[ml2_type_gre]
tunnel_id_ranges = 1:1000
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver =
neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDrive
r
[ovs]
local_ip = <TUNNEL INTERFACE IP>
enable_tunneling = True
bridge_mappings = external:br-ex
[agent]
tunnel_types = gre
[ml2_odl]
password = admin
username = admin
url = http://<OPENDAYLIGHT MANAGEMENT
IP>:8080/controller/nb/v2/neutron
```

Compute Nodes

Edit vi /etc/neutron/plugins/ml2/ml2_conf.ini and put the following configuration.

```
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = opendaylight
[ml2_type_gre]
tunnel_id_ranges = 1:1000
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver =
neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDrive
r
[ovs]
local_ip = <TUNNEL INTERFACE IP>
enable_tunneling = True
[agent]
tunnel_types = gre
[ml2_odl]
password = admin
username = admin
url = http://<OPENDAYLIGHT MANAGEMENT
IP>:8080/controller/nb/v2/neutron
```

Controller Node: Configure Neutron Database

Reset the neutron database, in order to be configured with OpenDaylight.

```
mysql -uroot -p
drop database neutron;
create database neutron;
grant all privileges on neutron.* to 'neutron'@'localhost'
identified by '<YOUR NEUTRON PASSWORD>';
grant all privileges on neutron.* to 'neutron'@'%' identified by
'<YOUR NEUTRON PASSWORD>';
exit
su -s /bin/sh -c "neutron-db-manage --config-file
/etc/neutron/neutron.conf --config-file
/etc/neutron/plugins/ml2/ml2_conf.ini upgrade juno" neutron
```

If everything is ok, without errors you can start the neutron-server.

```
service neutron-server start
```

Controller Node: Create Initial Networks

Controller Node: Launch Instances

Get preferred <HYPERVISOR NAME> from the command below.

```
source admin-openrc
nova hypervisor-list
```

Get demo <NETWORK ID> from the command below.

Get <IMAGE NAME> from the command below.

```
source demo-openrc
neutron net-list
nova image-list

nova boot --flavor m1.tiny --image <IMAGE NAME> --nic net-
id=<NETWORK ID> test1 --availability_zone=nova:<HYPERVISOR NAME>
nova boot --flavor m1.tiny --image <IMAGE NAME> --nic net-
id=<NETWORK ID> test2 --availability_zone=nova:<HYPERVISOR NAME>
nova boot --flavor m1.tiny --image <IMAGE NAME> --nic net-
id=<NETWORK ID> test3 --availability_zone=nova:<HYPERVISOR NAME>
nova boot --flavor m1.tiny --image <IMAGE NAME> --nic net-
id=<NETWORK ID> test4 --availability_zone=nova:<HYPERVISOR NAME>
```

Launch the instances!

Verify the Integration

If everything works correctly, you will be able to ping every VM. Also, you should be able to see the GRE tunnels from ovs-vsctl show in each node.

At this point, you should have the entire system up and running. To verify this, you can do the following:

- Point your web browser at the OpenStack Horizon GUI and login using your tenant credentials.
- Point your web browser at the OpenDaylight GUI and login using your OpenDaylight credentials.
- You can play around in both GUIs and attempt to launch new instances in OpenStack. As you launch VMs, you will see that OpenDaylight creates tunnel ports and links between compute hosts, which will become visible with a refresh in the OpenDaylight GUI.

6.3. Annex C – Testing environment installation and validation

6.3.1. Automated installation of the testing environment

The automated installation of the testing environment is performed through the execution of bash scripts. The first script, install_rally.sh, installs OpenStack Rally and OpenStack Tempest (installation via Rally). The second script, install_robot.sh, installs Robot Framework for OpenDaylight testing.

File: install_rally.sh

```
#!/usr/bin/env bash
# This script installs Rally.
# Specifically, it is able to install and configure
# Rally either globally (system-wide), or isolated in
# a virtual environment using the virtualenv tool.
# NOTE: The script assumes that you have the following
# programs already installed:
# -> Python 2.6, Python 2.7 or Python 3.4
set -e
PROG=$(basename "${0}")
running_as_root() {
  test "$(/usr/bin/id -u)" -eq 0
}
VERBOSE=""
ASKCONFIRMATION=1
RECREATEDEST="ask"
USEVIRTUALENV="yes"
# ansi colors for formatting heredoc
ESC=$(printf "\e")
GREEN="$ESC[0;32m"
NO_COLOR="$ESC[0;0m"
RED="$ESC[0;31m"

PYTHON2=$(which python || true)
PYTHON3=$(which python3 || true)
PYTHON=${PYTHON2:-$PYTHON3}
BASE_PIP_URL=${BASE_PIP_URL:-"https://pypi.python.org/simple"}
VIRTUALENV_191_URL="https://raw.github.com/pypa/virtualenv/1.9.1/virtualenv.py"

RALLY_GIT_URL="https://git.openstack.org/openstack/rally"
RALLY_GIT_BRANCH="master"
RALLY_CONFIGURATION_DIR=/etc/rally
RALLY_DATABASE_DIR=/var/lib/rally/database
DBTYPE=sqlite
DBNAME=rally.sqlite

# Variable used by script_interrupted to know what to cleanup
CURRENT_ACTION="none"

## Exit status codes (mostly following <sysexits.h>)
# successful exit
EX_OK=0

# wrong command-line invocation
EX_USAGE=64

# missing dependencies (e.g., no C compiler)
EX_UNAVAILABLE=69

# wrong python version
EX_SOFTWARE=70
```

```
# cannot create directory or file
EX_CANTCREAT=73

# user aborted operations
EX_TEMPFAIL=75

# misused as: unexpected error in some script we call
EX_PROTOCOL=76

# abort RC [MSG]
#
# Print error message MSG and abort shell execution with exit code RC.
# If MSG is not given, read it from STDIN.
#
abort () {
  local rc="$1"
  shift
  (echo -en "$RED$PROG: ERROR: $NO_COLOR";
      if [ $# -gt 0 ]; then echo "$@"; else cat; fi) 1>&2
  exit "$rc"
}

# die RC HEADER <<...
#
# Print an error message with the given header, then abort shell
# execution with exit code RC.  Additional text for the error message
# *must* be passed on STDIN.
#
die () {
  local rc="$1"
  header="$2"
  shift 2
  cat 1>&2 <<__EOF__
$RED==========================================================
$PROG: ERROR: $header
==========================================================
$NO_COLOR
__EOF__
  if [ $# -gt 0 ]; then
      # print remaining arguments one per line
      for line in "$@"; do
          echo "$line" 1>&2;
      done
  else
      # additional message text provided on STDIN
      cat 1>&2;
  fi
  cat 1>&2 <<__EOF__

If the above does not help you resolve the issue, please contact the
Rally team by sending an email to the OpenStack mailing list
openstack-dev@lists.openstack.org. Include the full output of this
script to help us identifying the problem.
$RED
Aborting installation!$NO_COLOR
__EOF__
  exit "$rc"
}

script_interrupted () {
    echo "Interrupted by the user. Cleaning up..."
    [ -n "${VIRTUAL_ENV}" -a "${VIRTUAL_ENV}" == "$VENVDIR" ] && deactivate

    case $CURRENT_ACTION in
```

```
        creating_venv|venv-created)
            if [ -d "$VENVDIR" ]
            then
                if ask_yn "Do you want to delete the virtual environment in
  '$VENVDIR'?"
                then
                    rm -rf "$VENVDIR"
                fi
            fi
            ;;
        downloading-src|src-downloaded)
            # This is only relevant when installing with --system,
            # otherwise the git repository is cloned into the
            # virtualenv directory
            if [ -d "$SOURCEDIR" ]
            then
                if ask_yn "Do you want to delete the downloaded source in
  '$SOURCEDIR'?"
                then
                    rm -rf "$SOURCEDIR"
                fi
            fi
            ;;
    esac

    abort $EX_TEMPFAIL "Script interrupted by the user"
}

trap script_interrupted SIGINT

print_usage () {
    cat <<__EOF__
Usage: $PROG [options]

This script will install Rally in your system.

Options:
$GREEN  -h, --help             $NO_COLOR Print this help text
$GREEN  -v, --verbose          $NO_COLOR Verbose mode
$GREEN  -s, --system           $NO_COLOR Install system-wide.
$GREEN  -d, --target DIRECTORY$NO_COLOR Install Rally virtual environment into
  DIRECTORY.
                         (Default: $HOME/rally if not root).
$GREEN  --url                  $NO_COLOR Git repository public URL to download Rally
  from.
                         This is useful when you have only installation script and
  want to install Rally
                         from custom repository.
                         (Default: ${RALLY_GIT_URL}).
                         (Ignored when you are already in git repository).
$GREEN  --branch               $NO_COLOR Git branch name name or git tag (Rally
  release) to install.
                         (Default: latest - master).
                         (Ignored when you are already in git repository).
$GREEN  -f, --overwrite        $NO_COLOR Deprecated. Use -r instead.
$GREEN  -r, --recreate         $NO_COLOR Remove target directory if it already exist.
                         If neither '-r' nor '-R' is set default behaviour is to ask.
$GREEN  -R, --no-recreate      $NO_COLOR Do not reemove target directory if it already
  exist.
                         If neither '-r' nor '-R' is set default behaviour is to ask.
$GREEN  -y, --yes              $NO_COLOR Do not ask for confirmation: assume a 'yes'
  reply
                         to every question.
$GREEN  -D, --dbtype TYPE      $NO_COLOR Select the database type. TYPE can be one of
                         'sqlite', 'mysql', 'postgres'.
                         Default: sqlite
$GREEN  --db-user USER        $NO_COLOR Database user to use. Only used when --dbtype
                         is either 'mysql' or 'postgres'.
$GREEN  --db-password PASSWORD$NO_COLOR Password of the database user. Only used when
                         --dbtype is either 'mysql' or 'postgres'.
$GREEN  --db-host HOST        $NO_COLOR Database host. Only used when --dbtype is
                         either 'mysql' or 'postgres'
$GREEN  --db-name NAME        $NO_COLOR Name of the database. Only used when --dbtype
  is
                         either 'mysql' or 'postgres'
$GREEN  -p, --python EXE      $NO_COLOR The python interpreter to use. Default:
  $PYTHON
$GREEN  --develop             $NO_COLOR Install Rally with editable source code try.
                         (Default: false)
$GREEN  --no-color            $NO_COLOR Disable output coloring.

__EOF__
}

# ask_yn PROMPT
#
# Ask a Yes/no question preceded by PROMPT.
# Set the env. variable REPLY to 'yes' or 'no'
# and return 0 or 1 depending on the users'
# answer.
#
ask_yn () {
    if [ $ASKCONFIRMATION -eq 0 ]; then
        # assume 'yes'
        REPLY='yes'
        return 0
    fi
    while true; do
        read -p "$1 [yN] " REPLY
        case "$REPLY" in
            [Yy]*)    REPLY='yes'; return 0 ;;
            [Nn]*|'') REPLY='no';  return 1 ;;
            *)        echo "Please type 'y' (yes) or 'n' (no)." ;;
        esac
    done
}

have_command () {
  type "$1" >/dev/null 2>/dev/null
}

require_command () {
  if ! have_command "$1"; then
    abort 1 "Could not find required command '$1' in system PATH. Aborting."
  fi
}

require_python () {
    require_command "$PYTHON"
    if "$PYTHON" -c 'import sys; sys.exit(sys.version_info[:2] >= (2, 6))'
    then
        die $EX_UNAVAILABLE "Wrong version of python is installed" <<__EOF__

Rally requires Python version 2.6+. Unfortunately, we do not support
your version of python: $("$PYTHON" -V 2>&1 | sed 's/python//gi').

If a version of Python suitable for using Rally is present in some
non-standard location, you can specify it from the command line by
running this script again with option '--python' followed by the path of
the correct 'python' binary.
```

```
__EOF__
    fi
}

have_sw_package () {
    # instead of guessing which distribution this is, we check for the
    # package manager name as it basically identifies the distro
    if have_command dpkg; then
        (dpkg -l "$1" | egrep -q ^i ) >/dev/null 2>/dev/null
    elif have_command rpm; then
        rpm -q "$1" >/dev/null 2>/dev/null
    fi
}

which_missing_packages () {
    local missing=''
    for pkgname in "$@"; do
        if have_sw_package "$pkgname"; then
            continue;
        else
            missing="$missing $pkgname"
        fi
    done
    echo "$missing"
}

# Download command
download() {
    wget -nv $VERBOSE --no-check-certificate -O "$@";
}

download_from_pypi () {
    local pkg=$1
    local url=$(download - "$BASE_PIP_URL"/"$pkg"/ | sed -n '/source\/.\/'"$pkg"'.*gz/
  { s:.*href="\([^#"]*\)["#].*:\1:g; p; }' | sort | tail -1)
    if [ -n "$url" ]; then
        download "$(basename "$url")" "$BASE_PIP_URL"/"$pkg"/"$url"
    else
        die $EX_PROTOCOL "Package '$pkg' not found on PyPI!" <<__EOF__
Unable to download package '$pkg' from PyPI.
__EOF__
    fi
}

install_required_sw () {
    # instead of guessing which distribution this is, we check for the
    # package manager name as it basically identifies the distro
    local missing pkg_manager
    if have_command apt-get; then
        # Debian/Ubuntu
        missing=$(which_missing_packages build-essential libssl-dev libffi-dev python-
  dev libxml2-dev libxslt1-dev libpq-dev git wget)

        if [ "$ASKCONFIRMATION" -eq 0 ]; then
            pkg_manager="apt-get install --yes"
        else
            pkg_manager="apt-get install"
        fi
    elif have_command yum; then
        # RHEL/CentOS
        missing=$(which_missing_packages gcc libffi-devel python-devel openssl-devel
  gmp-devel libxml2-devel libxslt-devel postgresql-devel git wget)

        if [ "$ASKCONFIRMATION" -eq 0 ]; then
            pkg_manager="yum install -y"
        else
            pkg_manager="yum install"
        fi
    elif have_command zypper; then
        # SuSE
        missing=$(which_missing_packages gcc libffi48-devel python-devel openssl-devel
  gmp-devel libxml2-devel libxslt-devel postgresql93-devel git wget)

        if [ "$ASKCONFIRMATION" -eq 0 ]; then
            pkg_manager="zypper -n --no-gpg-checks --non-interactive install --auto-
  agree-with-licenses"
        else
            pkg_manager="zypper install"
        fi
    else
        # MacOSX maybe?
        echo "Cannot determine what package manager this system has, so I cannot check
  if requisite software is installed. I'm proceeding anyway, but you may run into
  errors later."
    fi
    if ! have_command pip; then
        missing="$missing python-pip"
    fi

    if [ -n "$missing" ]; then
        cat <<__EOF__
The following software packages need to be installed
in order for Rally to work:$GREEN $missing
$NO_COLOR
__EOF__

        # If we are root
        if running_as_root; then
            cat <<__EOF__
In order to install the required software you would need to run as
'root' the following command:
$GREEN
    $pkg_manager $missing
$NO_COLOR
__EOF__
            # ask if we have to install it
            if ask_yn "Do you want me to install these packages for you?"; then
                # install
                if [[ "$missing" == *python-pip* ]]; then
                    missing=${missing//python-pip/}
                    if ! $pkg_manager python-pip; then
                        if ask_yn "Error installing python-pip. Install from external
  source?"; then
                            local pdir=$(mktemp -d)
                            local getpip="$pdir/get-pip.py"
                            download "$getpip"
  https://raw.github.com/pypa/pip/master/contrib/get-pip.py
                            if ! "$PYTHON" "$getpip"; then
                                abort $EX_PROTOCOL "Error while installing python-pip
  from external source."
                            fi
                        else
                            abort $EX_TEMPFAIL \
                                "Please install python-pip manually."
                        fi
                    fi
                fi
                if ! $pkg_manager $missing; then
                    abort $EX_UNAVAILABLE "Error while installing $missing"
                fi
                # installation successful
            else # don't want to install the packages
                die $EX_UNAVAILABLE "missing software prerequisites" <<__EOF__
Please, install the required software before installing Rally

__EOF__
            fi
        else # Not running as root
            cat <<__EOF__
There is a small chance that the required software
is actually installed though we failed to detect it,
so you may choose to proceed with Rally installation
anyway.  Be warned however, that continuing is very
likely to fail!

__EOF__
            if ask_yn "Proceed with installation anyway?"
            then
                echo "Proceeding with installation at your request... keep fingers
  crossed!"
            else
                die $EX_UNAVAILABLE "missing software prerequisites" <<__EOF__
Please ask your system administrator to install the missing packages,
or, if you have root access, you can do that by running the following
command from the 'root' account:
$GREEN
    $pkg_manager $missing
$NO_COLOR
__EOF__
            fi
        fi
    fi

}

install_db_connector () {
    case $DBTYPE in
        mysql)
            pip install pymysql
            ;;
        postgres)
            pip install psycopg2
            ;;
    esac
}

install_virtualenv () {
    DESTDIR=$1

    if [ -n "$VIRTUAL_ENV" ]; then
        die $EX_SOFTWARE "Virtualenv already active" <<__EOF__
A virtual environment seems to be already active. This will cause
this script to FAIL.

Run 'deactivate', then run this script again.
__EOF__
    fi

    # Use the latest virtualenv that can use `.tar.gz` files
    VIRTUALENV_DST="$DESTDIR/virtualenv-191.py"
    mkdir -p "$DESTDIR"
    download "$VIRTUALENV_DST" "$VIRTUALENV_191_URL"
    "$PYTHON" "$VIRTUALENV_DST" $VERBOSE -p "$PYTHON" "$DESTDIR"

    . "$DESTDIR"/bin/activate
    # Recent versions of `pip` insist that setuptools>=0.8 is installed,
    # because they try to use the "wheel" format for any kind of package.
    # So we need to update setuptools, or `pip` will error out::
    #
    #     Wheel installs require setuptools >= 0.8 for dist-info support.
    #
    if pip wheel --help 1>/dev/null 2>/dev/null; then
        (cd "$DESTDIR" && download_from_pypi setuptools)
        # setup.py must be called with `python', which will be the
        # python executable inside the virtualenv, not `$PYTHON',
        # which is the system python.
        if ! (cd "$DESTDIR" && tar -xzf setuptools-*.tar.gz && \
              cd setuptools-* && python setup.py install);
        then
            die $EX_SOFTWARE \
                "Failed to install the latest version of Python 'setuptools'"
  <<__EOF__

The required Python package setuptools could not be installed.

__EOF__
        fi
    fi
}

setup_rally_configuration () {
    SRCDIR=$1
    ETCDIR=$RALLY_CONFIGURATION_DIR
    DBDIR=$RALLY_DATABASE_DIR

    [ -d "$ETCDIR" ] || mkdir -p "$ETCDIR"
    cp "$SRCDIR"/etc/rally/rally.conf.sample "$ETCDIR"/rally.conf

    [ -d "$DBDIR" ] || mkdir -p "$DBDIR"
    local CONF_TMPFILE=$(mktemp)
    sed "s|#connection *=.*|connection = \"$DBCONNSTRING\"|" "$ETCDIR"/rally.conf >
  "$CONF_TMPFILE"
    cat "$CONF_TMPFILE" > "$ETCDIR"/rally.conf
    rm "$CONF_TMPFILE"
    rally-manage db recreate
}

rally_venv () {
    echo "Installing Rally virtualenv in directory '$VENVDIR' ..."
    CURRENT_ACTION="creating-venv"
    if ! install_virtualenv "$VENVDIR"; then
        die $EX_PROTOCOL "Unable to create a new virtualenv in '$VENVDIR':
  'virtualenv.py' script exited with code $rc." <<__EOF__
The script was unable to create a valid virtual environment.
__EOF__
    fi
    CURRENT_ACTION="venv-created"
    rc=0
}

### Main program ###
short_opts='d:vsyfrRhD:p:'
long_opts='target:,verbose,overwrite,recreate,no-
  recreate,system,yes,dbtype:,python:,db-user:,db-password:,db-host:,db-
  name:,help,url:,branch:,develop,no-color'

set +e
if [ "x$(getopt -T)" = 'x' ]; then
    # GNU getopt
    args=$(getopt --name "$PROG" --shell sh -l "$long_opts" -o "$short_opts" -- "$@")
    if [ $? -ne 0 ]; then
        abort 1 "Type '$PROG --help' to get usage information."
    fi
    # use 'eval' to remove getopt quoting
    eval set -- "$args"
else
    # old-style getopt, use compatibility syntax
    args=$(getopt "$short_opts" "$@")
    if [ $? -ne 0 ]; then
        abort 1 "Type '$PROG -h' to get usage information."
    fi
    eval set -- "$args"
fi
set -e

# Command line parsing
while true
do
    case "$1" in
        -d|--target)
            shift
            VENVDIR=$(readlink -m "$1")
            ;;
        -h|--help)
            print_usage
            exit $EX_OK
            ;;
        -v|--verbose)
            VERBOSE="-v"
            ;;
        -s|--system)
            USEVIRTUALENV="no"
            ;;
        -f|--overwrite)
            RECREATEDEST=yes
            ;;
        -r|--recreate)
            RECREATEDEST=yes
            ;;
        -R|--no-recreate)
            RECREATEDEST=no
            ;;
        -y|--yes)
            ASKCONFIRMATION=0
            ;;
        --url)
            shift
            RALLY_GIT_URL=$1
            ;;
        --branch)
            shift
            RALLY_GIT_BRANCH=$1
            ;;
        -D|--dbtype)
            shift
            DBTYPE=$1
            case $DBTYPE in
                sqlite|mysql|postgres);;
                *)
                    print_usage | die $EX_USAGE \
                        "An invalid option has been detected."
                    ;;
            esac
            ;;
        --db-user)
            shift
            DBUSER=$1
            ;;
        --db-password)
            shift
            DBPASSWORD=$1
            ;;
        --db-host)
            shift
            DBHOST=$1
            ;;
        --db-name)
            shift
            DBNAME=$1
            ;;
        -p|--python)
            shift
            PYTHON=$1
            ;;
        --develop)
            DEVELOPMENT_MODE=true
            ;;
        --no-color)
            RED=""
            GREEN=""
            NO_COLOR=""
            ;;
        --)
            shift
            break
            ;;
        *)
            print_usage | die $EX_USAGE "An invalid option has been detected."
    esac
    shift
done

### Post-processing ###

if [ "$USEVIRTUALENV" == "no" ] && [ -n "$VENVDIR" ]; then
    die $EX_USAGE "Ambiguous arguments" <<__EOF__
Option -d/--target can not be used with --system.
__EOF__
fi

if running_as_root; then
    if [ -z "$VENVDIR" ]; then
        USEVIRTUALENV='no'
    fi
else
    if [ "$USEVIRTUALENV" == 'no' ]; then
        die $EX_USAGE "Insufficient privileges" <<__EOF__
$REDRoot permissions required in order to install system-wide.
As non-root user you may only install in virtualenv.$NO_COLOR
__EOF__
    fi
    if [ -z "$VENVDIR" ]; then
        VENVDIR="$HOME"/rally
    fi
fi

# Fix RALLY_DATABASE_DIR if virtualenv is used
if [ "$USEVIRTUALENV" = 'yes' ]
then
```

```
    RALLY_CONFIGURATION_DIR=$VENVDIR/etc/rally
    RALLY_DATABASE_DIR="$VENVDIR"/database
fi

if [ "$DBTYPE" = 'sqlite' ]; then
    if [ "${DBNAME:0:1}" = '/' ]; then
        DBFILE="$DBNAME"
    else
        DBFILE="${RALLY_DATABASE_DIR}/${DBNAME}"
    fi
    DBCONNSTRING="sqlite:///${DBFILE}"
else
    if [ -z "$DBUSER" -o -z "$DBPASSWORD" -o -z "$DBHOST" -o -z "$DBNAME" ]
    then
        die $EX_USAGE "Missing mandatory options" <<__EOF__
When specifying a database type different than 'sqlite', you also have
to specify the database name, host, and username and password of a
valid user with write access to the database.

Please, re-run the script with valid values for the options:
$GREEN
    --db-host
    --db-name
    --db-user
    --db-password$NO_COLOR
__EOF__
    fi
    DBAUTH="$DBUSER:$DBPASSWORD@$DBHOST"
    DBCONNSTRING="$DBTYPE://$DBAUTH/$DBNAME"
fi

# check and install prerequisites
install_required_sw
require_python


# Install virtualenv, if required
if [ "$USEVIRTUALENV" = 'yes' ]; then
    if [ -d "$VENVDIR" ]
    then
        if [ $RECREATEDEST = 'ask' ]; then
            echo "Destination directory '$VENVDIR' already exists."
            echo "I can wipe it out in order to make a new installation,"
            echo "but this means any files in that directory, and the ones"
            echo "underneath it will be deleted."
            echo

            if ! ask_yn "Do you want to wipe the installation directory '$VENVDIR'?"
            then
                echo "*Not* overwriting destination directory '$VENVDIR'."
                RECREATEDEST=no
            else
                RECREATEDEST=yes

            fi
        fi

        if [ $RECREATEDEST = 'yes' ];
        then
            echo "Removing directory $VENVDIR as requested."
            rm $VERBOSE -rf "$VENVDIR"
            rally_venv
        elif [ $RECREATEDEST = 'no' ];
        then
            echo "Using existing virtualenv at $VENVDIR..."
            . "$VENVDIR"/bin/activate
        else
            abort 66 "Internal error: unexpected value '$RECREATEDEST' for
  RECREATEDEST."
        fi
    else
        rally_venv
    fi
fi

# Install rally
ORIG_WD=$(pwd)

BASEDIR=$(dirname "$(readlink -e "$0")")

# If we are inside the git repo, don't download it again.
if [ -d "$BASEDIR"/.git ]
then
    SOURCEDIR=$BASEDIR
    pushd $BASEDIR > /dev/null
    if find . -name '*.py[co]' -exec rm -f {} +
    then
        echo "Wiped python compiled files."
    else
        echo "Warning! Unable to wipe python compiled files"
    fi

    if [ "$USEVIRTUALENV" = 'yes' ]
    then
        if [ "$VENVDIR/src" != "$BASEDIR" ]
        then
            SOURCEDIR="$VENVDIR"/src
            if [ -d $SOURCEDIR ]
            then
                rm -rf $SOURCEDIR
            fi
            mkdir $SOURCEDIR
            cp -r . $SOURCEDIR/
        fi
    fi
    popd > /dev/null
else
    if [ "$USEVIRTUALENV" = 'yes' ]
    then
        SOURCEDIR="$VENVDIR"/src
    else
        SOURCEDIR="$ORIG_WD"/rally.git
    fi

    # Check if source directory is present
    if [ -d "$SOURCEDIR" ]
    then
        if [ $RECREATEDEST != 'yes' ]
        then
            echo "Source directory '$SOURCEDIR' already exists."
            echo "I can wipe it out in order to make a new installation,"
            echo "but this means any files in that directory, and the ones"
            echo "underneath it will be deleted."
            echo
            if ! ask_yn "Do you want to wipe the source directory '$SOURCEDIR'?"
            then
                echo "*Not* overwriting destination directory '$SOURCEDIR'."
            else
                rm -rf $SOURCEDIR
                if [ -d "$SOURCEDIR"/.git ]
                then
                    abort $EX_CANTCREAT "Unable to wipe source directory $SOURCEDIR"
                fi
            fi
        fi
    fi

    if ! [ -d "$SOURCEDIR"/.git ]
    then
        echo "Downloading Rally from subversion repository $RALLY_GIT_URL ..."
        CURRENT_ACTION="downloading-src"
        git clone "$RALLY_GIT_URL" -b "$RALLY_GIT_BRANCH" "$SOURCEDIR"
        if ! [ -d $SOURCEDIR/.git ]
            then
            abort $EX_CANTCREAT "Unable to download git repository"
        fi
        CURRENT_ACTION="src-downloaded"
    fi
fi

install_db_connector

# Install rally
cd "$SOURCEDIR"
# Get latest available pip and reset shell cache
pip install -i $BASE_PIP_URL -U 'pip'
hash -r

# Install dependencies
pip install -i $BASE_PIP_URL pbr 'tox<=1.6.1'
# Uninstall possible previous version
pip uninstall -y rally || true
# Install rally
if [ $DEVELOPMENT_MODE ]
then
    pip install -i $BASE_PIP_URL -e .
else
    pip install -i $BASE_PIP_URL .
fi

cd "$ORIG_WD"

# Post-installation
if [ "$USEVIRTUALENV" = 'yes' ]
then
    # Fix bash_completion
    cat >> "$VENVDIR"/bin/activate <<__EOF__

. "$VENVDIR/etc/bash_completion.d/rally.bash_completion"
__EOF__

    setup_rally_configuration "$SOURCEDIR"

    if ! [ $DEVELOPMENT_MODE ]
    then
        SAMPLESDIR=$VENVDIR/samples
        mkdir -p $SAMPLESDIR
        cp -r $SOURCEDIR/samples/* $SAMPLESDIR/
        if [ "$BASEDR" != "$SOURCEDIR" ]
        then
            rm -rf $SOURCEDIR
            echo "Source directory is removed."
        else
            echo "Unabled to remove source directory, becaus this script was started
  from it."
```

```
        fi
    else
        SAMPLESDIR=$SOURCEDIR/samples
    fi

    cat <<__EOF__
$GREEN==============================
Installation of Rally is done!
==============================
$NO_COLOR
In order to work with Rally you have to enable the virtual environment
with the command:

    . $VENVDIR/bin/activate

You need to run the above command on every new shell you open before
using Rally, but just once per session.

Information about your Rally installation:

  * Method:$GREEN virtualenv$NO_COLOR
  * Virtual Environment at:$GREEN $VENVDIR$NO_COLOR
  * Database at:$GREEN $RALLY_DATABASE_DIR$NO_COLOR
  * Configuration file at:$GREEN $RALLY_CONFIGURATION_DIR$NO_COLOR
  * Samples at:$GREEN $SAMPLESDIR$NO_COLOR

__EOF__
else
    setup_rally_configuration "$SOURCEDIR"

    if ! [ $DEVELOPMENT_MODE ]
    then
        SAMPLESDIR=/usr/share/rally/samples
        mkdir -p $SAMPLESDIR
        cp -r $SOURCEDIR/samples/* $SAMPLESDIR/
        if [ "$BASEDIR" != "$SOURCEDIR" ]
        then
            rm -rf $SOURCEDIR
            echo "Source directory is removed."
        else
            echo "Unabled to remove source directory, because this script was started
  from it."

        fi
    else
        SAMPLESDIR=$SOURCEDIR/samples
    fi
    ln -s /usr/local/etc/bash_completion.d/rally.bash_completion
  /etc/bash_completion.d/ 2> /dev/null || true
    if [ -f "${DBFILE}" ]; then
        chmod 777 "$DBFILE"
    fi

    cat <<__EOF__
$GREEN==============================
Installation of Rally is done!
==============================
$NO_COLOR
Rally is now installed in your system. Information about your Rally
installation:

  * Method:$GREEN system$NO_COLOR
  * Database at:$GREEN $RALLY_DATABASE_DIR$NO_COLOR
  * Configuration file at:$GREEN $RALLY_CONFIGURATION_DIR$NO_COLOR
  * Samples at:$GREEN $SAMPLESDIR$NO_COLOR
```

```
__EOF__
fi
```

File: install_robot.sh

```
#!/bin/bash
sudo apt-get update && sudo apt-get install -y build-essential fakeroot autoconf
  libtool git automake autoconf gcc uml-utilities pkg-config linux-headers-`uname -r`
  openvswitch-switch python-qt4 python-twisted-conch
virtualenv $HOME/robot_virtualenv
source $HOME/robot_virtualenv/bin/activate
$HOME/robot_virtualenv/bin/pip install paramiko simplejson requests robotframework
  robotframework-sshlibrary -U robotframework-requests --upgrade robotframework-
  httplibrary
deactivate
cd
git clone git://github.com/mininet/mininet
sudo ./mininet/util/install.sh -nf
```

6.3.2. OpenStack Rally/Tempest installation for OpenStack cloud testing

Software Prerequisities

```
apt-get install libssl-dev libffi-dev python-dev libxml2-dev
libxslt1-dev libpq-dev git
```

Install Rally

The easiest way to install Rally is by executing its installation script:

```
wget -q -O-
https://raw.githubusercontent.com/openstack/rally/master/install_ral
ly.sh | bash
curl
https://raw.githubusercontent.com/openstack/rally/master/install_ral
ly.sh | bash
```

Or use curl:

By default, it will install Rally in a virtualenv in ~/rally when ran as standard user, or install system wide when ran as root. You can install Rally in a venv by using the option --target:

```
./install rally.sh --target /foo/bar
```

You can also install Rally system wide by running script as root and without --target

```
sudo ./install rally.sh
```

option:

Create a rally deployment from your environment

To add your cloud to Rally:

Option A: Using an OpenRC file

After the installation is done, source the openrc file to start using OpenStack:

```
source ./admin-openrc.sh
and then register a deployment to Rally:
```

```
rally deployment create --fromenv --name t-nova-cloud
```

Option B: Provide a configuration file

There is a second option here, to create a deployment by providing configuration details of your OpenStack deployment in form of a json file.

Go to the rally directory and create a my-deployment.json file. The content of the file should be like this:

```
{
    "type": "ExistingCloud",
    [illegible]
rally deployment create --filename=my-deployment.json --name=t-nova-
cloud
    [illegible]
    "admin": {
        "username": "admin",
        "password": "myadminpass",
        "tenant_name": "demo"
    },
    "https_insecure": False,
    "https_cacert": "",
}
```

Now, run 'rally deployment create' using the –file argument:

After adding your deployment, you can use `rally deployment list` to display a list of all known deployments:

Check to see if Rally has been deployed correctly, and to determine which Rally

```
rally deployment list
```

```
rally deployment check
rally show images
rally show flavors
rally show networks
```

services are available:

Running Rally Scenarios

First, choose a specific deployment to use by specifying the deployment id:

```
rally deployment use 8108584e-22b4-4e3c-b05c-d98dcdcb9f36
```

After selecting your deployment, you can run a rally sample scenario and get back the results. Here is an example of running a neutron scenario testing the creation and deletion of routers:

```
rally task start
/home/localadmin/rally/samples/tasks/scenarios/neutron/create_and_de
lete_routers.json
```

Install Tempest

You can use Rally to install and run Tempest tests. The command `rally-manage tempest install` takes care of cloning the repository, generates the configuration file for Tempest and installs the virtual environment with all dependencies:

```
rally-manage tempest install
```

Run Tempest

The command `rally verify start` launches auto-verification. This command expects only one argument: the test set name. If this argument is not specified, smoke tests (the default) will be executed. Output of this command is similar to the Tempest output. Valid test set names include: full, smoke, baremetal, compute, data_processing, identity, image, network, object_storage, orchestration, telemetry, and volume. We have decided to run smoke tests of Tempest to verify correct functionality of OpenStack at a high level:

```
rally verify start --deployment t-nova-cloud
rally verify start smoke
```

View Tempest results

You can list all tempest results with:

```
rally verify list
```

```
rally verify show <UUID-of-deployment>
rally verify detailed <UUID>
```

Detailed information for one execution can be derived with two commands:

The latter displays tracebacks for failed tests.

6.3.3. Robot Framework installation for OpenDaylight testing

We assume that Open vSwitch and OpenDaylight controller are already installed. To set up a System Test environment you need to install Robot Framework and Mininet.

Mininet Installation

Install Mininet:

```
git clone git://github.com/mininet/mininet
cd mininet/
git checkout -b 2.2.1 2.2.1
cd ./util
./install.sh -nf
```

Verify the Mininet installation with the following command:

```
sudo mn --test=pingall
```

To verify that Mininet works for OF1.3 run:

```
Verify mininet works for OF1.3:
sudo mn --controller=remote,ip=10.125.136.52 --topo tree,2 --switch
ovsk,protocols=OpenFlow13
```

To test the version of used protocol by switch “s1”:

```
sudo ovs-ofctl -O OpenFlow13 show s1
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

Robot Framework Installation

To install Robot Framework run the following commands:

```
sudo apt-get install python-pip
sudo apt-get install python-paramiko
sudo pip install requests
sudo pip install robotframework
sudo pip install robotframework-sshlibrary
sudo pip install -U robotframework-requests
sudo pip install --upgrade robotframework-httplibrary
```

Verify the Robot Framework installation with the following command:

```
pybot --version
```

OpenDaylight testing

Clone the latest tests from the OpenDaylight Integration project:

```
git clone https://github.com/yeasy/robot_tool.git
```

To run all the tests in the base suite:

```
cd /home/localadmin/robot_tool/suites/

pybot --variable topo_tree_level:2 base
```

To run a particular test (e.g. switch_manager.txt):

```
cd /home/localadmin/robot_tool/suites/base

pybot --variable topo_tree_level:2 switch_manager.txt
```

7. LIST OF ACRONYMS

Acronym	Explanation
AoE	ATA over Ethernet
APICv	Advanced Programmable Interrupt Controller virtualization
AWS S3	Amazon Simple Storage Service
BIOS	Basic Input/Output System
BKM	Best Known Methods
CIFS	Common Internet File System
CPE	Customer Premises Equipment
COTS	Commercial of the shelf
CRUD	Create Read Update and Delete
DIMM	Dual in-line Memory Module
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
SOC	System on a Chip
SR-IOV	Single Root I/O Virtualisation
TFTP	Trivial File Transfer Protocol
UDIMM	Unegistered Dual in-line Memory Module
vHG	Virtual Home Gateway
VMDq	Virtual Machine Device Queues
vPxaaS	Virtual Proxy as a Service
vSA	Virtual Security Appliance
vSBC	Virtual Session Border Controller
vTC	Virtual Transcoding Unit
VXLAN	Virtual Extensible LAN

8. References

[1] OpenStack Tempest [Online] http://docs.openstack.org/developer/tempest/

[2] OpenStack Tempest Github project [Online] https://github.com/openstack/tempest

[3] OpenStack Rally [Online] https://wiki.openstack.org/wiki/Rally

[4] OpenStack Rally samples [Online] https://github.com/openstack/rally/tree/master/samples/tasks/scenarios

[5] Robot Framework [Online] http://robotframework.org/

[6] Keith Tenzer – OpenStack Multiple Node Configurations [Online] http://keithtenzer.com/2015/01/26/openstack-multiple-node-configurations/

[7] G. Gardikis (ed.) et al, "Monitoring and Maintenance – Interim", T-NOVA Deliverable D4.41, November 2015

[8] OpenStack Kilo hypervisor feature support matrix [Online] http://docs.openstack.org/developer/nova/support-matrix.html

[9] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle, MoonGen: A Scriptable High-Speed Packet Generator, Available http://arxiv.org/ftp/arxiv/papers/1410/1410.3322.pdf

[10] Ixia, Enabling and Testing Network Functions Virtualization (NFV) to Ensure Carrier-Grade Delivery, White Paper, 915-0945-01 Rev. A, April 2014.

[11] G. Xilouris (ed.) et al, "Overall System Architecture and Interfaces", T-NOVA Deliverable D2.21, July 2014

[12] M. McGrath (ed.) et al, "Interim Report on Infrastructure Virtualisation and Management", T-NOVA Deliverable D4.01, December 2014

[13] M. McGrath (ed.) et al, "Infrastructure Virtualisation", T-NOVA Deliverable D4.1, September 2015

[14] L. Zuccaro (ed.) et al, "SDN Control Plane Interim", T-NOVA Deliverable D4.21, November 2015

[15] I. Trajkovska (ed.) et al, "SDK for SDN Interim", T-NOVA Deliverable D4.31, September 2015

[16] P. Paglierani (ed.) et al, "Network Functions Implementation and Testing", T-NOVA Deliverable D5.31, November 2015

[17] A. Gamelas (ed.) et al, "Specification of the Infrastructure Virtualisation, Management and Orchestration - Interim", T-NOVA Deliverable D2.31, September 2014

[18] Amazon Simple Storage Service [Online] https://aws.amazon.com/s3/

[19] Ceph Object Gateway S3 API [Online] http://docs.ceph.com/docs/master/radosgw/s3/

[20] DPDK supported NICs [Online] http://dpdk.org/doc/nics

[21] J. Carapinha (ed.) et al, "System Use Cases and Requirements", T-NOVA Deliverable D4.1, June 2014