OPNFV Configuration Guide

Release arno.2015.1.0 (deee452)

OPNFV

February 15, 2016

CHAPTER

ONE

ABSTRACT

This document provides guidance and instructions for the configuration of the Brahmaputra release of OPNFV.

The release includes four installer tools leveraging different technologies; Apex, Compass4nfv, Fuel and JOID, which deploy components of the platform.

This document provides a guide for the selection of tools and components including guidelines for how to deploy and configure the platform to an operational state.

CHAPTER TWO

CONFIGURATION OPTIONS

OPNFV provides a variety of virtual infrastructure deployments called scenarios designed to host virtualised network functions (VNF's). Each scenario provide specific capabilities and/or components aimed to solve specific problems for the deployment of VNF's. A scenario may include components such as OpenStack, OpenDaylight, OVS, KVM etc. where each scenario will include different source components or configurations.

2.1 OPNFV Scenario's

Each OPNFV scenario provides unique features and capabilities, it is important to understand your target platform capabilities before installing and configuring your target scenario. This configuration guide outlines how to install and configure components in order to enable the features you require.

Scenarios are implemented as deployable compositions through integration with an installation tool. OPNFV supports multiple installation tools and for any given release not all tools will support all scenarios. While our target is to establish parity across the installation tools to ensure they can provide all scenarios, the practical challenge of achieving that goal for any given feature and release results in some disparity.

2.1.1 Scenario Naming

In OPNFV scenarios are identified by short scenario names, these names follow a scheme that identifies the key components and behaviours of the scenario. The rules for scenario naming are as follows:

os-[controller]-[feature]-[mode]-[option]

Details of the fields are

    - os: mandatory
        - Refers to the platform type used
        - possible value: os (OpenStack)

- [controller]: mandatory
    - Refers to the SDN controller integrated in the platform
    - example values: nosdn, ocl, odl, onos
    - [feature]: mandatory
        - Refers to the feature projects supported by the scenario
        - example values: nofeature, kvm, ovs
    - [mode]: mandatory

        * Refers to the deployment type, which may include for instance high availability
        * possible values: ha, noha
    - [option]: optional
        * Used for the scenarios those do not fit into naming scheme.
        * The optional field in the short scenario name should not be included if there is no optional scenario.

Some examples of supported scenario names are:

- os-nosdn-kvm-noha
    - This is an OpenStack based deployment using neutron including the OPNFV enhanced KVM hypervisor
- os-odl_l2-nofeature-ha
    - This is an OpenStack deployment in high availability mode including OpenDaylight layer2 networking
- os-onos-kvm_ovs-noha
    - This is an OpenStack deployment using ONOS including OPNFV enhanced KVM and OVS versions

2.1.2 Installing your scenario

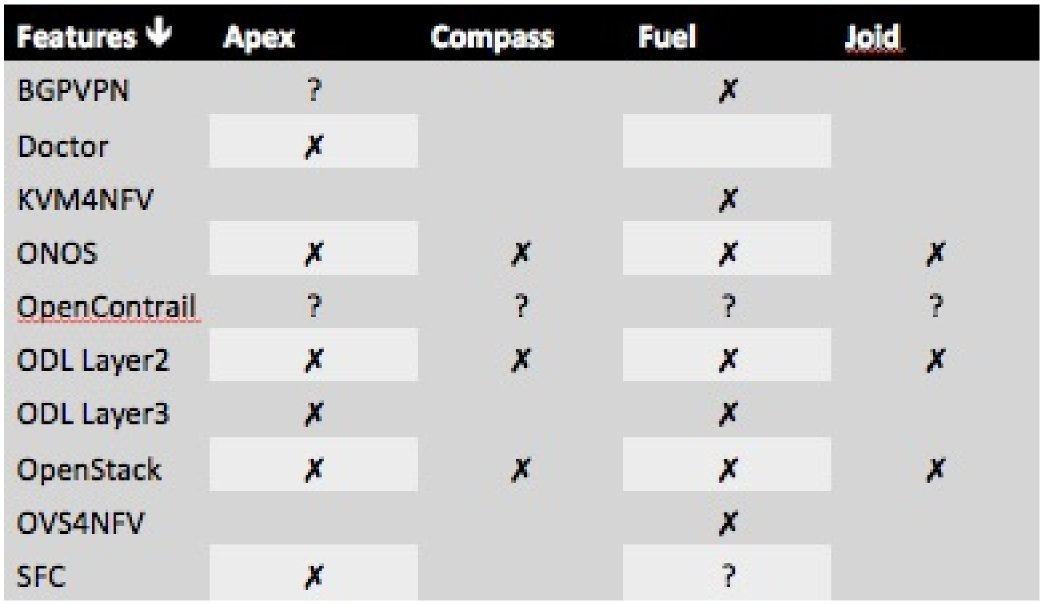
There are two main methods of deploying your target scenario, one method is to follow this guide which will walk you through the process of deploying to your hardware using scripts or ISO images, the other method is to set up a Jenkins slave and connect your infrastructure to the OPNFV Jenkins master.

For the purposes of evaluation and development a number of Brahmaputra scenarios are able to be deployed virtually to mitigate the requirements on physical infrastructure. Details and instructions on performing virtual deployments can be found in the installer specific installation instructions.

To set up a Jenkins slave for automated deployment to your lab, refer to the Jenkins slave connect guide.

2.1.3 Brahmaputra scenario overeview

The following table provides an overview of the installation tools and available scenario's in the Brahmaputra release of OPNFV.



Features ↓	Apex	Compass	Fuel	Joid
BGPVPN	?		✗	
Doctor	✗			
KVM4NFV			✗	
ONOS	✗	✗	✗	✗
OpenContrail	?	?	?	?
ODL Layer2	✗	✗	✗	✗
ODL Layer3	✗		✗	
OpenStack	✗	✗	✗	✗
OVS4NFV			✗	
SFC	✗		?	

This document will describe how to install and configure your target OPNFV scenarios. Remember to check the associated validation procedures section following your installation for details of the use cases and tests that have been run.

CHAPTER

THREE

INSTALLER CONFIGURATION

The following sections describe the per installer configuration options. Further details for each installer are captured in the referred project documentation.

3.1 Apex configuration

3.2 Fuel configuration

This section provides brief guidelines on how to install and configure the Brahmaputra release of OPNFV when using Fuel as a deployment tool including required software and hardware configurations.

For detailed instructions on how to install the Brahmaputra release using Fuel, see:<TODO>

3.2.1 Pre-configuration activities

Planning the deployment

Before starting the installation of the Brahmaputra release of OPNFV when using Fuel as a deployment tool, some planning must be done.

Familiarize yourself with the Fuel by reading the following documents:

- Fuel planning guide <https://docs.mirantis.com/openstack/fuel/fuel-7.0/planning-guide.html>
- Fuel user guide <http://docs.mirantis.com/openstack/fuel/fuel-7.0/user-guide.html>
- Fuel operations guide <http://docs.mirantis.com/openstack/fuel/fuel-7.0/operations.html>

Before the installation can start, a number of deployment specific parameters must be collected, those are:

1. Provider sub-net and gateway information
2. Provider VLAN information
3. Provider DNS addresses
4. Provider NTP addresses
5. Network overlay you plan to deploy (VLAN, VXLAN, FLAT)
6. Monitoring Options you want to deploy (Ceilometer, Syslog, etc.)
7. How many nodes and what roles you want to deploy (Controllers, Storage, Computes)
8. Other options not covered in the document are available in the links above

Retrieving the ISO image

First of all, the Fuel deployment ISO image needs to be retrieved, the .iso image of the Brahmaputra release of OPNFV when using Fuel as a deployment tool can be found at: <TODO>

Alternatively, you may build the .iso from source by cloning the opnfv/fuel git repository. Detailed instructions on how to build a Fuel OPNFV .iso can be found here: <TODO>

3.2.2 Hardware requirements

Following high level hardware requirements must be met:

HW Aspect	Requirement
# of nodes	Minimum 5 (3 for non redundant deployment): • 1 Fuel deployment master (may be virtualized) • 3(1) Controllers (1 colocated mongo/ceilometer role, 2 Ceph-OSD roles) • 1 Compute (1 co-located Ceph-OSD role)
CPU	Minimum 1 socket x86_AMD64 with Virtualization support
RAM	Minimum 16GB/server (Depending on VNF work load)
Disk	Minimum 256GB 10kRPM spinning disks
Networks	4 Tagged VLANs (PUBLIC, MGMT, STORAGE, PRIVATE) 1 Un-Tagged VLAN for PXE Boot - ADMIN Network note: These can be run on single NIC - or spread out over other nics as your hardware supports

For a detailed hardware compatibility matrix - please see: <https://www.mirantis.com/products/openstack-drivers-and-plugins/hardware-compatibility-list/>

Top of the rack (TOR) Configuration requirements

The switching infrastructure provides connectivity for the OPNFV infrastructure operations, tenant networks (East/West) and provider connectivity (North/South); it also provides needed connectivity for the storage Area Network (SAN). To avoid traffic congestion, it is strongly suggested that three physically separated networks are used, that is: 1 physical network for administration and control, one physical network for tenant private and public networks, and one physical network for SAN. The switching connectivity can (but does not need to) be fully redundant, in such case it and comprises a redundant 10GE switch pair for each of the three physically separated networks.

The physical TOR switches are **not** automatically configured from the OPNFV reference platform. All the networks involved in the OPNFV infrastructure as well as the provider networks and the private tenant VLANs needs to be manually configured.

Manual configuration of the Brahmaputra hardware platform should be carried out according to the OPNFV Pharos specification <TODO>

3.2.3 Jumphost configuration

The Jumphost server, also known as the "Fuel master" provides needed services/functions to deploy an OPNFV/OpenStack cluster as well functions for cluster life-cycle management (extensions, repair actions and upgrades).

The Jumphost server requires 2 (4 if redundancy is required) Ethernet interfaces - one for external management of the OPNFV installation, and another for jump-host communication with the OPNFV cluster.

Install the Fuel jump-host

Mount the Fuel Brahmaputra ISO file as a boot device to the jump host server, reboot it, and install the Fuel Jumphost in accordance with the instructions found here: <TODO>

3.2.4 Platform components configuration

Fuel-Plugins

Fuel plugins enable you to install and configure additional capabilities for your Fuel OPNFV based cloud, such as additional storage types, networking functionality, or NFV features developed by OPNFV.

Fuel offers an open source framework for creating these plugins, so there's a wide range of capabilities that you can enable Fuel to add to your OpenStack clouds.

The OPNFV Brahmaputra version of Fuel provides a set of pre-packaged plugins developed by OPNFV:

Plugin name	Short description
Open-Day-light	OpenDaylight provides an open-source SDN Controller providing networking features such as L2 and L3 network control, "Service Function Chaining", routing, networking policies, etc. More information on OpenDaylight in the OPNFV Brahmaputra release can be found in a separate section in this document.
ONOS	ONOS is another open-source SDN controller which in essense fill the same role as OpenDaylight. More information on ONOS in the OPNFV Brahmaputra release can be found in a separate section in this document.
BGP-VPN	BGP-VPN provides an BGP/MPLS VPN service More information on BGP-VPN in the OPNFV Brahmaputra release can be found in a separate section in this document.
OVS-NSH	OVS-NSH provides a variant of Open-vSwitch which supports "Network Service Headers" needed for the "Service function chaining" feature More information on "Service Function Chaining" in the OPNFV Brahmaputra release can be found in a in a separate section in this document.
OVS-NFV	OVS-NFV provides a variant of Open-vSwitch with carrier grade characteristics essential for NFV workloads. More information on OVS-NFV in the OPNFV Brahmaputra release can be found in a in a separate section in this document.
KVM-NFV	KVM-NFV provides a variant of KVM with improved virtualization characteristics essential for NFV workloads. More information on KVM-NFV in the OPNFV Brahmaputra release can be found in a in a separate section in this document.
VSPERF	VSPERF provides a networking characteristics test bench that facilitates characteristics/performance evaluation of vSwithches More information on VSPERF in the OPNFV Brahmaputra release can be found in a in a separate section in this document.

Additional third-party plugins can be found here: https://www.mirantis.com/products/openstack-drivers-and-plugins/fuel-plugins/ **Note: Plugins are not necessarily compatible with each other, see <TODO> for compatibility information**

The plugins come prepackaged, ready to install. To do so follow the instructions provided here: <TODO>

Fuel environment

A Fuel environment is an OpenStack instance managed by Fuel, one Fuel instance can manage several OpenStack instances/environments with different configurations, etc.

To create a Fuel instance, follow the instructions provided here: <TODO>

Provisioning of aditional features and services

Although the plugins have already previously been installed, they are not per default enabled for the environment we just created. The plugins of you choice need to be enabled and configured.

To enable a plugin, follow the instructions in the installation instructions <TODO>

For configuration of the plugins, please refer to the corresponding feature in the ????? <TODO>

Networking

All the networking aspects need to be configured in terms of: - Interfaces/NICs - VLANs - Sub-nets - Gateways - User network segmentation (VLAN/VXLAN) - DNS - NTP - etc.

For guidelines on how to configure networking, please refer to the installation instructions here: <TODO>

Node allocation

Now, it is time to allocate the nodes in your OPNFV cluster to OpenStack-, SDN-, and other feature/service roles. Some roles may require redundancy, while others don't; Some roles may be co-located with other roles, while others may not. The Fuel GUI will guide you in the allocation of roles and will not permit you to perform invalid allocations.

For detailed guide-lines on node allocation, please refer to the installation instructions: <TODO>

Off-line deployment

The OPNFV Brahmaputra version of Fuel can be deployed using on-line upstream repositories (default) or off-line using built-in local repositories on the Fuel jump-start server.

For instructions on how to configure Fuel for off-line deployment, please refer to the installation instructions: <TODO>

Deployment

You should now be ready to deploy your OPNFV Brahmaputra environment - but before doing so you may want to verify your network settings.

For further details on network verification and deployment, please refer to the installation instructions: <TODO>

3.3 Deploy JOID in your LAB

3.3.1 Bare Metal Installations:

3.3.2 Requirements as per Pharos:

3.3.3 Networking:

Minimum 2 networks

```
1. First for Admin network with gateway to access external network
2. Second for public network to consume by tenants for floating ips
```

NOTE: JOID support multiple isolated networks for data as well as storage. Based on your network options for Openstack.

Minimum 6 physical servers

1. Jump host server:

```
Minimum H/W Spec needed
CPU cores: 16
Memory: 32 GB
Hard Disk: 1(250 GB)
NIC: eth0(Admin, Management), eth1 (external network)
```

2. Control node servers (minimum 3):

```
Minimum H/W Spec
CPU cores: 16
Memory: 32 GB
Hard Disk: 1(500 GB)
NIC: eth0(Admin, Management), eth1 (external network)
```

3. Compute node servers (minimum 2):

```
Minimum H/W Spec
CPU cores: 16
Memory: 32 GB
Hard Disk: 1(1 TB) this includes the space for ceph as well
NIC: eth0(Admin, Management), eth1 (external network)
```

NOTE: Above configuration is minimum and for better performance and usage of the Openstack please consider higher spec for each nodes.

Make sure all servers are connected to top of rack switch and configured accordingly. No DHCP server should be up and configured. Only gateway at eth0 and eth1 network should be configure to access the network outside your lab.

Jump node configuration:

1. Install Ubuntu 14.04 LTS server version of OS on the nodes. 2. Install the git and bridge-utils packages on the server and configure minimum two bridges on jump host:

brAdm and brPublic cat /etc/network/interfaces

```
# The loopback network interface
auto lo
iface lo inet loopback
iface eth0 inet manual
auto brAdm
iface brAdm inet static
    address 10.4.1.1
    netmask 255.255.248.0
    network 10.4.0.0
    broadcast 10.4.7.255
    gateway 10.4.0.1
    # dns-* options are implemented by the resolvconf package, if installed
    dns-nameservers 10.4.0.2
    bridge_ports eth0
auto brPublic
iface brPublic inet static
    address 10.2.66.2
    netmask 255.255.255.0
    bridge_ports eth2
```

NOTE: If you choose to use the separate network for management, data and storage then you need to create bridge for each interface. In case of VLAN tags use the appropriate network on jump-host depend upon VLAN ID on the interface.

3.3.4 Configure JOID for your lab

Get the joid code from gerritt

git clone https://gerrit.opnfv.org/gerrit/p/joid.git

cd joid/ci

Enable MAAS

- Create a directory in maas/<company name>/<pod number>/ for example

mkdir maas/intel/pod7/

- Copy files from pod5 to pod7

cp maas/intel/pod5/ maas/intel/pod7/*

4 files will get copied: deployment.yaml environments.yaml interfaces.host lxc-add-more-interfaces

3.3.5 deployment.yaml file

Prerequisite:

1. Make sure Jump host node has been configured with bridges on each interface, so that appropriate MAAS and JUJU bootstrap VM can be created. For example if you have three network admin, data and public then I would suggest to give names like brAdm, brData and brPublic. 2. You have information about the node MAC address and power management details (IPMI IP, username, password) of the nodes used for control and compute node.

3.3.6 modify deployment.yaml

This file has been used to configure your maas and bootstrap node in a VM. Comments in the file are self explanatory and we expect fill up the information according to match lab infrastructure information. Sample deployment.yaml can be found at https://gerrit.opnfv.org/gerrit/gitweb?p=joid.git;a=blob;f=ci/maas/intel/pod5/deployment.yaml

modify joid/ci/01-deploybundle.sh

under section case $3 add the intelpod7 section and make sure you have information provided correctly. Before example consider your network has 192.168.1.0/24 your default network. and eth1 is on public network which will be used to assign the floating ip.

```
    'intelpod7' )

# As per your lab vip address list be deafult uses 10.4.1.11 - 10.4.1.20
        sed -i -- 's/10.4.1.1/192.168.1.2/g' ./bundles.yaml
       # Choose the external port to go out from gateway to use.

sed -i -- 's/#        "ext-port": "eth1"/        "ext-port": "eth1"/g' ./bundles.yaml
       ;;
```

NOTE: If you are using seprate data network then add this line below also along with other changes. which represents network 10.4.9.0/24 will be used for data network for openstack

```
        sed -i -- 's/#os-data-network: 10.4.8.0\/21/os-data-network: 10.4.9.0\/24/g' ./bund
```

modify joid/ci/02-maasdeploy.sh

under section case $1 add the intelpod7 section and make sure you have information provided correctly.

```
     'intelpod7' )
       cp maas/intel/pod7/deployment.yaml ./deployment.yaml
       ;;
```

NOTE: If you are using VLAN tags or more network for data and storage then make sure you modify the case $1 section under Enable vlan interface with maas appropriately. In the example below eth2 has been used as separate data network for tenants in openstack with network 10.4.9.0/24 on compute and control nodes.

```
   'intelpod7' )
       maas refresh
       enableautomodebyname eth2 AUTO "10.4.9.0/24" compute || true
       enableautomodebyname eth2 AUTO "10.4.9.0/24" control || true
       ;;
```

MAAS Install

After integrating the changes as mentioned above run the MAAS install. Suppose you name the integration lab as intelpod7 then run the below commands to start the MAAS deployment.

```
./02-maasdeploy.sh intelpod7
```

This will take approximately 40 minutes to couple hours depending on your environment. This script will do the following:

1. Create 2 VMs (KVM).
2. Install MAAS in one of the VMs.
3. Configure the MAAS to enlist and commission a VM for Juju bootstrap node.
4. Configure the MAAS to enlist and commission bare metal servers.

When it's done, you should be able to view MAAS webpage (http://<MAAS IP>/MAAS) and see 1 bootstrap node and bare metal servers in the 'Ready' state on the nodes page.

Virtual deployment

By default, just running the script ./02-maasdeploy.sh will automatically create the KVM VMs on a single machine and configure everything for you.

OPNFV Install

JOID allows you to deploy different combinations of OpenStack release and SDN solution in HA or non-HA mode.

For OpenStack, it supports Juno and Liberty. For SDN, it supports Openvswitch, OpenContrail, OpenDayLight and ONOS.

In addition to HA or non-HA mode, it also supports to deploy the latest from the development tree (tip).

The deploy.sh in the joid/ci directoy will do all the work for you. For example, the following deploy OpenStack Libery with OpenDayLight in a HA mode in the Intelpod7.

```
./deploy.sh -o liberty -s odl -t ha -l intelpod7 -f none
```

By default, the SDN is Openvswitch, non-HA, Liberty, Intelpod5, OPNFV Brahmaputra release and ODL_L2 for the OPNFV feature.

Possible options for each choice are as follows:

```
[-s ]
nosdn: openvswitch only and no other SDN.
odl: OpenDayLight Lithium version.
opencontrail: OpenContrail SDN.
onos: ONOS framework as SDN.

[-t ]
nonha: NO HA mode of Openstack.
```

```
   ha: HA mode of openstack.
    tip:  the tip of the development.

   [-o ]
   juno: OpenStack Juno version.
   liberty: OpenStack Liberty version.

   [-l ] etc...

default: For virtual deployment where installation will be done on KVM created using ./02-m
   intelpod5: Install on bare metal OPNFV pod5 of Intel lab.
   intelpod6
   orangepod2
   ..
   (other pods)
   Note:  if you make changes as per your pod above then please use your pod.

   [-f ]
   none: no special feature will be enabled.
   ipv6: ipv6 will be enabled for tenant in openstack.
```

By default debug is enabled in script and error messages will be printed on the SSH terminal where you are running the scripts. It could take an hour to couple hours (max) to complete.

Is the deployment done successfully?

Once juju-deployer is complete, use juju status to verify that all deployed unit are in the ready state.

```
juju status --format tabular
```

Find the Openstack-dashboard IP address from the *juju status* output, and see if you can log in via browser. The username and password is admin/openstack.

Optionall, see if you can log in Juju GUI. Juju GUI is on the Juju bootstrap node which is the second VM you define in the 02-maasdeploy.sh. The username and password is admin/admin.

If you deploy ODL, OpenContrail or ONOS, find the IP address of the web UI and login. Please refer to each SDN bundle.yaml for username/password.

Troubleshoot

To access to any deployed units, juju ssh for example to login into nova-compute unit and look for /var/log/juju/unit-<of interest> for more info.

```
juju ssh nova-compute/0
```

Example:

```
   ubuntu@R4N4B1:~$ juju ssh nova-compute/0
   Warning:  Permanently added '172.16.50.60' (ECDSA) to the list of known
hosts.
   Warning:  Permanently added '3-r4n3b1-compute.maas' (ECDSA) to the list of
known hosts.
   Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-77-generic x86_64)

    * Documentation:  https://help.ubuntu.com/
   <skipped>
   Last login:  Tue Feb 2 21:23:56 2016 from bootstrap.maas
   ubuntu@3-R4N3B1-compute:~$ sudo -i
   root@3-R4N3B1-compute:~# cd /var/log/juju/
   root@3-R4N3B1-compute:/var/log/juju# ls
   machine-2.log unit-ceilometer-agent-0.log unit-ceph-osd-0.log
unit-neutron-contrail-0.log unit-nodes-compute-0.log unit-nova-compute-0.log
unit-ntp-0.log
   root@3-R4N3B1-compute:/var/log/juju#
```

By default juju will add the Ubuntu user keys for authentication into the deployed server and only ssh access will be available.

Once you resolve the error, go back to the jump host to rerun the charm hook with:

```
   juju resolved --retry <unit>
```

CHAPTER

FOUR

FEATURE CONFIGURATION

The following sections describe the configuration options for specific platform features provided in Brahmaputra. Further details for each feature are captured in the referred project documentation.

4.1 Copper configuration

This release focused on use of the OpenStack Congress service for managing configuration policy. The Congress install procedure described here is largely manual. This procedure, as well as the longer-term goal of automated installer support, is a work in progress. The procedure is further specific to one OPNFV installer (JOID, i.e. MAAS/JuJu) based environment. Support for other OPNFV installer deployed environments is also a work in progress.

4.1.1 Pre-configuration activities

This procedure assumes OPNFV has been installed via the JOID installer.

4.1.2 Hardware configuration

There is no specific hardware configuration required for the Copper project.

4.1.3 Feature configuration

Following are instructions for installing Congress on an Ubuntu 14.04 LXC container in the OPNFV Controller node, as installed by the JOID installer. This guide uses instructions from the Congress intro guide on readthedocs. Specific values below will need to be modified if you intend to repeat this procedure in your JOID-based install environment.

Install base VM for congress on controller node

```
sudo juju ssh ubuntu@192.168.10.21
```

Clone the container

```
sudo lxc-clone -o juju-trusty-lxc-template -n juju-trusty-congress
```

Start the container

```
sudo lxc-start -n juju-trusty-congress -d
```

Get the container IP address

```
sudo lxc-info -n juju-trusty-congress
```

If you need to start over

```
sudo lxc-destroy --name juju-trusty-congress
```

Exit from controller (back to jumphost) and login to congress container

```
sudo juju ssh ubuntu@192.168.10.117
```

Update package repos

```
sudo apt-get update
```

Setup environment variables

```
export CONGRESS_HOST=192.168.10.106
export KEYSTONE_HOST=192.168.10.119
export CEILOMETER_HOST=192.168.10.116
export CINDER_HOST=192.168.10.117
export GLANCE_HOST=192.168.10.118
export NEUTRON_HOST=192.168.10.125
export NOVA_HOST=192.168.10.121
```

Install pip

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

Install java

```
sudo apt-get install default-jre -y
```

Install other dependencies

```
# when prompted, set and remember mysql root user password
sudo apt-get install git gcc python-dev libxml2 libxslt1-dev libzip-dev \
mysql-server python-mysqldb -y
sudo pip install virtualenv
```

Clone congress

```
git clone https://github.com/openstack/congress.git
cd congress
git checkout stable/liberty
```

Create virtualenv

```
virtualenv ~/congress
source bin/activate
```

Setup Congress

```
sudo mkdir -p /etc/congress
sudo mkdir -p /etc/congress/snapshot
sudo mkdir /var/log/congress
sudo chown ubuntu /var/log/congress
sudo cp etc/api-paste.ini /etc/congress
sudo cp etc/policy.json /etc/congress
```

Install requirements.txt and tox dependencies

The need for this stepo was detected by errors during "tox -egenconfig".

```
sudo apt-get install libffi-dev -y
sudo apt-get install openssl -y
sudo apt-get install libssl-dev -y
```

Install dependencies in virtualenv

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

Install tox

```
pip install tox
```

Generate congress.conf.sample

```
tox -egenconfig
```

Edit congress.conf.sample as needed

```
sed -i -- 's/#verbose = true/verbose = true/g' etc/congress.conf.sample
sed -i -- 's/#log_file = <None>/log_file = congress.log/g' \
etc/congress.conf.sample
sed -i -- 's/#log_dir = <None>/log_dir = \/var\/log\/congress/g' \
etc/congress.conf.sample
```

```
sed -i -- 's/#bind_host = 0.0.0.0/bind_host = 192.168.10.117/g' \
etc/congress.conf.sample
sed -i -- 's/#policy_path = <None>/policy_path = \
\/etc\/congress\/snapshot/g' etc/congress.conf.sample
sed -i -- 's/#auth_strategy = keystone/auth_strategy = noauth/g' \
etc/congress.conf.sample
sed -i -- 's/#drivers =/drivers =\
congress.datasources.neutronv2_driver.NeutronV2Driver,\
congress.datasources.glancev2_driver.GlanceV2Driver,\
congress.datasources.nova_driver.NovaDriver,\
congress.datasources.keystone_driver.KeystoneDriver,\
congress.datasources.ceilometer_driver.CeilometerDriver,\
congress.datasources.cinder_driver.CinderDriver/g' etc/congress.conf.sample
sed -i -- 's/#auth_host = 127.0.0.1/auth_host = 192.168.10.108/g' \
etc/congress.conf.sample
sed -i -- 's/#auth_port = 35357/auth_port = 35357/g' etc/congress.conf.sample
sed -i -- 's/#auth_protocol = https/auth_protocol = http/g' \
etc/congress.conf.sample
sed -i -- 's/#admin_tenant_name = admin/admin_tenant_name = admin/g' \
etc/congress.conf.sample
sed -i -- 's/#admin_user = <None>/admin_user = congress/g' \
etc/congress.conf.sample
sed -i -- 's/#admin_password = <None>/admin_password = congress/g' \
etc/congress.conf.sample
sed -i -- 's/#connection = <None>/connection = mysql:\/\/ubuntu:\
<mysql password>@localhost:3306\/congress/g' etc/congress.conf.sample
```

Copy congress.conf.sample to /etc/congress

```
sudo cp etc/congress.conf.sample /etc/congress/congress.conf
```

Create congress database

```
sudo mysql -u root -p
CREATE DATABASE congress;
GRANT ALL PRIVILEGES ON congress.* TO 'ubuntu'@'localhost' \
IDENTIFIED BY '<mysql password>';
GRANT ALL PRIVILEGES ON congress.* TO 'ubuntu'@'%' IDENTIFIED \
BY '<mysql password>';
exit
```

Install congress-db-manage dependencies

The need for this step was detected by errors in subsequent steps.

```
sudo apt-get build-dep python-mysqldb -y
pip install MySQL-python
```

Create database schema

```
congress-db-manage --config-file /etc/congress/congress.conf upgrade head
```

Install dependencies of OpenStack, Congress, Keystone client operations

```
pip install python-openstackclient
pip install python-congressclient
pip install python-keystoneclient
```

Execute admin-openrc.sh as downloaded from Horizon

```
source ~/admin-openrc.sh
```

Setup Congress user

TODO: needs update in **‘Congress intro in readthedocs < http://congress.readthedocs.org/en/latest/readme.html#installing-congress>‘_**.

```
pip install cliff --upgrade
export ADMIN_ROLE=$(openstack role list | \
awk "/ Admin / { print \$2 }")
export SERVICE_TENANT=$(openstack project list | \
awk "/ admin / { print \$2 }")
openstack user create --password congress --project admin \
--email "congress@example.com" congress
export CONGRESS_USER=$(openstack user list | \
awk "/ congress / { print \$2 }")
openstack role add $ADMIN_ROLE --user $CONGRESS_USER \
--project $SERVICE_TENANT
```

Create Congress service

```
openstack service create congress --type "policy" \
--description "Congress Service"
export CONGRESS_SERVICE=$(openstack service list | \
awk "/ congress / { print \$2 }")
```

Create Congress endpoint

```
openstack endpoint create $CONGRESS_SERVICE \
--region $OS_REGION_NAME \
--publicurl http://$CONGRESS_HOST:1789/ \
--adminurl http://$CONGRESS_HOST:1789/ \
--internalurl http://$CONGRESS_HOST:1789/
```

Start the Congress service in the background

```
bin/congress-server &
# disown the process (so it keeps running if you get disconnected)
disown -h %1
```

Create data sources

To remove datasources: openstack congress datasource delete <name>

It's probably good to do these commands in a new terminal tab, as the congress server log from the last command will be flooding your original terminal screen.

```
openstack congress datasource create nova "nova" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
openstack congress datasource create neutronv2 "neutronv2" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
openstack congress datasource create ceilometer "ceilometer" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
openstack congress datasource create cinder "cinder" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
openstack congress datasource create glancev2 "glancev2" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
openstack congress datasource create keystone "keystone" \
--config username=$OS_USERNAME \
--config tenant_name=$OS_TENANT_NAME \
--config password=$OS_PASSWORD \
--config auth_url=http://$KEYSTONE_HOST:5000/v2.0
```

Run Congress Tempest Tests

```
tox -epy27
```

Restarting after server power loss etc

Currently this install procedure is manual. Automated install and restoral after host recovery is TBD. For now, this procedure will get the Congress service running again.

```
# On jumphost, SSH to Congress server
sudo juju ssh ubuntu@192.168.10.117
# If that fails
  # On jumphost, SSH to controller node
  sudo juju ssh ubuntu@192.168.10.119
  # Start the Congress container
  sudo lxc-start -n juju-trusty-congress -d
  # Verify the Congress container status
  sudo lxc-ls -f juju-trusty-congress
  NAME                   STATE    IPV4            IPV6  GROUPS  AUTOSTART
  ----------------------------------------------------------------------
  juju-trusty-congress  RUNNING  192.168.10.117  -     -       NO
  # exit back to the Jumphost, wait a minute, and go back to the \
  "SSH to Congress server" step above
# On the Congress server that you have logged into
source ~/admin-openrc.sh
cd congress
source bin/activate
bin/congress-server &
disown -h  %1
```

4.2 Doctor Configuration

4.2.1 Doctor Inspector

Doctor Inspector is suggested to be placed in one of the controller nodes, but it can be put on any host where Doctor Monitor can reach and accessible to the OpenStack Controller (Nova).

Make sure OpenStack env parameters are set properly, so that Doctor Inspector can issue admin actions such as compute host force-down and state update of VM.

Then, you can configure Doctor Inspector as follows:

```
git clone https://gerrit.opnfv.org/gerrit/doctor -b stable/brahmaputra
cd doctor/tests
INSPECTOR_PORT=12345
python inspector.py $INSPECTOR_PORT > inspector.log 2>&1 &
```

4.2.2 Doctor Monitor

Doctor Monitors are suggested to be placed in one of the controller nodes, but those can be put on any host which is reachable to target compute host and accessible to the Doctor Inspector. You need to configure Monitors for all compute hosts one by one.

Make sure OpenStack env parameters are set properly, so that Doctor Inspector can issue admin actions such as compute host force-down and state update of VM.

Then, you can configure Doctor Monitor as follows:

```
git clone https://gerrit.opnfv.org/gerrit/doctor -b stable/brahmaputra
cd doctor/tests
INSPECTOR_PORT=12345
COMPUTE_HOST='overcloud-novacompute-0'
sudo python monitor.py "$COMPUTE_HOST" \
    "http://127.0.0.1:$INSPECTOR_PORT/events" > monitor.log 2>&1 &
```

4.3 IPv6 Configuration - Setting Up a Service VM as an IPv6 vRouter

This section provides instructions to set up a service VM as an IPv6 vRouter using OPNFV Brahmaputra Release installers. The environment may be pure OpenStack option or Open Daylight L2-only option. The deployment model may be HA or non-HA. The infrastructure may be bare metal or virtual environment.

For complete instructions and documentations of setting up service VM as an IPv6 vRouter using ANY method, please refer to:

1. IPv6 Configuration Guide (HTML): http://artifacts.opnfv.org/ipv6/docs/setupservicevm/index.html
2. IPv6 User Guide (HTML): http://artifacts.opnfv.org/ipv6/docs/gapanalysis/index.html

4.3.1 Pre-configuration Activities

The configuration will work in 2 environments:

1. OpenStack-only environment
2. OpenStack with Open Daylight L2-only environment

Depending on which installer will be used to deploy OPNFV, each environment may be deployed on bare metal or virtualized infrastructure. Each deployment may be HA or non-HA.

Refer to the previous installer configuration chapters, installations guide and release notes.

4.3.2 Setup Manual in OpenStack-Only Environment

If you intend to set up a service VM as an IPv6 vRouter in OpenStack-only environment of OPNFV Brahmaputra Release, please **NOTE** that:

- Because the anti-spoofing rules of Security Group feature in OpenStack prevents a VM from forwarding packets, we need to disable Security Group feature in the OpenStack-only environment.
- The hostnames, IP addresses, and username are for exemplary purpose in instructions. Please change as needed to fit your environment.
- The instructions apply to both deployment model of single controller node and HA (High Availability) deployment model where multiple controller nodes are used.

Install OPNFV and Preparation

OPNFV-NATIVE-INSTALL-1: To install OpenStack-only environment of OPNFV Brahmaputra Release:

Apex Installer:

```
# HA deployment in OpenStack-only environment
./opnfv-deploy -d /etc/opnfv/os-nosdn-nofeature-ha.yaml

# Non-HA deployment in OpenStack-only environment
# Non-HA deployment is currently not supported by Apex installer.
```

Compass Installer:

```
# HA deployment in OpenStack-only environment
export ISO_URL=file://$BUILD_DIRECTORY/compass.iso
export OS_VERSION=${COMPASS_OS_VERSION}
export OPENSTACK_VERSION=${COMPASS_OPENSTACK_VERSION}
export CONFDIR=$WORKSPACE/deploy/conf/vm_environment
./deploy.sh --dha $CONFDIR/os-nosdn-nofeature-ha.yml \
--network $CONFDIR/$NODE_NAME/network.yml

# Non-HA deployment in OpenStack-only environment
# Non-HA deployment is currently not supported by Compass installer
```

Fuel Installer:

```
# HA deployment in OpenStack-only environment
./deploy.sh -s os-nosdn-nofeature-ha

# Non-HA deployment in OpenStack-only environment
./deploy.sh -s os-nosdn-nofeature-noha
```

Joid Installer:

```
# HA deployment in OpenStack-only environment
./deploy.sh -o liberty -s nosdn -t ha

# Non-HA deployment in OpenStack-only environment
./deploy.sh -o liberty -s nosdn -t nonha
```

Please **NOTE** that you need to refer to installer's documentation for other necessary parameters applicable to your deployment.

OPNFV-NATIVE-INSTALL-2: Clone the following GitHub repository to get the configuration and metadata files

```
git clone https://github.com/sridhargaddam/opnfv_os_ipv6_poc.git \
/opt/stack/opnfv_os_ipv6_poc
```

Disable Security Groups in OpenStack ML2 Setup

OPNFV-NATIVE-SEC-1: Change the settings in `/etc/neutron/plugins/ml2/ml2_conf.ini` as follows

```
# /etc/neutron/plugins/ml2/ml2_conf.ini
[securitygroup]
extension_drivers = port_security
enable_security_group = False
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

OPNFV-NATIVE-SEC-2: Change the settings in `/etc/nova/nova.conf` as follows

```
# /etc/nova/nova.conf
[DEFAULT]
security_group_api = nova
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

OPNFV-NATIVE-SEC-3: After updating the settings, you will have to restart the `Neutron` and `Nova` services.

Please note that the commands of restarting `Neutron` and `Nova` would vary depending on the installer. Please refer to relevant documentation of specific installers

Set Up Service VM as IPv6 vRouter

OPNFV-NATIVE-SETUP-1: Now we assume that OpenStack multi-node setup is up and running. We have to source the tenant credentials in this step. Please **NOTE** that the method of sourcing tenant credentials may vary depending on installers. For example:

Apex installer:

```
# source the tenant credentials using Apex installer of OPNFV
# you need to copy the file /home/stack/overcloudrc from the installer VM called "instack"
# to a location in controller node, for example, in the directory /opt
source /opt/overcloudrc
```

Compass installer:

```
# source the tenant credentials using Compass installer of OPNFV
source /opt/admin-openrc.sh
```

Joid installer:

```
# source the tenant credentials using Joid installer of OPNFV
source $HOME/joid_config/admin-openrc
```

devstack:

```
# source the tenant credentials in devstack
source openrc admin demo
```

Please refer to relevant documentation of installers if you encounter any issue.

OPNFV-NATIVE-SETUP-2: Download `fedora22` image which would be used for `vRouter`

```
wget https://download.fedoraproject.org/pub/fedora/linux/releases/22/Cloud/x86_64/\
Images/Fedora-Cloud-Base-22-20150521.x86_64.qcow2
```

OPNFV-NATIVE-SETUP-3: Import Fedora22 image to `glance`

```
glance image-create --name 'Fedora22' --disk-format qcow2 --container-format bare \
--file ./Fedora-Cloud-Base-22-20150521.x86_64.qcow2
```

OPNFV-NATIVE-SETUP-4: This step is Informational. OPNFV Installer has taken care of this step during deployment. You may refer to this step only if there is any issue, or if you are using other installers.

We have to move the physical interface (i.e. the public network interface) to `br-ex`, including moving the public IP address and setting up default route. Please refer to `OS-NATIVE-SETUP-4` and `OS-NATIVE-SETUP-5` in our more complete instruction.

OPNFV-NATIVE-SETUP-5: Create Neutron routers `ipv4-router` and `ipv6-router` which need to provide external connectivity.

```
neutron router-create ipv4-router
neutron router-create ipv6-router
```

OPNFV-NATIVE-SETUP-6: Create an external network/subnet `ext-net` using the appropriate values based on the data-center physical network setup.

Please **NOTE** that you may only need to create the subnet of `ext-net` because OPNFV installers should have created an external network during installation. You must use the same name of external network that installer creates when you create the subnet. For example:

- **Apex** installer: `external`
- **Compass** installer: `ext-net`
- **Fuel** installer: `net04_ext`
- **Joid** installer: `ext-net`

Please refer to the documentation of installers if there is any issue

```
# This is needed only if installer does not create an external work
# Otherwise, skip this command "net-create"
neutron net-create --router:external ext-net

# Note that the name "ext-net" may work for some installers such as Compass and Joid
# Change the name "ext-net" to match the name of external network that an installer creates
```

```
neutron subnet-create --disable-dhcp --allocation-pool start=198.59.156.251,\
end=198.59.156.254 --gateway 198.59.156.1 ext-net 198.59.156.0/24
```

OPNFV-NATIVE-SETUP-7: Create Neutron networks `ipv4-int-network1` and `ipv6-int-network2` with port_security disabled

```
neutron net-create --port_security_enabled=False ipv4-int-network1
neutron net-create --port_security_enabled=False ipv6-int-network2
```

OPNFV-NATIVE-SETUP-8: Create IPv4 subnet `ipv4-int-subnet1` in the internal network `ipv4-int-network1`, and associate it to `ipv4-router`.

```
neutron subnet-create --name ipv4-int-subnet1 --dns-nameserver 8.8.8.8 \
ipv4-int-network1 20.0.0.0/24

neutron router-interface-add ipv4-router ipv4-int-subnet1
```

OPNFV-NATIVE-SETUP-9: Associate the `ext-net` to the Neutron routers `ipv4-router` and `ipv6-router`.

```
# Note that the name "ext-net" may work for some installers such as Compass and Joid
# Change the name "ext-net" to match the name of external network that an installer creates
neutron router-gateway-set ipv4-router ext-net
neutron router-gateway-set ipv6-router ext-net
```

OPNFV-NATIVE-SETUP-10: Create two subnets, one IPv4 subnet `ipv4-int-subnet2` and one IPv6 subnet `ipv6-int-subnet2` in `ipv6-int-network2`, and associate both subnets to `ipv6-router`

```
neutron subnet-create --name ipv4-int-subnet2 --dns-nameserver 8.8.8.8 \
ipv6-int-network2 10.0.0.0/24

neutron subnet-create --name ipv6-int-subnet2 --ip-version 6 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac ipv6-int-network2 2001:db8:0:1::/64

neutron router-interface-add ipv6-router ipv4-int-subnet2
neutron router-interface-add ipv6-router ipv6-int-subnet2
```

OPNFV-NATIVE-SETUP-11: Create a keypair

```
nova keypair-add vRouterKey > ~/vRouterKey
```

OPNFV-NATIVE-SETUP-12: Create ports for vRouter (with some specific MAC address - basically for automation - to know the IPv6 addresses that would be assigned to the port).

```
neutron port-create --name eth0-vRouter --mac-address fa:16:3e:11:11:11 ipv6-int-network2
neutron port-create --name eth1-vRouter --mac-address fa:16:3e:22:22:22 ipv4-int-network1
```

OPNFV-NATIVE-SETUP-13: Create ports for VM1 and VM2.

```
neutron port-create --name eth0-VM1 --mac-address fa:16:3e:33:33:33 ipv4-int-network1
neutron port-create --name eth0-VM2 --mac-address fa:16:3e:44:44:44 ipv4-int-network1
```

OPNFV-NATIVE-SETUP-14: Update `ipv6-router` with routing information to subnet `2001:db8:0:2::/64`

```
neutron router-update ipv6-router --routes type=dict list=true \
destination=2001:db8:0:2::/64,nexthop=2001:db8:0:1:f816:3eff:fe11:1111
```

OPNFV-NATIVE-SETUP-15: Boot Service VM (`vRouter`), VM1 and VM2

```
nova boot --image Fedora22 --flavor m1.small \
--user-data /opt/stack/opnfv_os_ipv6_poc/metadata.txt \
--availability-zone nova:opnfv-os-compute \
--nic port-id=$(neutron port-list | grep -w eth0-vRouter | awk '{print $2}') \
--nic port-id=$(neutron port-list | grep -w eth1-vRouter | awk '{print $2}') \
--key-name vRouterKey vRouter

nova list

# Please wait for some 10 to 15 minutes so that necessary packages (like radvd)
# are installed and vRouter is up.
nova console-log vRouter

nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.tiny \
--user-data /opt/stack/opnfv_os_ipv6_poc/set_mtu.sh \
--availability-zone nova:opnfv-os-controller \
--nic port-id=$(neutron port-list | grep -w eth0-VM1 | awk '{print $2}') \
--key-name vRouterKey VM1

nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.tiny
--user-data /opt/stack/opnfv_os_ipv6_poc/set_mtu.sh \
--availability-zone nova:opnfv-os-compute \
--nic port-id=$(neutron port-list | grep -w eth0-VM2 | awk '{print $2}') \
--key-name vRouterKey VM2

nova list # Verify that all the VMs are in ACTIVE state.
```

OPNFV-NATIVE-SETUP-16: If all goes well, the IPv6 addresses assigned to the VMs would be as shown as follows:

```
# vRouter eth0 interface would have the following IPv6 address:
#     2001:db8:0:1:f816:3eff:fe11:1111/64
# vRouter eth1 interface would have the following IPv6 address:
#     2001:db8:0:2::1/64
# VM1 would have the following IPv6 address:
#     2001:db8:0:2:f816:3eff:fe33:3333/64
# VM2 would have the following IPv6 address:
#     2001:db8:0:2:f816:3eff:fe44:4444/64
```

OPNFV-NATIVE-SETUP-17: Now we can `SSH` to VMs. You can execute the following command.

```
# 1. Create a floatingip and associate it with VM1, VM2 and vRouter (to the port id that is passed).
#    Note that the name "ext-net" may work for some installers such as Compass and Joid
#    Change the name "ext-net" to match the name of external network that an installer creates
neutron floatingip-create --port-id $(neutron port-list | grep -w eth0-VM1 | \
awk '{print $2}') ext-net
neutron floatingip-create --port-id $(neutron port-list | grep -w eth0-VM2 | \
awk '{print $2}') ext-net
neutron floatingip-create --port-id $(neutron port-list | grep -w eth1-vRouter | \
awk '{print $2}') ext-net

# 2. To know / display the floatingip associated with VM1, VM2 and vRouter.
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth0-VM1 | awk '{print $2}') | awk '{print $2}'
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth0-VM2 | awk '{print $2}') | awk '{print $2}'
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth1-vRouter | awk '{print $2}') | awk '{print $2}'

# 3. To ssh to the vRouter, VM1 and VM2, user can execute the following command.
ssh -i ~/vRouterKey fedora@<floating-ip-of-vRouter>
ssh -i ~/vRouterKey cirros@<floating-ip-of-VM1>
ssh -i ~/vRouterKey cirros@<floating-ip-of-VM2>
```

4.3.3 Setup Manual in OpenStack with Open Daylight L2-Only Environment

If you intend to set up a service VM as an IPv6 vRouter in an environment of OpenStack and Open Daylight L2-only of OPNFV Brahmaputra Release, please **NOTE** that:

- The hostnames, IP addresses, and username are for exemplary purpose in instructions. Please change as needed to fit your environment.
- The instructions apply to both deployment model of single controller node and HA (High Availability) deployment model where multiple controller nodes are used.
- However, in case of HA, when `ipv6-router` is created in step **SETUP-SVM-11**, it could be created in any of the controller node. Thus you need to identify in which controller node `ipv6-router` is created in order to manually spawn `radvd` daemon inside the `ipv6-router` namespace in steps **SETUP-SVM-24** through **SETUP-SVM-30**.

Install OPNFV and Preparation

OPNFV-INSTALL-1: To install OpenStack with Open Daylight L2-only environment of OPNFV Brahmaputra Release:

Apex Installer:

```
# HA deployment in OpenStack with Open Daylight L2-only environment
./opnfv-deploy -d /etc/opnfv/os-odl_l2-nofeature-ha.yaml

# Non-HA deployment in OpenStack with Open Daylight L2-only environment
# Non-HA deployment is currently not supported by Apex installer.
```

Compass Installer:

```
# HA deployment in OpenStack with Open Daylight L2-only environment
export ISO_URL=file://$BUILD_DIRECTORY/compass.iso
export OS_VERSION=${COMPASS_OS_VERSION}
export OPENSTACK_VERSION=${COMPASS_OPENSTACK_VERSION}
export CONFDIR=$WORKSPACE/deploy/conf/vm_environment
./deploy.sh --dha $CONFDIR/os-odl_l2-nofeature-ha.yml \
--network $CONFDIR/$NODE_NAME/network.yml

# Non-HA deployment in OpenStack with Open Daylight L2-only environment
# Non-HA deployment is currently not supported by Compass installer
```

Fuel Installer:

```
# HA deployment in OpenStack with Open Daylight L2-only environment
./deploy.sh -s os-odl_l2-nofeature-ha

# Non-HA deployment in OpenStack with Open Daylight L2-only environment
./deploy.sh -s os-odl_l2-nofeature-noha
```

Joid Installer:

```
# HA deployment in OpenStack with Open Daylight L2-only environment
./deploy.sh -o liberty -s odl -t ha
```

```
# Non-HA deployment in OpenStack with Open Daylight L2-only environment
./deploy.sh -o liberty -s odl -t nonha
```

Please **NOTE** that you need to refer to installer's documentation for other necessary parameters applicable to your deployment.

OPNFV-INSTALL-2: Clone the following GitHub repository to get the configuration and metadata files

```
git clone https://github.com/sridhargaddam/opnfv_os_ipv6_poc.git \
/opt/stack/opnfv_os_ipv6_poc
```

Disable Security Groups in OpenStack ML2 Setup

Please **NOTE** that although Security Groups feature has been disabled automatically through `local.conf` configuration file by some installers such as `devstack`, it is very likely that other installers such as `Apex`, `Compass`, `Fuel` or `Joid` will enable Security Groups feature after installation.

Please make sure that Security Groups are disabled in the setup

OPNFV-SEC-1: Change the settings in `/etc/neutron/plugins/ml2/ml2_conf.ini` as follows

```
# /etc/neutron/plugins/ml2/ml2_conf.ini
[securitygroup]
enable_security_group = False
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

OPNFV-SEC-2: Change the settings in `/etc/nova/nova.conf` as follows

```
# /etc/nova/nova.conf
[DEFAULT]
security_group_api = nova
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

OPNFV-SEC-3: After updating the settings, you will have to restart the `Neutron` and `Nova` services.

Please note that the commands of restarting `Neutron` and `Nova` would vary depending on the installer. Please refer to relevant documentation of specific installers

Source the Credentials in OpenStack Controller Node

SETUP-SVM-1: Login in OpenStack Controller Node. Start a new terminal, and change directory to where OpenStack is installed.

SETUP-SVM-2: We have to source the tenant credentials in this step. Please **NOTE** that the method of sourcing tenant credentials may vary depending on installers. For example:

Apex installer:

```
# source the tenant credentials using Apex installer of OPNFV
# you need to copy the file /home/stack/overcloudrc from the installer VM called "instack"
# to a location in controller node, for example, in the directory /opt
source /opt/overcloudrc
```

Compass installer:

```
# source the tenant credentials using Compass installer of OPNFV
source /opt/admin-openrc.sh
```

Joid installer:

```
# source the tenant credentials using Joid installer of OPNFV
source $HOME/joid_config/admin-openrc
```

devstack:

```
# source the tenant credentials in devstack
source openrc admin demo
```

Please refer to relevant documentation of installers if you encounter any issue.

Informational Note: Move Public Network from Physical Network Interface to `br-ex`

SETUP-SVM-3: Move the physical interface (i.e. the public network interface) to `br-ex`

SETUP-SVM-4: Verify setup of `br-ex`

Those 2 steps are Informational. OPNFV Installer has taken care of those 2 steps during deployment. You may refer to this step only if there is any issue, or if you are using other installers.

We have to move the physical interface (i.e. the public network interface) to `br-ex`, including moving the public IP address and setting up default route. Please refer to `SETUP-SVM-3` and `SETUP-SVM-4` in our more complete instruction.

Create IPv4 Subnet and Router with External Connectivity

SETUP-SVM-5: Create a Neutron router `ipv4-router` which needs to provide external connectivity.

```
neutron router-create ipv4-router
```

SETUP-SVM-6: Create an external network/subnet `ext-net` using the appropriate values based on the data-center physical network setup.

Please **NOTE** that you may only need to create the subnet of `ext-net` because OPNFV installers should have created an external network during installation. You must use the same name of external network that installer creates when you create the subnet. For example:

- **Apex** installer: `external`
- **Compass** installer: `ext-net`
- **Fuel** installer: `net04_ext`
- **Joid** installer: `ext-net`

Please refer to the documentation of installers if there is any issue

```
# This is needed only if installer does not create an external work
# Otherwise, skip this command "net-create"
neutron net-create --router:external ext-net

# Note that the name "ext-net" may work for some installers such as Compass and Joid
# Change the name "ext-net" to match the name of external network that an installer creates
neutron subnet-create --disable-dhcp --allocation-pool start=198.59.156.251,\
end=198.59.156.254 --gateway 198.59.156.1 ext-net 198.59.156.0/24
```

Please note that the IP addresses in the command above are for exemplary purpose. **Please replace the IP addresses of your actual network**.

SETUP-SVM-7: Associate the `ext-net` to the Neutron router `ipv4-router`.

```
# Note that the name "ext-net" may work for some installers such as Compass and Joid
# Change the name "ext-net" to match the name of external network that an installer creates
neutron router-gateway-set ipv4-router ext-net
```

SETUP-SVM-8: Create an internal/tenant IPv4 network `ipv4-int-network1`

```
neutron net-create ipv4-int-network1
```

SETUP-SVM-9: Create an IPv4 subnet `ipv4-int-subnet1` in the internal network `ipv4-int-network1`

```
neutron subnet-create --name ipv4-int-subnet1 --dns-nameserver 8.8.8.8 \
ipv4-int-network1 20.0.0.0/24
```

SETUP-SVM-10: Associate the IPv4 internal subnet `ipv4-int-subnet1` to the Neutron router `ipv4-router`.

```
neutron router-interface-add ipv4-router ipv4-int-subnet1
```

Create IPv6 Subnet and Router with External Connectivity

Now, let us create a second neutron router where we can “manually” spawn a `radvd` daemon to simulate an external IPv6 router.

SETUP-SVM-11: Create a second Neutron router `ipv6-router` which needs to provide external connectivity

```
neutron router-create ipv6-router
```

SETUP-SVM-12: Associate the `ext-net` to the Neutron router `ipv6-router`

```
# Note that the name "ext-net" may work for some installers such as Compass and Joid
# Change the name "ext-net" to match the name of external network that an installer creates
neutron router-gateway-set ipv6-router ext-net
```

SETUP-SVM-13: Create a second internal/tenant IPv4 network `ipv4-int-network2`

```
neutron net-create ipv4-int-network2
```

SETUP-SVM-14: Create an IPv4 subnet `ipv4-int-subnet2` for the `ipv6-router` internal network `ipv4-int-network2`

```
neutron subnet-create --name ipv4-int-subnet2 --dns-nameserver 8.8.8.8 \
ipv4-int-network2 10.0.0.0/24
```

SETUP-SVM-15: Associate the IPv4 internal subnet `ipv4-int-subnet2` to the Neutron router `ipv6-router`.

```
neutron router-interface-add ipv6-router ipv4-int-subnet2
```

Prepare Image, Metadata and Keypair for Service VM

SETUP-SVM-16: Download `fedora22` image which would be used as `vRouter`

```
wget https://download.fedoraproject.org/pub/fedora/linux/releases/22/Cloud/x86_64/\
Images/Fedora-Cloud-Base-22-20150521.x86_64.qcow2

glance image-create --name 'Fedora22' --disk-format qcow2 --container-format bare \
--file ./Fedora-Cloud-Base-22-20150521.x86_64.qcow2
```

SETUP-SVM-17: Create a keypair

```
nova keypair-add vRouterKey > ~/vRouterKey
```

SETUP-SVM-18: Create ports for `vRouter` and both the VMs with some specific MAC addresses.

```
neutron port-create --name eth0-vRouter --mac-address fa:16:3e:11:11:11 ipv4-int-network2
neutron port-create --name eth1-vRouter --mac-address fa:16:3e:22:22:22 ipv4-int-network1
neutron port-create --name eth0-VM1 --mac-address fa:16:3e:33:33:33 ipv4-int-network1
neutron port-create --name eth0-VM2 --mac-address fa:16:3e:44:44:44 ipv4-int-network1
```

Boot Service VM (`vRouter`) with `eth0` on `ipv4-int-network2` and `eth1` on `ipv4-int-network1`

Let us boot the service VM (`vRouter`) with `eth0` interface on `ipv4-int-network2` connecting to `ipv6-router`, and `eth1` interface on `ipv4-int-network1` connecting to `ipv4-router`.

SETUP-SVM-19: Boot the `vRouter` using `Fedora22` image on the OpenStack Compute Node with hostname `opnfv-os-compute`

```
nova boot --image Fedora22 --flavor m1.small \
--user-data /opt/stack/opnfv_os_ipv6_poc/metadata.txt \
--availability-zone nova:opnfv-os-compute \
--nic port-id=$(neutron port-list | grep -w eth0-vRouter | awk '{print $2}') \
--nic port-id=$(neutron port-list | grep -w eth1-vRouter | awk '{print $2}') \
--key-name vRouterKey vRouter
```

Please **note** that `/opt/stack/opnfv_os_ipv6_poc/metadata.txt` is used to enable the `vRouter` to automatically spawn a `radvd`, and

- Act as an IPv6 vRouter which advertises the RA (Router Advertisements) with prefix `2001:db8:0:2::/64` on its internal interface (`eth1`).
- Forward IPv6 traffic from internal interface (`eth1`)

SETUP-SVM-20: Verify that `Fedora22` image boots up successfully and vRouter has `ssh` keys properly injected

```
nova list
nova console-log vRouter
```

Please note that **it may take a few minutes** for the necessary packages to get installed and `ssh` keys to be injected.

```
# Sample Output
[   762.884523] cloud-init[871]: ec2: #############################################################
[   762.909634] cloud-init[871]: ec2: -----BEGIN SSH HOST KEY FINGERPRINTS-----
[   762.931626] cloud-init[871]: ec2: 2048 e3:dc:3d:4a:bc:b6:b0:77:75:a1:70:a3:d0:2a:47:a9   (RSA)
[   762.957380] cloud-init[871]: ec2: -----END SSH HOST KEY FINGERPRINTS-----
[   762.979554] cloud-init[871]: ec2: #############################################################
```

Boot Two Other VMs in `ipv4-int-network1`

In order to verify that the setup is working, let us create two cirros VMs with `eth1` interface on the `ipv4-int-network1`, i.e., connecting to `vRouter` `eth1` interface for internal network.

We will have to configure appropriate `mtu` on the VMs' interface by taking into account the tunneling overhead and any physical switch requirements. If so, push the `mtu` to the VM either using `dhcp` options or via `meta-data`.

SETUP-SVM-21: Create VM1 on OpenStack Controller Node with hostname `opnfv-os-controller`

```
nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.tiny \
--user-data /opt/stack/opnfv_os_ipv6_poc/set_mtu.sh \
--availability-zone nova:opnfv-os-controller \
--nic port-id=$(neutron port-list | grep -w eth0-VM1 | awk '{print $2}') \
--key-name vRouterKey VM1
```

SETUP-SVM-22: Create VM2 on OpenStack Compute Node with hostname `opnfv-os-compute`

```
nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.tiny \
--user-data /opt/stack/opnfv_os_ipv6_poc/set_mtu.sh \
--availability-zone nova:opnfv-os-compute \
--nic port-id=$(neutron port-list | grep -w eth0-VM2 | awk '{print $2}') \
--key-name vRouterKey VM2
```

SETUP-SVM-23: Confirm that both the VMs are successfully booted.

```
nova list
nova console-log VM1
nova console-log VM2
```

Spawn `RADVD` in `ipv6-router`

Let us manually spawn a `radvd` daemon inside `ipv6-router` namespace to simulate an external router. First of all, we will have to identify the `ipv6-router` namespace and move to the namespace.

Please **NOTE** that in case of HA (High Availability) deployment model where multiple controller nodes are used, `ipv6-router` created in step **SETUP-SVM-11** could be in any of the controller node. Thus you need to identify in which controller node `ipv6-router` is created in order to manually spawn `radvd` daemon inside the `ipv6-router` namespace in steps **SETUP-SVM-24** through **SETUP-SVM-30**. The following command in Neutron will display the controller on which the `ipv6-router` is spawned.

```
neutron l3-agent-list-hosting-router ipv6-router
```

Then you login to that controller and execute steps **SETUP-SVM-24** through **SETUP-SVM-30**

SETUP-SVM-24: identify the `ipv6-router` namespace and move to the namespace

```
sudo ip netns exec qrouter-$(neutron router-list | grep -w ipv6-router | \
awk '{print $2}') bash
```

SETUP-SVM-25: Upon successful execution of the above command, you will be in the router namespace. Now let us configure the IPv6 address on the <qr-xxx> interface.

```
export router_interface=$(ip a s | grep -w "global qr-*" | awk '{print $7}')
ip -6 addr add 2001:db8:0:1::1 dev $router_interface
```

SETUP-SVM-26: Update the sample file `/opt/stack/opnfv_os_ipv6_poc/scenario2/radvd.conf` with `$router_interface`.

```
cp /opt/stack/opnfv_os_ipv6_poc/scenario2/radvd.conf /tmp/radvd.$router_interface.conf
sed -i 's/$router_interface/'$router_interface'/g' /tmp/radvd.$router_interface.conf
```

SETUP-SVM-27: Spawn a `radvd` daemon to simulate an external router. This `radvd` daemon advertises an IPv6 subnet prefix of `2001:db8:0:1::/64` using RA (Router Advertisement) on its $router_interface so that `eth0` interface of `vRouter` automatically configures an IPv6 SLAAC address.

```
$radvd -C /tmp/radvd.$router_interface.conf -p /tmp/br-ex.pid.radvd -m syslog
```

SETUP-SVM-28: Add an IPv6 downstream route pointing to the `eth0` interface of vRouter.

```
ip -6 route add 2001:db8:0:2::/64 via 2001:db8:0:1:f816:3eff:fe11:1111
```

SETUP-SVM-29: The routing table should now look similar to something shown below.

```
ip -6 route show
2001:db8:0:1::1 dev qr-42968b9e-62 proto kernel metric 256
2001:db8:0:1::/64 dev qr-42968b9e-62 proto kernel metric 256 expires 86384sec
2001:db8:0:2::/64 via 2001:db8:0:1:f816:3eff:fe11:1111 dev qr-42968b9e-62 proto ra metric 1024 expire
fe80::/64 dev qg-3736e0c7-7c proto kernel metric 256
fe80::/64 dev qr-42968b9e-62 proto kernel metric 256
```

SETUP-SVM-30: If all goes well, the IPv6 addresses assigned to the VMs would be as shown as follows:

```
# vRouter eth0 interface would have the following IPv6 address:
#     2001:db8:0:1:f816:3eff:fe11:1111/64
# vRouter eth1 interface would have the following IPv6 address:
#     2001:db8:0:2::1/64
# VM1 would have the following IPv6 address:
#     2001:db8:0:2:f816:3eff:fe33:3333/64
# VM2 would have the following IPv6 address:
#     2001:db8:0:2:f816:3eff:fe44:4444/64
```

Testing to Verify Setup Complete

Now, let us `SSH` to those VMs, e.g. VM1 and / or VM2 and / or vRouter, to confirm that it has successfully configured the IPv6 address using `SLAAC` with prefix `2001:db8:0:2::/64` from `vRouter`.

We use `floatingip` mechanism to achieve `SSH`.

SETUP-SVM-31: Now we can `SSH` to VMs. You can execute the following command.

```
# 1. Create a floatingip and associate it with VM1, VM2 and vRouter (to the port id that is passed).
#    Note that the name "ext-net" may work for some installers such as Compass and Joid
#    Change the name "ext-net" to match the name of external network that an installer creates
neutron floatingip-create --port-id $(neutron port-list | grep -w eth0-VM1 | \
awk '{print $2}') ext-net
neutron floatingip-create --port-id $(neutron port-list | grep -w eth0-VM2 | \
awk '{print $2}') ext-net
neutron floatingip-create --port-id $(neutron port-list | grep -w eth1-vRouter | \
awk '{print $2}') ext-net

# 2. To know / display the floatingip associated with VM1, VM2 and vRouter.
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth0-VM1 | awk '{print $2}') | awk '{print $2}'
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth0-VM2 | awk '{print $2}') | awk '{print $2}'
neutron floatingip-list -F floating_ip_address -F port_id | grep $(neutron port-list | \
grep -w eth1-vRouter | awk '{print $2}') | awk '{print $2}'

# 3. To ssh to the vRouter, VM1 and VM2, user can execute the following command.
ssh -i ~/vRouterKey fedora@<floating-ip-of-vRouter>
ssh -i ~/vRouterKey cirros@<floating-ip-of-VM1>
ssh -i ~/vRouterKey cirros@<floating-ip-of-VM2>
```

If everything goes well, `ssh` will be successful and you will be logged into those VMs. Run some commands to verify that IPv6 addresses are configured on `eth0` interface.

SETUP-SVM-32: Show an IPv6 address with a prefix of `2001:db8:0:2::/64`

```
ip address show
```

SETUP-SVM-33: ping some external IPv6 address, e.g. `ipv6-router`

```
ping6 2001:db8:0:1::1
```

If the above ping6 command succeeds, it implies that `vRouter` was able to successfully forward the IPv6 traffic to reach external `ipv6-router`.

4.4 Promise Feature Configuration Overview

4.4.1 Promise installation

Install nodejs, npm and promise

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
sudo apt-get install -y nodejs
sudo npm -g install npm@latest
git clone https://github.com/opnfv/promise.git
cd promise
npm install
```

Please note that the last command 'npm install' will install all needed dependencies for promise (including yangforge and mocha)

4.4.2 Testing

Please perform the following preparation steps:

1. Set OpenStack environment parameters properly (e.g. source openrc admin demo in DevStack)
2. Create OpenStack tenant (e.g. promise) and tenant user (e.g. promiser)
3. Create a flavor in Nova with 1 vCPU and 512 MB RAM
4. Create a private network, subnet and router in Neutron
5. Create an image in Glance

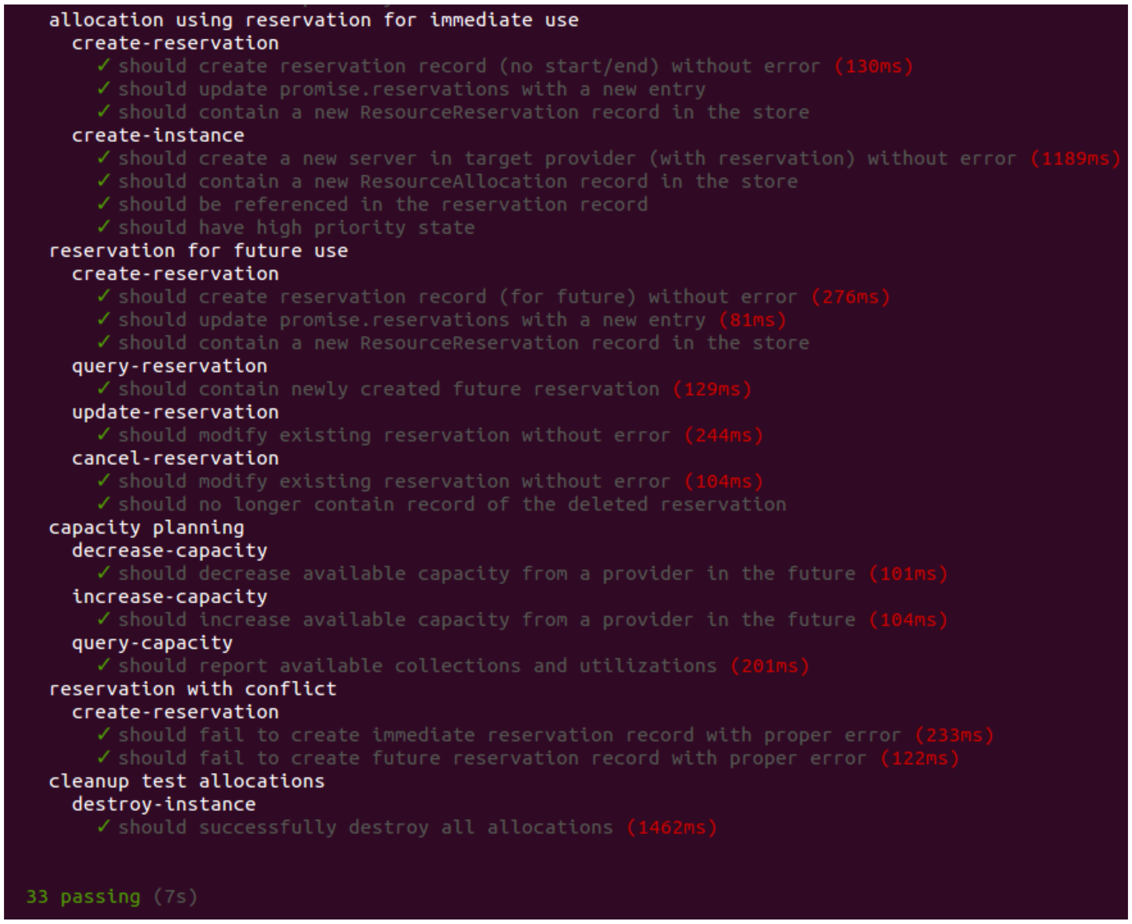
Once done, the promise test script can be invoked as follows (as a single line command):

```
NODE_ENV=mytest \
OS_TENANT_NAME=promise \
OS_USERNAME=promiser \
OS_PASSWORD=<user password from Step 2> \
OS_TEST_FLAVOR=<flavor ID from Step 3> \
OS_TEST_NETWORK=<network ID from Step 4> \
OS_TEST_IMAGE=<image ID from Step 5> \
npm run -s test -- --reporter json > promise-results.json
```

The results of the tests will be stored in the promise-results.json file.

The results can also be seen in the console ("npm run -s test")

All 33 tests passing?! Congratulations, promise has been successfully installed and configured.



```
  allocation using reservation for immediate use
    create-reservation
      ✓ should create reservation record (no start/end) without error (130ms)
      ✓ should update promise.reservations with a new entry
      ✓ should contain a new ResourceReservation record in the store
    create-instance
      ✓ should create a new server in target provider (with reservation) without error (1189ms)
      ✓ should contain a new ResourceAllocation record in the store
      ✓ should be referenced in the reservation record
      ✓ should have high priority state
  reservation for future use
    create-reservation
      ✓ should create reservation record (for future) without error (276ms)
      ✓ should update promise.reservations with a new entry (81ms)
      ✓ should contain a new ResourceReservation record in the store
    query-reservation
      ✓ should contain newly created future reservation (129ms)
    update-reservation
      ✓ should modify existing reservation without error (244ms)
    cancel-reservation
      ✓ should modify existing reservation without error (104ms)
      ✓ should no longer contain record of the deleted reservation
  capacity planning
    decrease-capacity
      ✓ should decrease available capacity from a provider in the future (101ms)
    increase-capacity
      ✓ should increase available capacity from a provider in the future (104ms)
    query-capacity
      ✓ should report available collections and utilizations (201ms)
  reservation with conflict
    create-reservation
      ✓ should fail to create immediate reservation record with proper error (233ms)
      ✓ should fail to create future reservation record with proper error (122ms)
  cleanup test allocations
    destroy-instance
      ✓ should successfully destroy all allocations (1462ms)


33 passing (7s)
```

CHAPTER

FIVE

POST CONFIGURATION ACTIVITIES

Once you have deployed and configured your scenario and features you should validate the state of the system using the following guides.

5.1 Scenario validation activities

The following guides provide information on how to validate the installation of you scenario based on the tools and test suites available for the installation tool you have selected:

5.1.1 IPv6 Post Installation Procedures

Congratulations, you have completed the setup of using a service VM to act as an IPv6 vRouter. You have validated the setup based on the instruction in previous sections. If you want to further test your setup, you can `ping6` among `VM1`, `VM2`, `vRouter` and `ipv6-router`.

This setup allows further open innovation by any 3rd-party. For more instructions and documentations, please refer to:

1. IPv6 Configuration Guide (HTML): http://artifacts.opnfv.org/ipv6/docs/setupservicevm/index.html
2. IPv6 User Guide (HTML): http://artifacts.opnfv.org/ipv6/docs/gapanalysis/index.html

Automated post installation activities

Refer to the relevant testing guides, results, and release notes of Yardstick Project.

5.1.2 JOID post installation procedures

Configure OpenStack

In each SDN directory, for example joid/ci/opencontrail, there is a folder for Juju deployer where you can find the charm bundle yaml files that the deploy.sh uses to deploy.

In the same directory, there is **scripts** folder where you can find shell scripts to help you configure the OpenStack cloud that you just deployed. These scripts are created to help you configure a basic OpenStack Cloud to verify the cloud. For more info on OpenStack Cloud configuration, please refer to the OpenStack Cloud Administrator Guide on docs.openstack.org. Similarly, for complete SDN configuration, please refer to the respective SDN adminstrator guide.

Each SDN solution requires slightly different setup, please refer to the **README** in each SDN folder. Most likely you will need to modify the **openstack.sh** and **cloud-setup.sh** scripts for the floating IP range, private IP network, and SSH keys. Please go through **openstack.sh**, **glance.sh** and **cloud-setup.sh** and make changes as you see fit.

5.2 Feature validation activities

The following sections provide information on how to validate the features you have installed in your scenario:

5.2.1 Copper post installation procedures

This release focused on use of the OpenStack Congress service for managing configuration policy. The Congress install verify procedure described here is largely manual. This procedure, as well as the longer-term goal of automated verification support, is a work in progress. The procedure is further specific to one OPNFV installer (JOID, i.e. MAAS/JuJu) based environment.

Automated post installation activities

No automated procedures are provided at this time.

Copper post configuration procedures

No configuration procedures are required beyond the basic install procedure.

Platform components validation

Following are notes on creating a container as test driver for Congress. This is based upon an Ubuntu host as installed by JOID.

Create and Activate the Container

On the jumphost:

```
sudo lxc-create -n trusty-copper -t /usr/share/lxc/templates/lxc-ubuntu \
-- -b ubuntu ~/opnfv
sudo lxc-start -n trusty-copper -d
sudo lxc-info --name trusty-copper
(typical output)
Name:           trusty-copper
State:          RUNNING
PID:            4563
IP:             10.0.3.44
CPU use:        28.77 seconds
BlkIO use:      522.79 MiB
Memory use:     559.75 MiB
KMem use:       0 bytes
Link:           vethDMFOAN
 TX bytes:      2.62 MiB
 RX bytes:      88.48 MiB
 Total bytes:   91.10 MiB
```

Login and configure the test server

```
ssh ubuntu@10.0.3.44
sudo apt-get update
sudo apt-get upgrade -y

# Install pip
sudo apt-get install python-pip -y

# Install java
sudo apt-get install default-jre -y

# Install other dependencies
sudo apt-get install git gcc python-dev libxml2 libxslt1-dev \
libzip-dev php5-curl -y

# Setup OpenStack environment variables per your OPNFV install
export CONGRESS_HOST=192.168.10.117
export KEYSTONE_HOST=192.168.10.108
export CEILOMETER_HOST=192.168.10.105
export CINDER_HOST=192.168.10.101
export GLANCE_HOST=192.168.10.106
export HEAT_HOST=192.168.10.107
export NEUTRON_HOST=192.168.10.111
export NOVA_HOST=192.168.10.112
source ~/admin-openrc.sh

# Install and test OpenStack client
mkdir ~/git
cd git
git clone https://github.com/openstack/python-openstackclient.git
cd python-openstackclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
openstack service list
(typical output)
+----------------------------------+------------+----------------+
| ID                               | Name       | Type           |
+----------------------------------+------------+----------------+
| 2f8799ae50f24c928c021fabf8a50f5f | keystone   | identity       |
| 351b13f56d9a4e25849406ec1d5a2726 | cinder     | volume         |
| 5129510c3143454f9ba8ec7e6735e267 | cinderv2   | volumev2       |
| 5ee1e220460f41dea9be06921400ce9b | congress   | policy         |
| 78e73a7789a14f56a5d248a0cd141201 | quantum    | network        |
| 9d5a00fb475a45b2ae6767528299ed6b | ceilometer | metering       |
| 9e4b1624ef0b434abc0b82f607c5045c | heat       | orchestration  |
| b6c01ceb5023442d9f394b83f2a18e01 | heat-cfn   | cloudformation |
| ba6199e3505045ad87e2a7175bd0c57f | glance     | image          |
| d753f304a0d541dbb989780ae70328a8 | nova       | compute        |
+----------------------------------+------------+----------------+

# Install and test Congress client
cd ~/git
git clone https://github.com/openstack/python-congressclient.git
cd python-congressclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
openstack congress driver list
(typical output)
+------------+--------------------------------------------------------------------------+
| id         | description                                                              |
+------------+--------------------------------------------------------------------------+
| ceilometer | Datasource driver that interfaces with ceilometer.                       |
| neutronv2  | Datasource driver that interfaces with OpenStack Networking aka Neutron. |
| nova       | Datasource driver that interfaces with OpenStack Compute aka nova.       |
| keystone   | Datasource driver that interfaces with keystone.                         |
| cinder     | Datasource driver that interfaces with OpenStack cinder.                 |
| glancev2   | Datasource driver that interfaces with OpenStack Images aka Glance.      |
+------------+--------------------------------------------------------------------------+

# Install and test Glance client
cd ~/git
git clone https://github.com/openstack/python-glanceclient.git
cd python-glanceclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
glance image-list
(typical output)
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 6ce4433e-65c0-4cd8-958d-b06e30c76241 | cirros-0.3.3-x86_64 |
+--------------------------------------+---------------------+

# Install and test Neutron client
cd ~/git
git clone https://github.com/openstack/python-neutronclient.git
cd python-neutronclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
neutron net-list
(typical output)
+--------------------------------------+----------+-------------------------------------------------
| id                                   | name     | subnets
+--------------------------------------+----------+-------------------------------------------------
| dc6227df-af41-439f-bd2c-c2c2f0fe7fc5 | public   | 5745846c-dd79-4900-a7da-bf506348ceac 192.168.10.
| a3f9f13a-5de9-4d3b-98c8-d2e40a2ef8e9 | internal | 5e0be862-90da-44ab-af43-56d5c65aa049 10.0.0.0/24
+--------------------------------------+----------+-------------------------------------------------

# Install and test Nova client
cd ~/git
git clone https://github.com/openstack/python-novaclient.git
cd python-novaclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
nova hypervisor-list
(typical output)
+----+---------------------+-------+---------+
| ID | Hypervisor hostname | State | Status  |
+----+---------------------+-------+---------+
| 1  | compute1.maas       | up    | enabled |
+----+---------------------+-------+---------+

# Install and test Keystone client
cd ~/git
git clone https://github.com/openstack/python-keystoneclient.git
cd python-keystoneclient
git checkout stable/liberty
sudo pip install -r requirements.txt
sudo python setup.py install
```

Setup the Congress Test Webapp

```
# Clone Copper (if not already cloned in user home)
cd ~/git
if [ ! -d ~/git/copper ]; then \
git clone https://gerrit.opnfv.org/gerrit/copper; fi

# Copy the Apache config
sudo cp ~/git/copper/components/congress/test-webapp/www/ubuntu-apache2.conf \
/etc/apache2/apache2.conf

# Point proxy.php to the Congress server per your install
sed -i -- "s/192.168.10.117/$CONGRESS_HOST/g" \
~/git/copper/components/congress/test-webapp/www/html/proxy/index.php

# Copy the webapp to the Apache root directory and fix permissions
sudo cp -R ~/git/copper/components/congress/test-webapp/www/html /var/www
sudo chmod 755 /var/www/html -R

# Make webapp log directory and set permissions
mkdir ~/logs
chmod 777 ~/logs

# Restart Apache
sudo service apache2 restart
```

Using the Test Webapp

Browse to the trusty-copper server IP address.

Interactive options are meant to be self-explanatory given a basic familiarity with the Congress service and data model. But the app will be developed with additional features and UI elements.

5.3 Additional testing and validation activities

Many of our testing tools can be manually installed to facilitate targeted testing of features and capabilities of your scenario. The following guides provide instruction on setting up these testing suites:

5.3.1 Functional testing Installation

Pull the Functest Docker image from the Docker hub:

```
$ docker pull opnfv/functest:brahmaputra.1.0
```

Check that the image is available:

```
$ docker images
```

Run the docker container giving the environment variables:

```
- INSTALLER_TYPE. Possible values are "apex", "compass", "fuel" or "joid".
- INSTALLER_IP. each installer has its installation strategy.
```

Functest may need to know the IP of the installer to retrieve the credentials (e.g. usually "10.20.0.2" for fuel, not neede for joid...).

The minimum command to create the Functest docker file can be described as follows:

```
docker run -it -e "INSTALLER_IP=10.20.0.2" -e "INSTALLER_TYPE=fuel" opnfv/functest:brahmaputra.1.0 /b
```

Optionally, it is possible to precise the container name through the option –name:

```
docker run --name "CONTAINER_NAME" -it -e "INSTALLER_IP=10.20.0.2" -e "INSTALLER_TYPE=fuel" opnfv/fun
```

It is also possible to to indicate the path of the OpenStack creds using -v:

```
docker run  -it -e "INSTALLER_IP=10.20.0.2" -e "INSTALLER_TYPE=fuel" -v <path_to_your_local_creds_fil
```

The local file will be mounted in the container under /home/opnfv/functest/conf/openstack.creds

After the run command the prompt appears which means that we are inside the container and ready to run Functest.

Inside the container, the following directory structure should be in place:

```
`-- home
    `-- opnfv
      |-- functest
      |   |-- conf
      |   |-- data
      |   `-- results
      `-- repos
          |-- bgpvpn
          |-- functest
          |-- odl_integration
          |-- rally
          |-- releng
          `-- vims-test
```

Basically the container includes:

- Functest directory to store the configuration (the OpenStack creds are paste in /home/opngb/functest/conf), the data (images neede for test for offline testing), results (some temporary artifacts may be stored here)
- Repositories: the functest repository will be used to prepare the environment, run the tests. Other repositories are used for the installation of the tooling (e.g. rally) and/or the retrieval of feature projects scenarios (e.g. bgpvpn)

The arborescence under the functest repo can be described as follow:

```
.
  |-- INFO
  |-- LICENSE
  |-- commons
  |   |-- ims
  |   |-- mobile
  |   `-- traffic-profile-guidelines.rst
  |-- docker
  |   |-- Dockerfile
  |   |-- common.sh
  |   |-- prepare_env.sh
  |   |-- requirements.pip
  |   `-- run_tests.sh
  |-- docs
  |   |-- configguide
  |   |-- functest.rst
  |   |-- images
  |   `-- userguide
  `-- testcases
      |-- Controllers
      |-- VIM
      |-- __init__.py
      |-- config_functest.py
      |-- config_functest.yaml
      |-- functest_utils.py
      |-- functest_utils.pyc
      |-- vIMS
      `-- vPing
```

We may distinguish 4 different folders:

- commons: it is a folder dedicated to store traffic profile or any test inputs that could be reused by any test project
- docker: this folder includes the scripts that will be used to setup the environment and run the tests
- docs: this folder includes the user and installation/configuration guide
- testcases: this folder includes the scripts required by Functest internal test cases

Firstly run the script to install functest environment:

```
$ ${repos_dir}/functest/docker/prepare_env.sh
```

NOTE: ${repos_dir} is a default environment variable inside the docker container, which points to /home/opnfv/repos

Run the script to start the tests:

```
$ ${repos_dir}/functest/docker/run_tests.sh
```

5.3.2 Installing vswitchperf

Supported Operating Systems

- CentOS 7
- Fedora 20
- Fedora 21
- Fedora 22
- Ubuntu 14.04

Supported vSwitches

The vSwitch must support Open Flow 1.3 or greater.

- OVS (built from source).
- OVS with DPDK (built from source).

Supported Hypervisors

- Qemu version 2.3.

Available VNFs

A simple VNF that forwards traffic through a VM, using:

- DPDK testpmd
- Linux Brigde
- custom l2fwd module

The VM image can be downloaded from: http://artifacts.opnfv.org/vswitchperf/vloop-vnf-ubuntu-14.04_20151216.qcow2

Other Requirements

The test suite requires Python 3.3 and relies on a number of other packages. These need to be installed for the test suite to function.

Installation of required packages, preparation of Python 3 virtual environment and compilation of OVS, DPDK and QEMU is performed by script **systems/build_base_machine.sh**. It should be executed under user account, which will be used for vsperf execution.

Please Note: Password-less sudo access must be configured for given user account before script is executed.

Execution of installation script:

```
$ cd systems
$ ./build_base_machine.sh
```

Please Note: you don't need to go into any of the systems subdirectories, simply run the top level **build_base_machine.sh**, your OS will be detected automatically.

Script **build_base_machine.sh** will install all the vsperf dependencies in terms of system packages, Python 3.x and required Python modules. In case of CentOS 7 it will install Python 3.3 from an additional repository provided by Software Collections (a link). Installation script will also use virtualenv to create a vsperf virtual environment, which is isolated from the default Python environment. This environment will reside in a directory called **vsperfenv** in $HOME.

You will need to activate the virtual environment every time you start a new shell session. Its activation is specific to your OS:

CentOS 7

```
$ scl enable python33 bash
$ cd $HOME/vsperfenv
$ source bin/activate
```

Fedora and Ubuntu

```
$ cd $HOME/vsperfenv
$ source bin/activate
```

Working Behind a Proxy

If you're behind a proxy, you'll likely want to configure this before running any of the above. For example:

```
    export http_proxy=proxy.mycompany.com:123
    export https_proxy=proxy.mycompany.com:123
```