OPNFV Configuration Guide

Release arno.2015.1.0 (22cdec3)

OPNFV

July 06, 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.

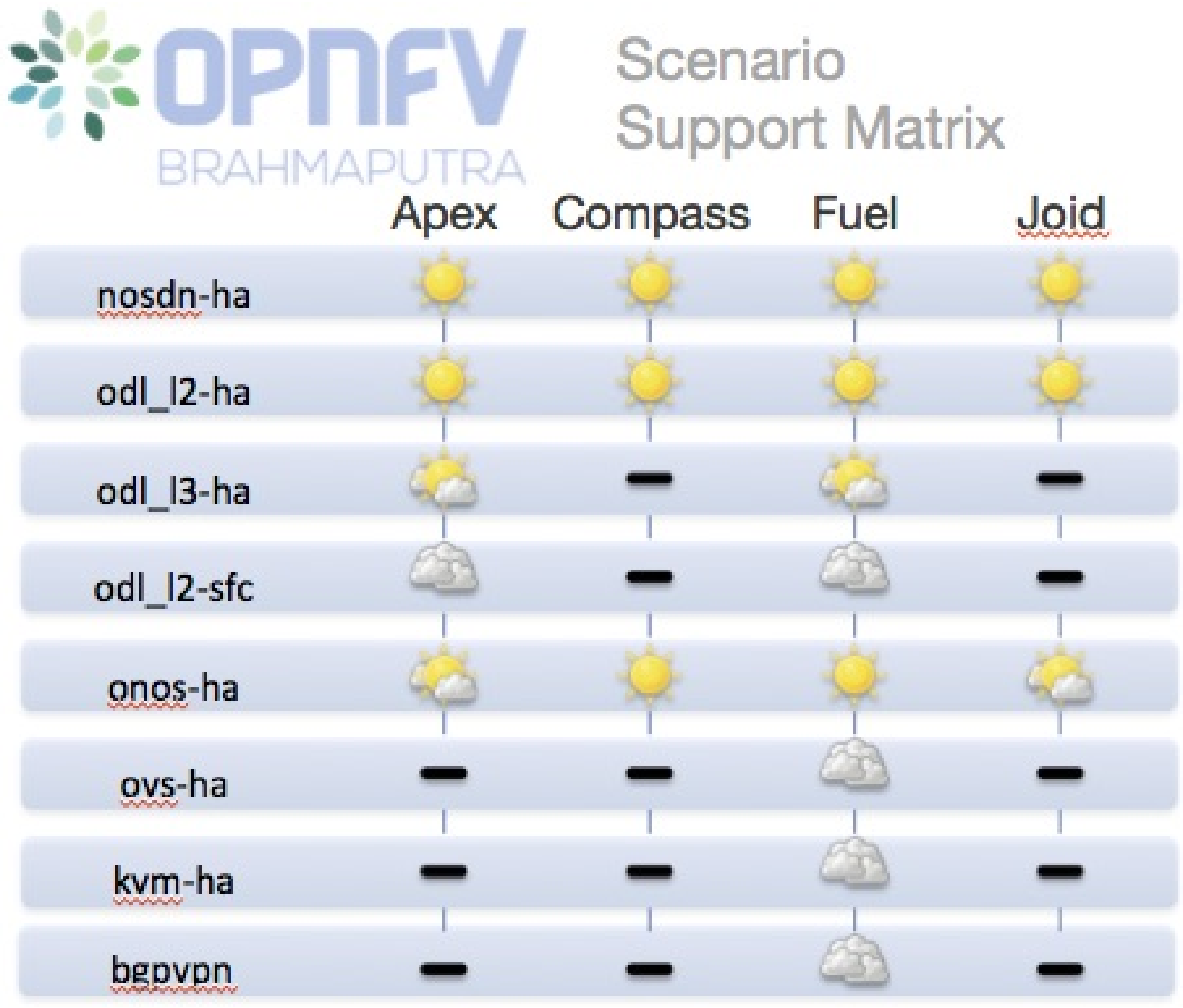
2.1 OPNFV Scenarios

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 Brahmaputra scenario overeview

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


OPNFV
BRAHMAPUTRA

Scenario Support Matrix

	Apex	Compass	Fuel	Joid
nosdn-ha	☀	☀	☀	☀
odl_l2-ha	☀	☀	☀	☀
odl_l3-ha	⛅	—	⛅	—
odl_l2-sfc	☁	—	☁	—
onos-ha	⛅	☀	☀	⛅
ovs-ha	—	—	☁	—
kvm-ha	—	—	☁	—
bgpvpn	—	—	☁	—

Scenario status is indicated by a weather pattern icon. All scenarios listed with a weather pattern are possible to deploy and run in your environment or a Pharos lab, however they may have known limitations or issues as indicated by the icon.

Weather pattern icon legend:

Weather Icon	Scenario Status
☀	Stable, no known issues
⛅	Stable, documented limitations
☁	Deployable, stability or feature limitations
—	Not deployed with this installer

Scenarios that are not yet in a state of "Stable, no known issues" will continue to be stabilised and updates will be made on the stable/brahmaputra branch. While we intend that all Brahmaputra scenarios should be stable it is worth checking regularly to see the current status. Due to our dependency on upstream communities and code some issues may not be resolved prior to the C release.

2.1.2 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, sfc
  - [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-onos-nofeature-ha
  - This is an OpenStack deployment in high availability mode including ONOS as the SDN controller
- os-odl_l2-sfc
  - This is an OpenStack deployment using OpenDaylight and OVS enabled with SFC features

2.1.3 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.

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

Installing the OPNFV platform requires either a physical environment as defined in the Pharos lab specification, or a virtual infrastructure. When configuring a physical infrastructure it is strongly advised to follow the Pharos configuration material.

3.1 Lab Setup Guide

Provides an overview for setting up a Pharos lab. A full set of pharos_master documents are maintained in the *pharos* repo.

When setting up an OPNFV community lab ...

- Provide the Pharos community with details of the intended setup, including ...
  - Overview of resources are being offered to the community, intended purpose and known limitations
  - Lab owner name with contacts
  - Timelines for availablity for development, test, release production, ...
- Update the Pharos Wiki with lab details
  - Lab map, organization, contacts, status, location, resources, role, etc.
  - https://wiki.opnfv.org/pharos#community_labs
  - pharos_wiki
- Update the Pharos project information file "Current Labs"
  - pharos_information
- Create new Wiki pages for lab and POD specific information
  - Access procedures
  - Usage guidelines for developers
  - Update infomtation as PODs are re-assigned or usage/availability changes
- Fill Lab and POD templates ... pharos_lab ... pharos_pod
  - Note that security sensitive lab information should be stored in the secure Pharos repo
- Connect PODs to Jenkins/CI

3.1.1 Jump Server Configuration

Jump server install procedures are maintained by each installer project. Addional Jump server configuraton BKMs will be maintained here. The below install information was used for Fuel however may be outdated (please refer to Fuel Installer documents).

Procedure

1. Obtain CentOS 7 Minimal ISO and install

   ```
   wget http://mirrors.kernel.org/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1503-01.is
   ```

2. Set parameters appropriate for your environment during installation
3. Disable NetworkManager

   ```
   systemctl disable NetworkManager
   ```

4. Configure your /etc/sysconfig/network-scripts/ifcfg-* files for your network
5. Restart networking

   ```
   service network restart
   ```

6. Edit /etc/resolv.conf and add a nameserver

   ```
   vi /etc/resolv.conf
   ```

7. Install libvirt & kvm

   ```
   yum -y update   yum -y install kvm qemu-kvm libvirt   systemctl enable
   libvirtd
   ```

8. Reboot:

   ```
   shutdown -r now
   ```

9. If you wish to avoid annoying delay when use ssh to log in, disable DNS lookups:

   ```
   vi /etc/ssh/sshd_config
   ```

   Uncomment “UseDNS yes”, change ‘yes’ to ‘no’.

   Save

10. Restart sshd

    ```
    systemctl restart sshd
    ```

11. Install virt-install

    ```
    yum -y install virt-install
    ```

12. Visit artifacts.opnfv.org and D/L the OPNFV Fuel ISO
13. Create a bridge using the interface on the PXE network, for example: br0
14. Make a directory owned by qemu:

    ```
    mkdir /home/qemu; mkdir -p /home/qemu/VMs/fuel-6.0/disk
    ```

    ```
    chown -R qemu:qemu /home/qemu
    ```

15. Copy the ISO to /home/qemu

    ```
    cd /home/qemu
    ```

    ```
    virt-install -n opnfv-2015-05-22_18-34-07-fuel -r 4096
    --vcpus=4 --cpuset=0-3 -c opnfv-2015-05-22_18-34-07.iso
    ```

```
    --os-type=linux --os-variant=rhel6 --boot hd,cdrom --disk
    path=/home/qemu/VMs/mirantis-fuel-6.0/disk/fuel-vhd0.qcow2,bus=virtio,size=50,format=q
    -w bridge=br0,model=virtio --graphics vnc,listen=0.0.0.0
```

16. Temporarily flush the firewall rules to make things easier:

    ```
    iptables -F
    ```

17. Connect to the console of the installing VM with your favorite VNC client.
18. Change the IP settings to match the pod, use an IP in the PXE/Admin network for the Fuel Master

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

3.2 Apex configuration

3.2.1 Introduction

This document describes the steps to install an OPNFV Colorado reference platform, as defined by the Genesis Project using the Apex installer.

The audience is assumed to have a good background in networking and Linux administration.

3.2.2 Preface

Apex uses Triple-O from the RDO Project OpenStack distribution as a provisioning tool. The Triple-O image based life cycle installation tool provisions an OPNFV Target System (3 controllers, n number of compute nodes) with OPNFV specific configuration provided by the Apex deployment tool chain.

The Apex deployment artifacts contain the necessary tools to deploy and configure an OPNFV target system using the Apex deployment toolchain. These artifacts offer the choice of using the Apex bootable ISO (`opnfv-apex-colorado.iso`) to both install CentOS 7 and the necessary materials to deploy or the Apex RPMs (`opnfv-apex*.rpm`), and their associated dependencies, which expects installation to a CentOS 7 libvirt enabled host. The RPM contains a collection of configuration files, prebuilt disk images, and the automatic deployment script (`opnfv-deploy`).

An OPNFV install requires a "Jumphost" in order to operate. The bootable ISO will allow you to install a customized CentOS 7 release to the Jumphost, which includes the required packages needed to run `opnfv-deploy`. If you already have a Jumphost with CentOS 7 installed, you may choose to skip the ISO step and simply install the (`opnfv-apex*.rpm`) RPMs. The RPMs are the same RPMs included in the ISO and include all the necessary disk images and configuration files to execute an OPNFV deployment. Either method will prepare a host to the same ready state for OPNFV deployment.

`opnfv-deploy` instantiates a Triple-O Undercloud VM server using libvirt as its provider. This VM is then configured and used to provision the OPNFV target deployment (3 controllers, n compute nodes). These nodes can be either virtual or bare metal. This guide contains instructions for installing either method.

3.2.3 Installation High-Level Overview - Bare Metal Deployment

The setup presumes that you have 6 or more bare metal servers already setup with network connectivity on at least 2 interfaces for all servers via a TOR switch or other network implementation.

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.

The Jumphost can be installed using the bootable ISO or by other means including the (`opnfv-apex*.rpm`) RPMs, their dependencies and virtualization capabilities. The Jumphost should then be configured with an IP gateway on its admin or public interface and configured with a working DNS server. The Jumphost should also have routable access to the lights out network.

`opnfv-deploy` is then executed in order to deploy the Undercloud VM. `opnfv-deploy` uses three configuration files in order to know how to install and provision the OPNFV target system. The information gathered under section Execution Requirements (Bare Metal Only) is put into the YAML file `/etc/opnfv-apex/inventory.yaml` configuration file. Deployment options are put into the YAML file `/etc/opnfv-apex/deploy_settings.yaml`. Alternatively there are pre-baked deploy_settings files available in `/etc/opnfv-apex/`. These files are named with the naming convention os-sdn_controller-enabled_feature-[no]ha.yaml. These files can be used in place of the `/etc/opnfv-apex/deploy_settings.yaml` file if one suites your deployment needs. Networking definitions gathered under section Network Requirements are put into the YAML file `/etc/opnfv-apex/network_settings.yaml`. `opnfv-deploy` will boot the Undercloud VM and load the target deployment configuration into the provisioning toolchain. This includes MAC address, IPMI, Networking Environment and OPNFV deployment options.

Once configuration is loaded and the Undercloud is configured it will then reboot the nodes via IPMI. The nodes should already be set to PXE boot first off the admin interface. The nodes will first PXE off of the Undercloud PXE server and go through a discovery/introspection process.

Introspection boots off of custom introspection PXE images. These images are designed to look at the properties of the hardware that is booting off of them and report the properties of it back to the Undercloud node.

After introspection Undercloud will execute a Heat Stack Deployment to being node provisioning and configuration. The nodes will reboot and PXE again off the Undercloud PXE server to provision each node using the Glance disk images provided by Undercloud These disk images include all the necessary packages and configuration for an OPNFV deployment to execute. Once the node's disk images have been written to disk the nodes will boot off the newly written disks and execute cloud-init which will execute the final node configuration. This configuration is largly completed by executing a puppet apply on each node.

3.2.4 Installation High-Level Overview - VM Deployment

The VM nodes deployment operates almost the same way as the bare metal deployment with a few differences. `opnfv-deploy` still deploys an Undercloud VM. In addition to the Undercloud VM a collection of VMs (3 control nodes + 2 compute for an HA deployment or 1 control node and 1 compute node for a Non-HA Deployment) will be defined for the target OPNFV deployment. The part of the toolchain that executes IPMI power instructions calls into libvirt instead of the IPMI interfaces on baremetal servers to operate the power managment. These VMs are then provisioned with the same disk images and configuration that baremetal would be.

To Triple-O these nodes look like they have just built and registered the same way as bare metal nodes, the main difference is the use of a libvirt driver for the power management.

3.2.5 Installation Guide - Bare Metal Deployment

This section goes step-by-step on how to correctly install and provision the OPNFV target system to bare metal nodes.

Install Bare Metal Jumphost

1a. If your Jumphost does not have CentOS 7 already on it, or you would like to do a fresh install, then download the Apex bootable ISO from OPNFV artifacts <http://artifacts.opnfv.org/>. There have been isolated reports of problems with the ISO having trouble completing installation successfully. In the unexpected event the ISO does not work please workaround this by downloading the CentOS 7 DVD and performing a "Virtualization Host" install. If you perform a "Minimal Install" or install type other than "Virtualization Host" simply run `sudo yum groupinstall "Virtualization Host" && chkconfig libvirtd on && reboot` to install virtualzation support and enable libvirt on boot. If you use the CentOS 7 DVD proceed to step 1b once the CentOS 7 with "Virtualzation Host" support is completed.

1b. If your Jump host already has CentOS 7 with libvirt running on it then install the install the RDO Release RPM:

```
sudo yum install -y https://www.rdoproject.org/repos/rdo-release.rpm
opnfv-apex-{version}.rpm
```

The RDO Project release repository is needed to install OpenVSwitch, which is a dependency of opnfv-apex. If you do not have external connectivity to use this repository you need to download the OpenVSwitch RPM from the RDO Project repositories and install it with the opnfv-apex RPM.

2a. Boot the ISO off of a USB or other installation media and walk through installing OPNFV CentOS 7. The ISO comes prepared to be written directly to a USB drive with dd as such:

```
dd if=opnfv-apex.iso of=/dev/sdX bs=4M
```

Replace /dev/sdX with the device assigned to your usb drive. Then select the USB device as the boot media on your Jumphost

2b. If your Jump host already has CentOS 7 with libvirt running on it then install the opnfv-apex RPMs from OPNFV artifacts <http://artifacts.opnfv.org/>. The following RPMS are available for installation:

- opnfv-apex - OpenDaylight L2 / L3 and ONOS support **
- opnfv-apex-onos - ONOS support **
- opnfv-apex-opendaylight-sfc - OpenDaylight SFC support **
- opnfv-apex-undercloud - (required) Undercloud Image
- opnfv-apex-common - (required) Supporting config files and scripts
- python34-markupsafe - (required) Dependency of opnfv-apex-common ***
- python3-jinja2 - (required) Dependency of opnfv-apex-common ***

** One or more of these RPMs is required Only one of opnfv-apex, opnfv-apex-onos and opnfv-apex-opendaylight-sfc is required. It is safe to leave the unneeded SDN controller's RPMs uninstalled if you do not intend to use them.

*** These RPMs are not yet distributed by CentOS or EPEL. Apex has built these for distribution with Apex while CentOS and EPEL do not distribute them. Once they are carried in an upstream channel Apex will no longer carry them and they will not need special handling for installation.

To install these RPMs download them to the local disk on your CentOS 7 install and pass the file names directly to yum: `sudo yum install python34-markupsafe-<version>.rpm python3-jinja2-<version>.rpm` `sudo yum install opnfv-apex-<version>.rpm opnfv-apex-undercloud-<version>.rpm opnfv-apex-common-<version>.rpm`

3. After the operating system and the opnfv-apex RPMs are installed, login to your Jumphost as root.
4. Configure IP addresses on the interfaces that you have selected as your networks.
5. Configure the IP gateway to the Internet either, preferably on the public interface.
6. Configure your `/etc/resolv.conf` to point to a DNS server (8.8.8.8 is provided by Google).

Creating a Node Inventory File

IPMI configuration information gathered in section Execution Requirements (Bare Metal Only) needs to be added to the `inventory.yaml` file.

1. Copy `/usr/share/doc/opnfv/inventory.yaml.example` as your inventory file template to `/etc/opnfv-apex/inventory.yaml`.
2. The nodes dictionary contains a definition block for each baremetal host that will be deployed. 1 or more compute nodes and 3 controller nodes are required. (The example file contains blocks for each of these already). It is optional at this point to add more compute nodes into the node list.
3. Edit the following values for each node:
    - `mac_address`: MAC of the interface that will PXE boot from Undercloud
    - `ipmi_ip`: IPMI IP Address
    - `ipmi_user`: IPMI username
    - `ipmi_password`: IPMI password
    - `pm_type`: Power Management driver to use for the node
    - `cpus`: (Introspected*) CPU cores available
    - `memory`: (Introspected*) Memory available in Mib
    - `disk`: (Introspected*) Disk space available in Gb
    - `arch`: (Introspected*) System architecture
    - `capabilities`: (Optional**) Intended node role (profile:control or profile:compute)

- Introspection looks up the overcloud node's resources and overrides these value. You can

leave default values and Apex will get the correct values when it runs introspection on the nodes.

** If capabilities profile is not specified then Apex will select node's roles in the OPNFV cluster in a non-deterministic fashion.

Creating the Settings Files

Edit the 2 settings files in /etc/opnfv-apex/. These files have comments to help you customize them.

1. deploy_settings.yaml This file includes basic configuration options deployment. Alternatively, there are pre-built deploy_settings files available in (`/etc/opnfv-apex/`). These files are named with the naming convention os-sdn_controller-enabled_feature-[no]ha.yaml. These files can be used in place of the (`/etc/opnfv-apex/deploy_settings.yaml`) file if one suites your deployment needs. If a pre-built deploy_settings file is choosen there is no need to customize (`/etc/opnfv-apex/deploy_settings.yaml`). The pre-built file can be used in place of the (`/etc/opnfv-apex/deploy_settings.yaml`) file.
2. network_settings.yaml This file provides Apex with the networking information that satisfies the prerequisite Network Requirements. These are specific to your environment.

Running `opnfv-deploy`

You are now ready to deploy OPNFV using Apex! `opnfv-deploy` will use the inventory and settings files to deploy OPNFV.

Follow the steps below to execute:

1. Execute opnfv-deploy `sudo opnfv-deploy [ --flat ] -n network_settings.yaml -i inventory.yaml -d deploy_settings.yaml` If you need more information about the options that can be passed to opnfv-deploy use `opnfv-deploy --help` –flat collapses all networks to a single nic, only uses the admin network from the network settings file. -n network_settings.yaml allows you to customize your networking topology.
2. Wait while deployment is executed. If something goes wrong during this part of the process, it is most likely a problem with the setup of your network or the information in your configuration files. You will also notice different outputs in your shell.
3. The message "Overcloud Deployed" will display when When the deployment is complete. Just above this message there will be a URL that ends in port http://<host>:5000. This url is also the endpoint for the OPNFV Horizon Dashboard if connected to on port 80.

3.3 Compass4nfv configuration

This document describes providing guidelines on how to install and configure the Brahmaputra release of OPNFV when using Compass as a deployment tool including required software and hardware configurations.

Installation and configuration of host OS, OpenStack, OpenDaylight, ONOS, Ceph etc. can be supported by Compass on VMs or Bare Metal nodes.

The audience of this document is assumed to have good knowledge in networking and Unix/Linux administration.

3.3.1 Preconditions

Before starting the installation of the Brahmaputra release of OPNFV, some planning must be done.

Retrieving the installation ISO image

First of all, The installation ISO is needed for deploying your OPNFV environment, it included packages of Compass, OpenStack, OpenDaylight, ONOS and so on.

The stable release ISO can be retrieved via OPNFV software download page

The daily build ISO can be retrieved via OPNFV artifacts repository:

http://artifacts.opnfv.org/

NOTE: Search the keyword "Compass4nfv/Brahmaputra" to locate the ISO image.

E.g. compass4nfv/brahmaputra/opnfv-2016-01-16_15-03-18.iso compass4nfv/brahmaputra/opnfv-2016-01-16_15-03-18.properties

The name of iso image includes the time of iso building, you can get the daily ISO according the building time. The git url and sha1 of Compass4nfv are recorded in properties files, According these, the corresponding deployment scripts can be retrieved.

Getting the deployment scripts

To retrieve the repository of Compass4nfv on Jumphost use the following command:

- git clone https://gerrit.opnfv.org/gerrit/compass4nfv

To get stable/brahmaputra release, you can use the following command:

- git checkout brahmaputra.1.0

NOTE: PLEASE DO NOT GIT CLONE COMPASS4NFV IN root DIRECTORY.

If you don't have a Linux foundation user id, get it first by the url:

https://wiki.opnfv.org/developer/getting_started

If you want to use a daily release ISO, please checkout the corresponding sha1 to get the deployment scripts:

E.g. Git sha1 in file "opnfv-2016-01-16_15-03-18.properties" is d5a13ce7cc2ce89946d34b0402ecf33c1d291851

- git checkout d5a13ce7cc2ce89946d34b0402ecf33c1d291851

Preparing the installation environment

If you have only 1 Bare Metal server, Virtual deployment is recommended. if more than or equal 3 servers, the Bare Metal deployment is recommended. The minimum number of servers for Bare metal deployment is 3, 1 for JumpServer(Jumphost), 1 for controller, 1 for compute.

3.3.2 Setup Requirements

Jumphost Requirements

The Jumphost requirements are outlined below:

1. Ubuntu 14.04 (Pre-installed).
2. Root access.
3. libvirt virtualization support.
4. Minimum 2 NICs.
    - PXE installation Network (Receiving PXE request from nodes and providing OS provisioning)
    - IPMI Network (Nodes power control and set boot PXE first via IPMI interface)
    - External Network (Optional: Internet access)
5. 16 GB of RAM for a Bare Metal deployment, 64 GB of RAM for a VM deployment.
6. Minimum 100G storage.

Bare Metal Node Requirements

Bare Metal nodes require:

1. IPMI enabled on OOB interface for power control.
2. BIOS boot priority should be PXE first then local hard disk.
3. Minimum 3 NICs.
    - PXE installation Network (Broadcasting PXE request)
    - IPMI Network (Receiving IPMI command from Jumphost)
    - External Network (OpenStack mgmt/external/storage/tenant network)

Network Requirements

Network requirements include:

1. No DHCP or TFTP server running on networks used by OPNFV.
2. 2-6 separate networks with connectivity between Jumphost and nodes.
    - PXE installation Network
    - IPMI Network
    - Openstack mgmt Network*
    - Openstack external Network*
    - Openstack tenant Network*
    - Openstack storage Network*
3. Lights out OOB network access from Jumphost with IPMI node enabled (Bare Metal deployment only).
4. External network has Internet access, meaning a gateway and DNS availability.

The networks with(*) can be share one NIC(Default configuration) or use an exclusive NIC(Reconfigurated in network.yml).

Execution Requirements (Bare Metal Only)

In order to execute a deployment, one must gather the following information:

1. IPMI IP addresses for the nodes.
2. IPMI login information for the nodes (user/pass).
3. MAC address of Control Plane / Provisioning interfaces of the Bare Metal nodes.

3.3.3 Installation Guide (BM Deployment)

Nodes Configuration (BM Deployment)

The bellow file is the inventory template of deployment nodes:

“compass4nfv/deploy/conf/hardware_environment/huawei_us_lab/pod1/dha.yml”

You can write your own IPMI IP/User/Password/Mac address/roles reference to it.

- ipmiVer – IPMI interface version for deployment node support. IPMI 1.0 or IPMI 2.0 is available.
- ipmiIP – IPMI IP address for deployment node. Make sure it can access from Jumphost.
- ipmiUser – IPMI Username for deployment node.
- ipmiPass – IPMI Password for deployment node.
- mac – MAC Address of deployment node PXE NIC .
- name – Host name for deployment node after installation.
- roles – Components deployed.

Assignment of different roles to servers

E.g. Openstack only deployment roles setting

```
hosts:
  - name: host1
    roles:
      - controller
      - ha

  - name: host2
    roles:
      - compute
```

NOTE: IF YOU SELECT MUTIPLE NODES AS CONTROLLER, THE 'ha' role MUST BE SELECT, TOO.

E.g. Openstack and ceph deployment roles setting

```
hosts:
  - name: host1
    roles:
      - controller
      - ha
      - ceph-adm
      - ceph-mon

  - name: host2
    roles:
      - compute
      - ceph-osd
```

E.g. Openstack and ODL deployment roles setting

```
hosts:
  - name: host1
    roles:
      - controller
      - ha
      - odl

  - name: host2
    roles:
      - compute
```

E.g. Openstack and ONOS deployment roles setting

```
hosts:
  - name: host1
    roles:
      - controller
      - ha
      - onos

  - name: host2
    roles:
      - compute
```

Network Configuration (BM Deployment)

Before deployment, there are some network configuration to be checked based on your network topology. Compass4nfv network default configuration file is "compass4nfv/deploy/conf/network_cfg.yaml". You can write your own reference to it.

The following figure shows the default network configuration.

```
+--+                          +--+     +--+
|  |                          |  |     |  |
|  |      +------------+      |  |     |  |
|  +------+  Jumphost  +------+  |     |  |
|  |      +------+-----+      |  |     |  |
|  |             |            |  |     |  |
|  |             +------------+  +-----+  |
|  |                          |  |     |  |
|  |      +------------+      |  |     |  |
|  +------+    host1   +------+  |     |  |
|  |      +------+-----+      |  |     |  |
|  |             |            |  |     |  |
|  |             +------------+  +-----+  |
|  |                          |  |     |  |
|  |      +------------+      |  |     |  |
|  +------+    host2   +------+  |     |  |
|  |      +------+-----+      |  |     |  |
|  |             |            |  |     |  |
|  |             +------------+  +-----+  |
|  |                          |  |     |  |
|  |      +------------+      |  |     |  |
|  +------+    host3   +------+  |     |  |
|  |      +------+-----+      |  |     |  |
|  |             |            |  |     |  |
|  |             +------------+  +-----+  |
|  |                          |  |     |  |
|  |                          |  |     |  |
+-++                          ++-+     +-++
  ^                             ^         ^
  |                             |         |
  |                             |         |
+-+-------------------------+   |         |
|      External Network     |   |         |
+---------------------------+   |         |
       +-----------------------+---+      |
       |       IPMI Network        |      |
       +---------------------------+      |
               +-------------------------+-+
               | PXE(Installation) Network |
               +---------------------------+
```

Start Deployment (BM Deployment)

1. Set PXE/Installation NIC for Jumphost. (set eth1 E.g.)

```
export INSTALL_NIC=eth1
```

2. Set OS version for nodes provisioning. (set Ubuntu14.04 E.g.)

```
export OS_VERSION=trusty
```

3. Set OpenStack version for deployment nodes. (set liberty E.g.)

```
export OPENSTACK_VERSION=liberty
```

4. Set ISO image that you want to deploy

```
export ISO_URL=file:///${YOUR_OWN}/compass.iso
or
export ISO_URL=http://artifacts.opnfv.org/compass4nfv/brahmaputra/opnfv-release.iso
```

5. Run `deploy.sh` with inventory and network configuration

```
./deploy.sh --dha ${YOUR_OWN}/dha.yml --network ${YOUR_OWN}/network.yml
```

3.4 Fuel configuration

This section provides 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 *Reference 13* in section *"Fuel associated references"* below.

3.4.1 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, please see *Reference: 8* in section *"Fuel associated references"* below.
- Fuel quick start guide, please see *Reference: 9* in section *"Fuel associated references"* below.
- Fuel operations guide, please see *Reference: 10* in section *"Fuel associated references"* below.
- Fuel Plugin Developers Guide, please see *Reference: 11* in section *"Fuel associated references"* below.

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 Fuel .iso image of the Brahmaputra release can be found at *Reference: 2*

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 in *Reference: 14* at section *"Fuel associated references"* below.

3.4.2 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 information on compatible hardware types available for use, please see *Reference: 11* in section *"Fuel associated references"* below.

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.

3.4.3 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 installation instructions, see *Reference 13* in section *"Fuel associated references"* below.

3.4.4 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-Daylight	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 section "Configuration options, OPNFV scenarios" for compatibility information**

The plugins come prepackaged, ready to install. To do so follow the installation instructions provided in *Reference 13* provided in section *"Fuel associated references"* below.

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 in the installation instructions, see *Reference 13* in section *"Fuel associated references"* below.

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 your choice need to be enabled and configured.

To enable a plugin, follow the installation instructions found in *Reference 13*, provided in section *"Fuel associated references"* below.

For configuration of the plugins, please see section "Feature Configuration".

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 found in *Reference 13* provided in section *"Fuel associated references"* below.

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 found in *Reference 13*, provided in section *"Fuel associated references"* below.

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 found in, *Reference 13*, provided in section *"Fuel associated references"* below.

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 found in, *Reference 13*, provided in section *"Fuel associated references"* below.

3.4.5 Fuel associated references

1. OPNFV Home Page
2. OPNFV documentation- and software downloads
3. OpenStack Liberty Release artifacts
4. OpenStack documentation
5. OpenDaylight artifacts
6. The Fuel OpenStack project
7. Fuel documentation overview
8. Fuel planning guide
9. Fuel quick start guide

10. Fuel operations guide
11. Fuel Plugin Developers Guide
12. Fuel OpenStack Hardware Compatibility List
13. OPNFV Installation instruction for the Brahmaputra release of OPNFV when using Fuel as a deployment tool
14. OPNFV Build instruction for the Brahmaputra release of OPNFV when using Fuel as a deployment tool
15. OPNFV Release Note for the Brahmaputra release of OPNFV when using Fuel as a deployment tool

3.5 JOID Configuration

3.5.1 Bare Metal Installations:

3.5.2 Requirements as per Pharos:

3.5.3 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.

3.5.4 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

3.5.5 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.

3.5.6 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' ./bunc
```

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 Procedure

The install currently occurs via four bash scripts provided in the copper repo. See these files for the detailed steps:

- install_congress_1.sh * creates and starts the linux container for congress on the controller node * copies install_congress_2.sh to the controller node and invokes it via ssh
- install_congress_2.sh * installs congress on the congress server.

Cleanup Procedure

If there is an error during installation, use the bash script clean_congress.sh which stops the congress server if running, and removes the congress user and service from the controller database.

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
source ~/env.sh
juju ssh ubuntu@$CONGRESS_HOST
# If that fails
  # On jumphost, SSH to controller node
  juju ssh ubuntu@node1-control
  # 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 ~/git/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-apex/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 -l default -f ipv6

# Non-HA deployment in OpenStack-only environment
./deploy.sh -o liberty -s nosdn -t nonha -l default -f ipv6
```

Please **NOTE** that:

- You need to refer to **installer's documentation** for other necessary parameters applicable to your deployment.
- You need to refer to **Release Notes** and **installer's documentation** if there is any issue in installation.

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
```

Fuel installer:

```
# source the tenant credentials using Fuel installer of OPNFV
source /root/openrc
```

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-apex/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 -l default -f ipv6

# Non-HA deployment in OpenStack with Open Daylight L2-only environment
./deploy.sh -o liberty -s odl -t nonha -l default -f ipv6
```

Please **NOTE** that:

- You need to refer to **installer's documentation** for other necessary parameters applicable to your deployment.
- You need to refer to **Release Notes** and **installer's documentation** if there is any issue in installation.

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
```

Fuel installer:

```
# source the tenant credentials using Fuel installer of OPNFV
source /root/openrc
```

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 Installing OVSNFV Fuel Plugin

- On the Fuel UI, create a new environment.
- In Settings > Userspace OVS support, check “Userspace OVS support”.
- Continue with environment configuration and deployment as normal.

4.4.1 Upgrading the plugin

From time to time new versions of the plugin may become available.

The plugin cannot be upgraded if an active environment is using the plugin.

In order to upgrade the plugin:

- Copy the updated plugin file to the fuel-master.
- On the Fuel UI, reset the environment.
- On the Fuel CLI "fuel plugins –update <fuel-plugin-file>"
- On the Fuel UI, re-deploy the environment.

4.5 Promise Feature Configuration Overview

4.5.1 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)

4.5.2 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

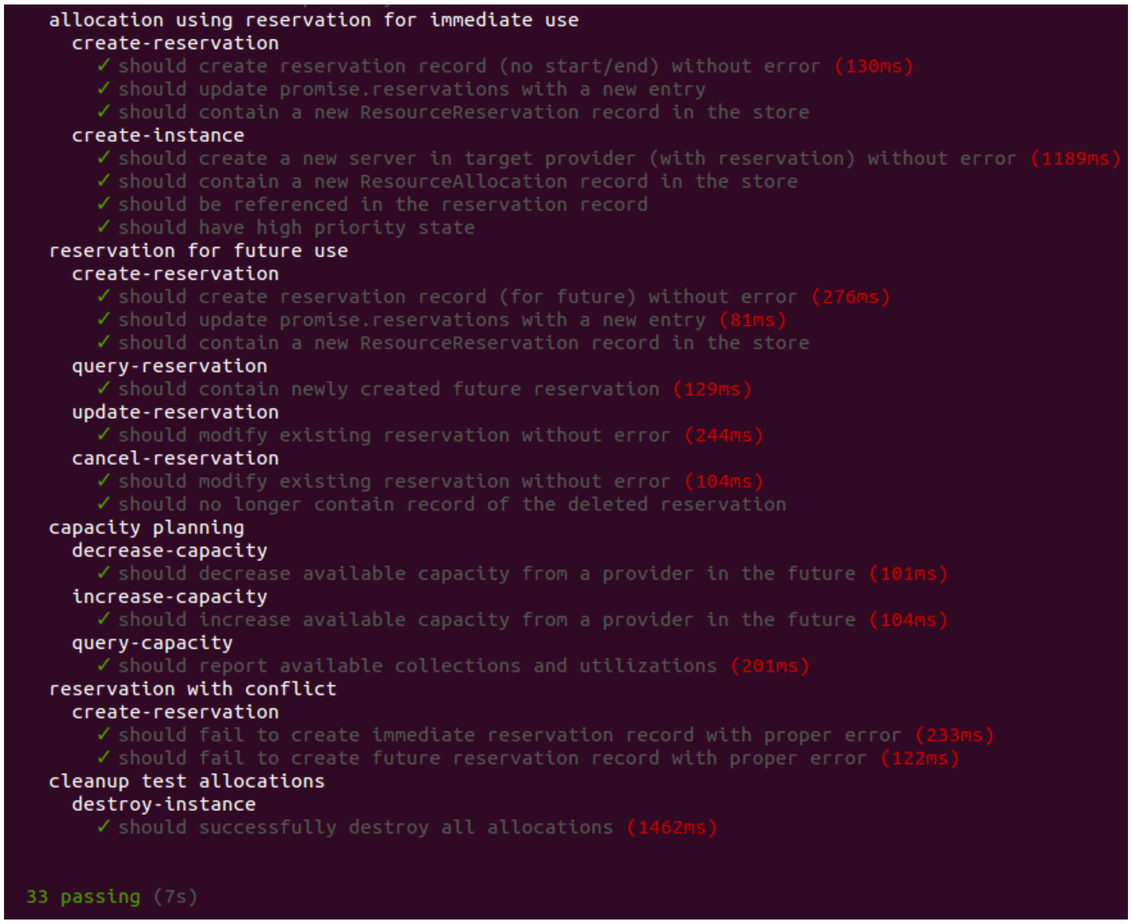
Once done, the promise test script can be invoked as follows (as a single line command):

```
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)
```

4.6 Configuring SDNVPN features

Fuel installer configuration

In order to install the BGPVPN feature, the corresponding checkbox in Fuel has to be selected. This will trigger installation of the OpenStack BGPVPN API extension for Neutron (set up for using the ODL driver).

In addition, ODL has to be installed, see the corresponding section in the respective installer documentation on how to install ODL. If the BGPVPN feature is installed, ODL will automatically be installed with VPN Service karaf feature activated.

No post-deploy configuration is necessary. The Fuel BGPVPN plugin and the ODL plugin should set up the cluster ready for BGPVPNs being created. This includes the set-up of internal VxLAN transport tunnels between compute nodes.

No post-configuration activities are required.

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 Fuel post installation procedures

Automated post installation activities

Fuel provides a fairly broad coverage of built in automated health checks. These validate the installation in terms of configuration, services, networking, storage, policies, etc. The execution of the full range of health checks takes less than 30 minutes.

For instructions on how to run health-checks, please read the Fuel installation instructions.

Platform components validation

Consult the feature sections in this document for any post-install feature specific validation/health-checks.

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.       |
```

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.

5.2.2 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.

5.3 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 Overview of the Functest suites

Functest is the OPNFV project primarily targeting function testing. In the Continuous Integration pipeline, it is launched after an OPNFV fresh installation to validate and verify the basic functions of the infrastructure.

The current list of test suites can be distributed over 4 main domains: VIM (Virtualised Infrastructure Manager), Controllers (i.e. SDN Controllers), Features and VNF (Virtual Network Functions).

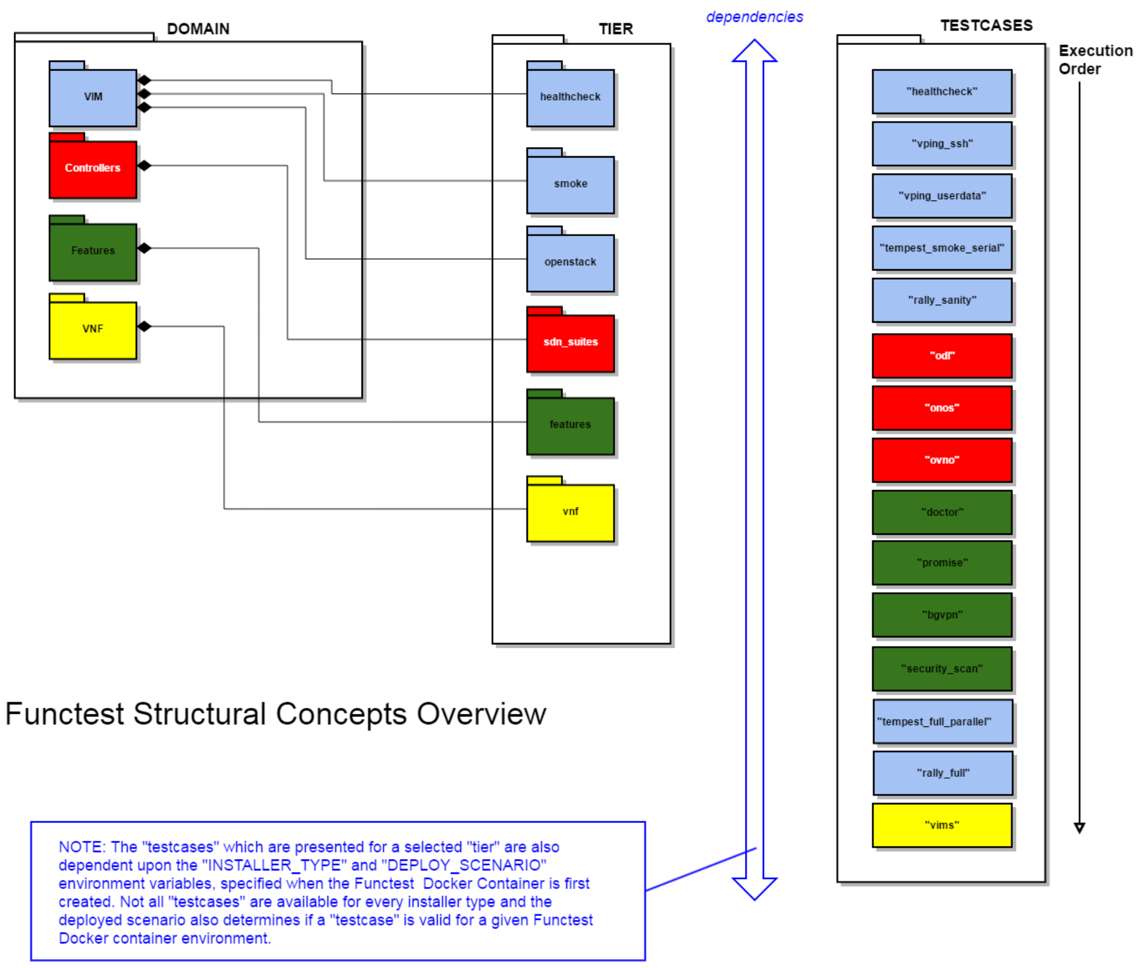
Domain	Tier	Test case	Comments
VIM	healthcheck	healthcheck	Verify basic operation in VIM
	smoke	vPing_SSH	NFV “Hello World” using an SSH connection to a destination VM over a created floating IP address on the SUT Public / External network. Using the SSH connection a test script is then copied to the destination VM and then executed via SSH. The script will ping another VM on a specified IP address over the SUT Private Tenant network.
		vPing_userdata	Uses Ping with given userdata to test intra-VM connectivity over the SUT Private Tenant network. The correct operation of the NOVA Metadata service is also verified in this test.
		tempest_smoke_serial	Generate and run a relevant Tempest Test Suite in smoke mode. The generated test set is dependent on the OpenStack deployment environment.
		rally_sanity	Run a subset of the OpenStack Rally Test Suite in smoke mode
	openstack	tempest_full_parallel	Generate and run a full set of the OpenStack Tempest Test Suite. See the OpenStack reference test suite [2]. The generated test set is dependent on the OpenStack deployment environment.
		rally_full	Run the OpenStack testing tool benchmarking OpenStack modules See the Rally documents [3].
Controllers	sdn_suites	odl	Opendaylight Test suite TODO: Find a document reference!
		onos	Test suite of ONOS L2 and L3 functions. See ONOSFW User Guide for details.
Features	features	Promise	Resource reservation and management project to identify NFV related requirements and realize resource reservation for future usage by capacity management of resource pools regarding compute, network and storage. See Promise User Guide for details.
		Doctor	Doctor platform, as of Colorado release, provides the two features: * Immediate Notification * Consistent resource state awareness (compute). See the See Doctor User Guide for details
		bgpvpn	Implementation of the OpenStack bgpvpn API from the SDNVPN feature project. It allows for the creation of BGP VPNs. See SDNVPN User Guide for details
		security_scan	Implementation of a simple security scan. (Currently available only for the Apex installer environment) TODO: Add document link from Luke Hinds; when received.
VNF	vnf	vims	Example of a real VNF deployment to show the NFV capabilities of the platform. The IP Multimedia Subsytem is a typical Telco test case, referenced by ETSI. It provides a fully functional VoIP System,

As shown in the above table, Functest is structured into different ‘domains’, ‘tiers’ and ‘test cases’. Each ‘test case’ usually represents an actual ‘Test Suite’ comprised -in turn- of several test cases internally.

Test cases also have an implicit execution order. For example, if the early ‘healthcheck’ Tier testcase fails, or if there are any failures in the ‘smoke’ Tier testcases, there is little point to launch a full testcase execution round.

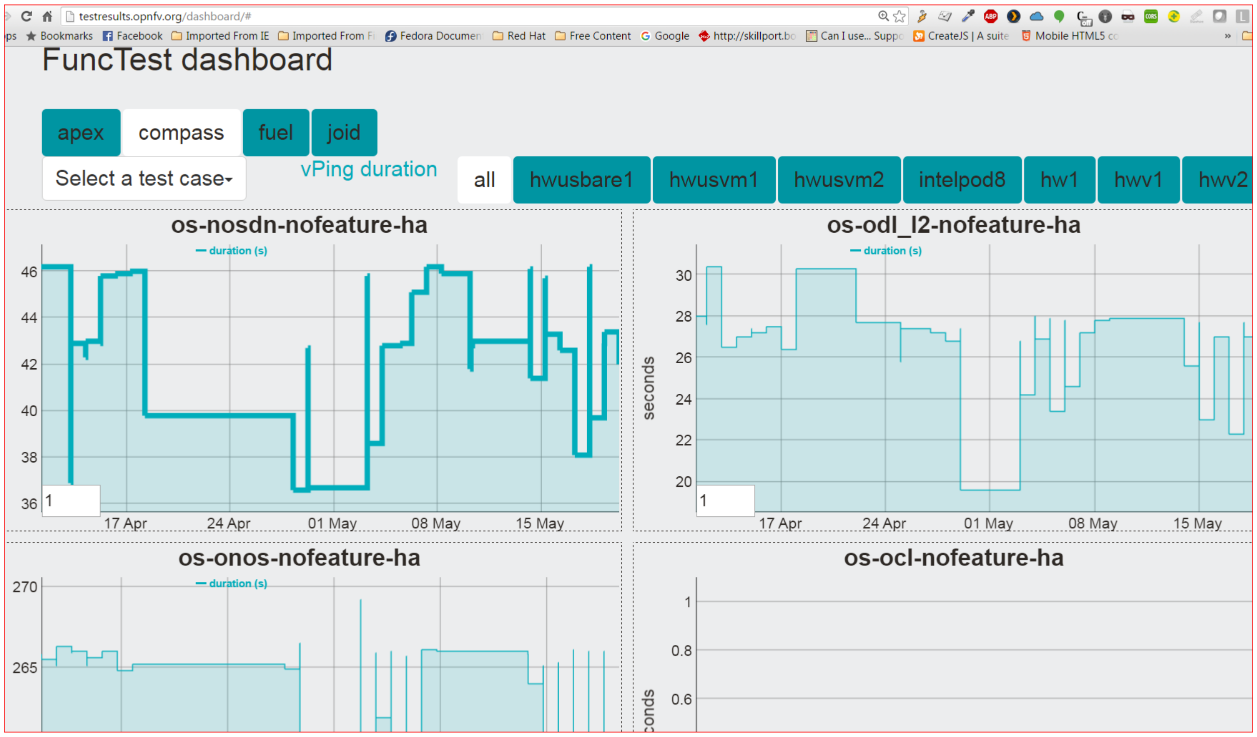
An overview of the Functest Structural Concept is depicted graphically below:

Some of the test cases are developed by Functest team members, whereas others are integrated from upstream communities or other OPNFV projects. For example, Tempest is the OpenStack integration test suite and Functest is in charge of the selection, integration and automation of those tests that fit suitably to OPNFV.


DOMAIN
VIM
Controllers
Features
VNF
TIER
healthcheck
smoke
openstack
sdn_suites
features
vnf
dependencies
TESTCASES
Execution Order
"healthcheck"
"vping_ssh"
"vping_userdata"
"tempest_smoke_serial"
"rally_sanity"
"odl"
"onos"
"ovno"
"doctor"
"promise"
"bgvpn"
"security_scan"
"tempest_full_parallel"
"rally_full"
"vims"
Functest Structural Concepts Overview
NOTE: The "testcases" which are presented for a selected "tier" are also dependent upon the "INSTALLER_TYPE" and "DEPLOY_SCENARIO" environment variables, specified when the Functest Docker Container is first created. Not all "testcases" are available for every installer type and the deployed scenario also determines if a "testcase" is valid for a given Functest Docker container environment.

The Tempest test suite has been customized but no new test cases have been created in OPNFV Functest.

The results produced by the tests run from CI are pushed and collected into a NoSQL database. The goal is to populate the database with results from different sources and scenarios and to show them on a Functest Dashboard. A screenshot of a live Functest Dashboard is shown below:



There is no real notion of Test domain or Test coverage. Basic components (VIM, SDN controllers) are tested through their own suites. Feature projects also provide their own test suites with different ways of running their tests.

vIMS test case was integrated to demonstrate the capability to deploy a relatively complex NFV scenario on top of the OPNFV infrastructure.

Functest considers OPNFV as a black box. As of Colorado release the OPNFV offers a lot of potential combinations:

- 3 controllers (OpenDaylight, ONOS, OpenContrail)
- 4 installers (Apex, Compass, Fuel, Joid)

Most of the tests are runnable by any combination, but some tests might have restrictions imposed by the utilized installers or due to the available deployed features. The system uses the environment variables (INSTALLER_IP and DEPLOY_SCENARIO) to automatically determine the valid test cases; for each given environment.

In the Colorado OPNFV System release a convenience Functest CLI utility is also introduced to simplify setting up the Functest evironment, management of the OpenStack environment (e.g. resource clean-up) and for executing tests. The Functest CLI organised the testcase into logical Tiers, which contain in turn one or more testcases. The CLI allow execution of a single specified testcase, all test cases in a specified Tier, or the special case of execution of **ALL** testcases. The Functest CLI is introduced in more detail in the section **'Executing the functest suites'_** of this document.

Preparing the Docker container

Pull the Functest Docker image ('opnfv/functest') from the public dockerhub registry under the OPNFV account: [dockerhub], with the following docker command:

```
docker pull opnfv/functest:<TagIdentifier>
```

where <TagIdentifier> identifies a specifically tagged release of the Functest docker container image in the public dockerhub registry. There are many different tags created automatically by the CI mechanisms, but you must ensure you pull an image with the **correct tag** to match the OPNFV software release installed in your environment. All available tagged images can be seen from location [FunctestDockerTags]. For example, when running on the first official release of the OPNFV Colorado system platform, tag “colorado.1.0” is needed. Pulling other tags might cause some problems while running the tests. If you need to specifically pull the latest Functest docker image, then omit the tag argument:

```
docker pull opnfv/functest
```

After pulling the Docker image, check that the pulled image is available with the following docker command:

```
[functester@jumphost ~]$ docker images

REPOSITORY      TAG              IMAGE ID      CREATED      SIZE
opnfv/functest latest           8cd6683c32ae  2 weeks ago  1.611 GB
opnfv/functest brahmaputra.3.0 94b78faa94f7  4 weeks ago  874.9 MB
hello-world    latest           94df4f0ce8a4  7 weeks ago    967 B

(Docker images pulled without a tag specifier bear the implicitly
 assigned label "latest", as seen above.)
```

The Functest docker container environment can -in principle- be also used with non-OPNFV official installers (e.g. ‘devstack), with the **disclaimer** that support for such environments is outside of the scope of responsibility of the OPNFV project.

The minimum command to create the Functest Docker container can be described as follows:

```
docker run -it opnfv/functest:<TagIdentifier> /bin/bash
```

For OPNFV official installers, it is recommended (although no longer mandatory) to provide two additional environment variables, in the ‘docker run ...’ command nvocation:

- **INSTALLER_TYPE** : possible values are **apex**, **compass**, **fuel** or **joid**.
- **INSTALLER_IP** : IP of the installer node/VM.

Functest may need to know the IP of the installer to retrieve automatically the credentials from the installer node/VM or even from the actual controllers.

Thus, the recommended minimum command to create the Functest Docker container for OPNFV installer can be described (using installer ‘fuel’, and an invented INSTALLER_IP of ‘10.20.0.2’, for example), as follows:

```
docker run -it \
-e "INSTALLER_IP=10.20.0.2" \
-e "INSTALLER_TYPE=fuel" \
opnfv/functest:<TagIdentifier> /bin/bash
```

Optionally, it is possible to assign precisely a container name through the **–name** option:

```
docker run --name "CONTAINER_NAME" -it \
-e "INSTALLER_IP=10.20.0.2" \
-e "INSTALLER_TYPE=fuel" \
opnfv/functest:<TagIdentifier> /bin/bash
```

It is also possible to to indicate the path of the OpenStack credentials using a **-v** option:

```
docker run -it \
-e "INSTALLER_IP=10.20.0.2" \
-e "INSTALLER_TYPE=fuel" \
-v <path_to_your_local_creds_file>:/home/opnfv/functest/conf/openstack.creds \
opnfv/functest:<TagIdentifier> /bin/bash

NOTE: Make sure you have placed the needed credential file into the
      Jumphost local path <path_to_your_local_cred_file>. For the
      Apex Installer you will need to pre-copy the required OpenStack
      credentials file from the Instack/Undercloud Virtual Machine.
      See the section 'Apex Installer Tips' later in this document.

Warning
-------
If you are using the Joid installer, you must use the method above
to provide the required OpenStack credentials. See the section
'Focus on the OpenStack credentials' later in this document.
```

The local openstack credential file will be mounted in the Docker container under the path: '/home/opnfv/functest/conf/openstack.creds'

If the intention is to run Functest against any of the supported OPNFV scenarios, it is recommended to include also the environment variable **DEPLOY_SCENARIO**. The **DEPLOY_SCENARIO** environment variable is passed with the format:

```
-e "DEPLOY_SCENARIO=os-<controller>-<nfv_feature>-<ha_mode>"

where:
os = OpenStack (No other VIM choices currently available)
controller  is one of ( nosdn | odl_l2 | odl_l3 | onos | ocl )
nfv_feature is one or more of ( ovs | kvm | sfc | bgpvpn | nofeature )
            If several features are pertinent then use the underscore
            character '_' to separate each feature (e.g. ovs_kvm)
            'nofeature' indicates no NFV feature is deployed
ha_mode     is one of ( ha | noha )
```

For example:

```
docker run -it \
-e "INSTALLER_IP=10.20.0.2" \
-e "INSTALLER_TYPE=fuel" \
-e "DEPLOY_SCENARIO=os-odl_l2-ovs_kvm-ha" \
opnfv/functest:<TagIdentifier> /bin/bash
```

NOTE: Not all possible combinations of "DEPLOY_SCENARIO" are supported. The scenario name passed in to the Functest Docker container must match the scenario used with the selected installer to create the actual OPNFV platform deployment.

Finally, three additional environment variables can also be passed in to the Functest Docker Container, using the -e "<EnvironmentVariableName>=<Value>" mechanism. The first two of these are only relevant to Jenkins CI invoked testing and **should not be used** when performing manual test scenarios:

```
-e "NODE_NAME=<Test POD Name>" \
-e "BUILD_TAG=<Jenkins Build Tag>" \
-e "CI_DEBUG=<DebugTraceValue>"

where:
<Test POD Name> = Symbolic name of the POD where the tests are run.
                  Visible in test results files, which are stored
                  to the database. This option is only used when
                  tests are activated under Jenkins CI control.
                  It indicates the POD/hardware where the test has
                  been run. If not specified, then the POD name is
                  defined as "Unknown" by default.
                  DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.

<Jenkins Build tag> = Symbolic name of the Jenkins Build Job.
                      Visible in test results files, which are stored
                      to the database. This option is only set when
                      tests are activated under Jenkins CI control.
                      It enables the correlation of test results, which
                      are independently pushed to the results datbase
                      from different Jenkins jobs.
                      DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.

<DebugTraceValue> = "true" or "false"
                    Default = "false", if not specified
                    If "true" is specified, then additional debug trace
                    text can be sent to the test results file / log files
                    and also to the standard console output.
```

Apex Installer Tips

Some specific tips are useful for the Apex Installer case. If not using Apex Installer; ignore this section.

1. The “INSTALLER_IP” environment variable should be set equal to the IP address of the so-called “Instack/undercloud Virtual Machine”.

   In the Jumphost, execute the following command and note the returned IP address:

```
sudo virsh domifaddr undercloud | grep -Eo "[0-9.]+{4}"

NOTE: In releases prior to Colorado, the name 'instack' was
used. From Colorado onward, the name 'undercloud' is used.
If in doubt, then execute -from the Jumphost- the command
"virsh list" to see which name is in use for the Installer
Virtual Machine.
```

   You can now enter the <Specific IP Address> as learned in the above step in the -e option specification:

```
-e "INSTALLER_IP=<Specific IP Address>"
```

2. If you want to ‘Bind mount’ a local Openstack credentials file (“overcloudrc”) to the Docker container, then you may need to first pre-copy that file from the ‘Instack/Undercloud VM’ to the Jump host.

   As before, in the Jumphost, execute the following command and note the returned IP address:

```
sudo virsh domifaddr undercloud | grep -Eo "[0-9.]+{4}"
```

   Using the <Specific IP Address> just learned above, execute the following shell commands **in the Jumphost**, before issuing the ‘docker run ...’ command invocation:

```
scp stack@<Specific IP Address>:overcloudrc .
sed -i 's/export no_proxy/#export no_proxy/' overcloudrc
# The above 'sed' command is needed *only* in cases where
# the Jumphost is operating behind a http proxy.
# See the 'Proxy Support' section later on in this document
```

```
    NOTE: There are two Openstack credential files present in the
    Instack/Undercloud VM: 'overcloudrc' and 'stackrc'.
    Don't mix these up! The file 'stackrc' is intended for use with
    'Triple O Undercloud'; only. The SUT always requires OpenStack
    Overcloud Credentials.
```

The file located at Jumphost path: '~/overcloudrc' is now 'Bind mounted' to the Docker path '/home/opnfv/functest/conf/openstack.creds' by specifying a **-v** option:

```
    -v ~/overcloudrc:/home/opnfv/functest/conf/openstack.creds
```

in the argument list of the 'docker run ...' command invocation. In the Apex installer case, the Openstack Credential file has the name 'overcloudrc' and is located in the home directory of the 'stack' user ('/home/stack/' or '~/']) in the 'Instack/Undercloud VM'.

3. In order that the docker container can access the Instack/Undercloud VM, even with 'stack' user, the SSH keys of the Jumphost root user **must be** 'Bind mounted' to the docker container by the following **-v** option in the 'docker run ...' command invocation:

```
    -v /root/.ssh/id_rsa:/root/.ssh/id_rsa
```

4. Here is an example of the docker command invocation for an Apex installed system, using latest Funtest docker container, for illustration purposes:

```
    docker run -it --name "ApexFuncTstODL" \
    -e "INSTALLER_IP=<Specific IP Address>" \
    -e "INSTALLER_TYPE=apex" \
    -e "DEPLOY_SCENARIO=os-odl_l2-nofeature-ha" \
    -v /root/.ssh/id_rsa:/root/.ssh/id_rsa \
    -v ~/overcloudrc:/home/opnfv/functest/conf/openstack.creds \
    opnfv/functest /bin/bash
```

Functest docker container directory structure

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

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

(The sub-directory 'ovno' holds SDN controller functional tests
 for the OpenContrail SDN Controller, which should be available
 for Colorado release)
```

Underneath the '/home/opnfv/' directory, the Functest docker container includes two main directories:

- The **functest** directory stores configuration files (e.g. the OpenStack creds are stored in path '/home/opnfv/functest/conf/openstack.creds'), the **data** directory stores a 'cirros' test image used in some functional tests and the **results** directory stores some temporary result log files
- The **repos** directory holds various repositories. The directory '/home/opnfv/repos/functest' is used to prepare the needed Functest environment and to run the tests. The other repository directories are used for the installation of the needed tooling (e.g. rally) or for the retrieval of feature projects scenarios (e.g. promise)

The structure under the **functest** repository can be described as follows:

```
. |-- INFO
  |-- LICENSE
  |-- __init__.py
  |-- ci
  |   |-- __init__.py
  |   |-- check_os.sh
  |   |-- config_functest.yaml
  |   |-- exec_test.sh
  |   |-- prepare_env.py
  |   |-- run_tests.py
  |   |-- testcases.yaml
  |   |-- tier_builder.py
  |   `-- tier_handler.py
  |-- cli
  |   |-- __init__.py
  |   |-- cli_base.py
  |   |-- commands
  |   |-- functest-complete.sh
  |   `-- setup.py
  |-- commons
  |   |-- ims
  |   |-- mobile
  |   `--traffic-profile-guidelines.rst
  |-- docker
  |   |-- Dockerfile
  |   |-- config_install_env.sh
  |   `-- requirements.pip
  |-- docs
  |   |-- com
  |   |-- configguide
  |   |-- devguide
  |   |-- images
  |   |-- release-notes
  |   |-- results
  |   `--userguide
  |-- testcases
  |   |-- Controllers
  |   |-- OpenStack
  |   |-- __init__.py
  |   |-- features
  |   |-- security_scan
  |   `-- vIMS
  `-- utils
      |-- __init__.py
      |-- functest_logger.py
      |-- functest_utils.py
      |-- openstack_clean.py
```

```
    |-- openstack_snapshot.py
    `-- openstack_utils.py

(Note: All *.pyc files removed from above list for brevity...)
```

We may distinguish 7 different directories:

- **ci**: This directory contains test structure defintion files (e.g <filename>.yaml) and bash shell/python scripts used to configure and execute Functional tests. The test execution script can be executed under the control of Jenkins CI jobs.
- **cli**: This directory holds the python based Functest CLI utility source code, which is based on the Python 'click' framework.
- **commons**: This directory is dedicated for storage of traffic profile or any other test inputs that could be reused by any test project.
- **docker**: This directory includes the needed files and tools to build the Funtest Docker container image.
- **docs**: This directory includes documentation: Release Notes, User Guide, Configuration Guide and Developer Guide. Test results are also located in a sub–directory called 'results'.
- **testcases**: This directory includes the scripts required by Functest internal test cases and other feature projects test cases.
- **utils**: this directory holds Python source code for some general purpose helper utilities, which testers can also re-use in their own test code. See for an example the Openstack helper utility: 'openstack_utils.py'.

After the *run* command, a new prompt appears which means that we are inside the container and ready to move to the next step.

Useful Docker commands

When typing **exit** in the container prompt, this will cause exiting the container and probably stopping it. When stopping a running Docker container all the changes will be lost, there is a keyboard shortcut to quit the container without stopping it: CTRL+P+Q. To reconnect to the running container **DO NOT** use the *run* command again (since it will create a new container), use the *exec* command instead:

```
docker ps <copy the container ID> docker exec -ti \
<CONTAINER_ID> /bin/bash
```

or simply:

```
docker exec -ti \
$(docker ps|grep functest|awk '{print $1}') /bin/bash
```

There are other useful Docker commands that might be needed to manage possible issues with the containers.

List the running containers:

```
docker ps
```

List all the containers including the stopped ones:

```
docker ps -a
```

It is useful sometimes to remove a container if there are some problems:

```
docker rm <CONTAINER_ID>
```

Use the *-f* option if the container is still running, it will force to destroy it:

```
docker -f rm <CONTAINER_ID>
```

The Docker image is called **opnfv/functest** and it is stored in the public Docker registry under the OPNFV account: dockerhub. The are many different tags that have been created automatically by the CI mechanisms, but the one that this document refers to is **brahmaputra.1.0**. Pulling other tags might cause some problems while running the tests.

Check the Docker documentation dockerdocs for more information.

Preparing the Functest environment

Once the Functest docker container is up and running, the required Functest environment needs to be prepared. A custom built **functest** CLI utility is availabe to perform the needed environment preparation action. Once the enviroment is prepared, the **functest** CLI utility can be used to run different functional tests. The usage of the **functest** CLI utility to run tests is described further in the Functest User Guide OPNFV_FuncTestUserGuide

Prior to commencing the Functest environment preparation, we can check the initial status of the environment. Issue the **functest env status** command at the prompt:

```
functest env status
Functest environment is not installed.

Note: When the Funtest environment is prepared, the command will
return the status: "Functest environment ready to run tests."
```

To prepare the Functest docker container for test case execution, issue the **functest env prepare** command at the prompt:

```
functest env prepare
```

This script will make sure that the requirements to run the tests are met and will install the needed libraries and tools by all Functest test cases. It should be run only once every time the Functest docker container is started from scratch. If you try to run this command, on an already prepared enviroment, you will be prompted whether you really want to continue or not:

```
functest env prepare
It seems that the environment has been already prepared.
Do you want to do it again? [y|n]

(Type 'n' to abort the request, or 'y' to repeat the
 environment preparation)
```

To list some basic information about an already prepared Functest docker container environment, issue the **functest env show** at the prompt:

```
functest env show
+======================================================+
| Functest Environment info                            |
+======================================================+
|  INSTALLER: apex, 192.168.122.89                     |
|   SCENARIO: os-odl_l2-nofeature-ha                   |
|        POD: localhost                                |
| GIT BRANCH: master                                   |
|   GIT HASH: 5bf1647dec6860464eeb082b2875798f0759aa91 |
| DEBUG FLAG: false                                    |
+------------------------------------------------------+
|     STATUS: ready                                    |
+------------------------------------------------------+
Where:

INSTALLER:  Displays the INSTALLER_TYPE value
            - here = "apex"
            and the INSTALLER_IP value
            - here = "192.168.122.89"
SCENARIO:   Displays the DEPLOY_SCENARIO value
            - here = "os-odl_l2-nofeature-ha"
POD:        Displays the value pass in NODE_NAME
            - here = "loclahost"
GIT BRANCH: Displays the git branch of the OPNFV Functest
            project repository included in the Functest
            Docker Container.
            - here = "master"
                     (In first official colorado release
                      would be "colorado.1.0")
GIT HASH:   Displays the git hash of the OPNFV Functest
            project repository included in the Functest
            Docker Container.
            - here = "5bf1647dec6860464eeb082b2875798f0759aa91"
DEBUG FLAG: Displays the CI_DEBUG value
            - here = "false"

NOTE: In Jenkins CI runs, an additional item "BUILD TAG"
      would also be listed. The valaue is set by Jenkins CI.
```

Finally, the **functest** CLI has a basic 'help' system with so called **–help** options:

Some examples:

```
functest --help Usage: functest [OPTIONS] COMMAND [ARGS]...

Options:
  --version  Show the version and exit.
  -h, --help Show this message and exit.

Commands:
  env
  openstack
  testcase
  tier

functest env --help
Usage: functest env [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help Show this message and exit.

Commands:
  prepare  Prepares the Functest environment.
  show     Shows information about the current...
  status   Checks if the Functest environment is ready...
```

Focus on the OpenStack credentials

The OpenStack credentials are needed to run the tests against the VIM. There are 3 ways to provide them to Functest:

- using the -v option when running the Docker container

- create an empty file in '/home/opnfv/functest/conf/openstack.creds' and paste the credentials into it. (Consult your installer guide to know from where you can retrieve credential files, which are set-up in the Openstack installation of the SUT)
- automatically retrieved using the following script:

```
    $repos_dir/releng/utils/fetch_os_creds.sh \
    -d /home/opnfv/functest/conf/openstack.creds \
    -i fuel \
    -a 10.20.0.2"

    (-d specifies the full destination path where to place the
        copied Openstack credential file
     -i specifies the INSTALLER_TYPE
     -a specifies the INSTALLER_IP
     If the installer is of type "fuel" and a Virtualized
     deployment is used, then this should be indicated by
     adding an option '-v'. The -v option takes no arguments.
     It enables some needed special handling in the script.)

    Note: If you omit the -d <full destination path> option in
    the command invocation, then the script will create the
    credential file with name 'opnfv-openrc.sh' in directory
    '/home/opnfv'. In that case, you need to copy/edit the file
    into the correct target path:
    '/home/opnfv/functest/conf/openstack.creds'.
```

Warning If you are using the Joid installer, the 'fetch_os_cred-sh' shell script **should not be used**. Use instead, the **-v** optin to Bind Mount a suitably prepared local copy of the Openstack credentials for usage by the Functest docker container

Once the credentials are there, they should be sourced **before** running the tests:

```
source /home/opnfv/functest/conf/openstack.creds
```

or simply using the environment variable **creds**:

```
. $creds
```

After this, try to run any OpenStack command to see if you get any output, for instance:

```
openstack user list
```

This will return a list of the actual users in the OpenStack deployment. In any other case, check that the credentials are sourced:

```
env|grep OS_
```

This command must show a set of environment variables starting with *OS_*, for example:

```
OS_REGION_NAME=RegionOne
OS_DEFAULT_DOMAIN=default
OS_PROJECT_NAME=admin
OS_PASSWORD=admin
OS_AUTH_STRATEGY=keystone
OS_AUTH_URL=http://172.30.10.3:5000/v2.0
OS_USERNAME=admin
OS_TENANT_NAME=admin
OS_ENDPOINT_TYPE=internalURL
OS_NO_CACHE=true
```

If the OpenStack command still does not show anything or complains about connectivity issues, it could be due to an incorrect url given to the OS_AUTH_URL environment variable. Check the deployment settings.

SSL Support

If you need to connect to a server that is TLS-enabled (the auth URL begins with 'https') and it uses a certificate from a private CA or a self-signed certificate, then you will need to specify the path to an appropriate CA certificate to use, to validate the server certificate with the environment variable OS_CACERT:

```
echo $OS_CACERT
/etc/ssl/certs/ca.crt
```

However, this certificate does not exist in the container by default. It has to be copied manually from the OpenStack deployment. This can be done in 2 ways:

1. Create manually that file and copy the contents from the OpenStack controller.
2. (Recommended) Add the file using a Docker volume when starting the container:

```
    -v <path_to_your_cert_file>:/etc/ssl/certs/ca.cert
```

You might need to export OS_CACERT environment variable inside the container:

```
export OS_CACERT=/etc/ssl/certs/ca.crt
```

Certificate verification can be turned off using OS_INSECURE=true. For example, Fuel uses self-signed cacerts by default, so an pre step would be:

```
export OS_INSECURE=true
```

Proxy support

If your Jumphost node is operating behind a http proxy, then there are 2 places where some special actions may be needed to make operations succeed:

1. Initial installation of docker engine First, try following the official Docker documentation for Proxy settings. Some issues were experienced on CentOS 7 based Jumphost. Some tips are documented in section: *Docker Installation on CentOS 7 behind http proxy* below.
2. Execution of the Functest environment preparation inside the created docker container Functest needs internet access to download some resources for some test cases. For example to install the Rally environment. This might not work properly if the Jumphost is running through a http Proxy.

If that is the case, make sure the resolv.conf and the needed http_proxy and https_proxy environment variables, as well as the 'no_proxy' environment variable are set correctly:

```
# Make double sure that the 'no_proxy=...' line in the
# 'openstack.creds' file is commented out first. Otherwise, the
# values set into the 'no_proxy' environment variable below will
# be ovewrwritten, each time the command
# 'source ~/functest/conf/openstack.creds' is issued.

sed -i 's/export no_proxy/#export no_proxy/' \
~/functest/conf/openstack.creds

source ~/functest/conf/openstack.creds

# Next calculate some IP addresses for which http_proxy
# usage should be excluded:

publicURL_IP=$(echo $OS_AUTH_URL| \
grep -Eo "([0-9]+\.){3}[0-9]+")

adminURL_IP=$(openstack catalog show identity | \
grep adminURL | grep -Eo "([0-9]+\.){3}[0-9]+")

export http_proxy="<your http proxy settings>"
export https_proxy="<your httpsproxy settings>"
export no_proxy="127.0.0.1,localhost,$publicURL_IP,$adminURL_IP"

# Ensure that "git" uses the http_proxy
# This may be needed if your firewall forbids SSL based git fetch
git config --global http.sslVerify True
git config --global http.proxy <Your http proxy settings>
```

Validation check: Before running **'functest env prepare'** CLI command, make sure you can reach http and https sites from inside the Functest docker container.

For example, try to use the **nc** command from inside the functest docker container:

```
nc -v google.com 80
Connection to google.com 80 port [tcp/http] succeeded!

nc -v google.com 443
Connection to google.com 443 port [tcp/https] succeeded!
```

Note: In a Jumphost node based on the CentOS 7, enviroment, it was observed that the **nc** commands did not function as described in the section above. You can however try using the **curl** command instead, if you encounter any issues with the **nc** command:

```
curl http://www.google.com:80

<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE>
</HEAD>
<BODY>
<H1>302 Moved</H1>
:
:
</BODY></HTML>

curl https://www.google.com:443

<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE>
</HEAD>
<BODY>
<H1>302 Moved</H1>
:
:
</BODY></HTML>

(Even Google complained the URL used, it proves the http and https
 protocols are working correctly through the http / https proxy.)
```

Docker Installation on CentOS 7 behind http proxy

There are good instructions in [InstallDockerCentOS7] for the installation of **docker** on CentOS 7. However, if your Jumphost is behind a http proxy, then the following steps are needed **before** following the instructions in the above reference:

```
1) # Make a directory '/etc/systemd/system/docker.service.d'
   # if it does not exist
   sudo mkdir /etc/systemd/system/docker.service.d

   # Create a file called 'env.conf' in that directory with
   # the following contents:
   [Service]
   EnvironmentFile=-/etc/sysconfig/docker

2) # Set up a file called 'docker' in directory '/etc/sysconfig'
   # with the following contents:

   HTTP_PROXY="<Your http proxy settings>"
   HTTPS_PROXY="<Your https proxy settings>"
   http_proxy="${HTTP_PROXY}"
   https_proxy="${HTTPS_PROXY}"

3) # Reload the daemon
   systemctl daemon-reload

4) # Sanity check - check the following docker settings:
   systemctl show docker | grep -i env

   Expected result:
   ----------------
   EnvironmentFile=/etc/sysconfig/docker (ignore_errors=yes)
   DropInPaths=/etc/systemd/system/docker.service.d/env.conf
```

Now follow the instructions in [InstallDockerCentOS7] to download and install the **docker-engine**. The instructions conclude with a “test pull” of a sample “Hello World” docker container. This should now work with the above pre-requisite actions.

5.3.2 Installing vswitchperf

Supported Operating Systems

- CentOS 7
- Fedora 20
- Fedora 21
- Fedora 22
- RedHat 7.2
- 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 official VM image is called vloop-vnf and it is available for free download at OPNFV website.

vloop-vnf changelog:

- vloop-vnf-ubuntu-14.04_20160303
  - snmpd service is disabled by default to avoid error messages during VM boot
  - security updates applied
- vloop-vnf-ubuntu-14.04_20151216
  - version with development tools required for build of DPDK and l2fwd

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). In case of RedHat 7 it will install Python 3.4 as an alternate installation in /usr/local/bin. 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, RedHat and Ubuntu

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

Gotcha Check what type of shell you are using

See what scripts are available in $HOME/vsperfenv/bin

source the appropriate script

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
```

Hugepage Configuration

Systems running vsperf with either dpdk and/or tests with guests must configure hugepage amounts to support running these configurations. It is recommended to configure 1GB hugepages as the pagesize.

The amount of hugepages needed depends on your configuration files in vsperf. Each guest image requires 4096 by default according to the default settings in the `04_vnf.conf` file.

```
GUEST_MEMORY = ['4096', '4096']
```

The dpdk startup parameters also require an amount of hugepages depending on your configuration in the `02_vswitch.conf` file.

```
VSWITCHD_DPDK_ARGS = ['-c', '0x4', '-n', '4', '--socket-mem 1024,1024']
VSWITCHD_DPDK_CONFIG = {
    'dpdk-init' : 'true',
    'dpdk-lcore-mask' : '0x4',
    'dpdk-socket-mem' : '1024,1024',
}
```

Note: Option VSWITCHD_DPDK_ARGS is used for vswitchd, which supports –dpdk parameter. In recent vswitchd versions, option VSWITCHD_DPDK_CONFIG will be used to configure vswitchd via ovs-vsctl calls.

With the –socket-mem argument set to use 1 hugepage on the specified sockets as seen above, the configuration will need 9 hugepages total to run all tests within vsperf if the pagesize is set correctly to 1GB.

Depending on your OS selection configuration of hugepages may vary. Please refer to your OS documentation to set hugepages correctly. It is recommended to set the required amount of hugepages to be allocated by default on reboots.

Information on hugepage requirements for dpdk can be found at http://dpdk.org/doc/guides/linux_gsg/sys_reqs.html

You can review your hugepage amounts by executing the following command

```
cat /proc/meminfo | grep Huge
```

5.3.3 Yardstick

The project's goal is to verify infrastructure compliance, from the perspective of a Virtual Network Function (VNF).

The Project's scope is the development of a test framework, *Yardstick*, test cases and test stimuli to enable Network Function Virtualization Infrastructure (NFVI) verification.

In OPNFV Brahmaputra release, generic test cases covering aspects of the metrics in the document ETSI GS NFV-TST001, "Pre-deployment Testing; Report on Validation of NFV Environments and Services" are available; further OPNFV releases will provide extended testing of these metrics.

The Project also includes a sample VNF, the Virtual Traffic Classifier (VTC) and its experimental framework, *ApexLake*.

Yardstick is used in OPNFV for verifying the OPNFV infrastructure and some of the OPNFV features. The *Yardstick* framework is deployed in several OPNFV community labs. It is *installer*, *infrastructure* and *application* independent.

See also:

This Presentation for an overview of *Yardstick* and Yardsticktst for material on alignment ETSI TST001 and Yardstick.

Yardstick Installation

Abstract

Yardstick currently supports installation on Ubuntu 14.04 or by using a Docker image. Detailed steps about installing Yardstick using both of these options can be found below.

To use Yardstick you should have access to an OpenStack environment, with at least Nova, Neutron, Glance, Keystone and Heat installed.

The steps needed to run Yardstick are:

1. Install Yardstick and create the test configuration .yaml file.
2. Build a guest image and load the image into the OpenStack environment.
3. Create a Neutron external network and load OpenStack environment variables.
4. Run the test case.

Installing Yardstick on Ubuntu 14.04

Installing Yardstick framework Install dependencies:

```
sudo apt-get update && sudo apt-get install -y \
    wget \
    git \
    sshpass \
    qemu-utils \
    kpartx \
    libffi-dev \
    libssl-dev \
    python \
    python-dev \
    python-virtualenv \
    libxml2-dev \
    libxslt1-dev \
    python-setuptools
```

Create a python virtual environment, source it and update setuptools:

```
virtualenv ~/yardstick_venv
source ~/yardstick_venv/bin/activate
easy_install -U setuptools
```

Download source code and install python dependencies:

```
git clone https://gerrit.opnfv.org/gerrit/yardstick
cd yardstick
python setup.py install
```

There is also a YouTube video, showing the above steps:

Installing extra tools

yardstick-plot Yardstick has an internal plotting tool `yardstick-plot`, which can be installed using the following command:

```
sudo apt-get install -y g++ libfreetype6-dev libpng-dev pkg-config
python setup.py develop easy_install yardstick[plot]
```

Building a guest image Yardstick has a tool for building an Ubuntu Cloud Server image containing all the required tools to run test cases supported by Yardstick. It is necessary to have sudo rights to use this tool.

Also you may need install several additional packages to use this tool, by follwing the commands below:

```
apt-get update && apt-get install -y \
    qemu-utils \
    kpartx
```

This image can be built using the following command while in the directory where Yardstick is installed (`~/yardstick` if the framework is installed by following the commands above):

```
sudo ./tools/yardstick-img-modify tools/ubuntu-server-cloudimg-modify.sh
```

Warning: the script will create files by default in: `/tmp/workspace/yardstick` and the files will be owned by root!

The created image can be added to OpenStack using the `glance image-create` or via the OpenStack Dashboard.

Example command:

```
glance --os-image-api-version 1 image-create \
--name yardstick-trusty-server --is-public true \
--disk-format qcow2 --container-format bare \
--file /tmp/workspace/yardstick/yardstick-trusty-server.img
```

Installing Yardstick using Docker

Yardstick has two Docker images, first one (**Yardstick-framework**) serves as a replacement for installing the Yardstick framework in a virtual environment (for example as done in *Installing Yardstick framework*), while the other image is mostly for CI purposes (**Yardstick-CI**).

Yardstick-framework image Download the source code:

```
git clone https://gerrit.opnfv.org/gerrit/yardstick
```

Build the Docker image and tag it as *yardstick-framework*:

```
cd yardstick
docker build -t yardstick-framework .
```

Run the Docker instance:

```
docker run --name yardstick_instance -i -t yardstick-framework
```

To build a guest image for Yardstick, see *Building a guest image*.

Yardstick-CI image Pull the Yardstick-CI Docker image from Docker hub:

```
docker pull opnfv/yardstick:$DOCKER_TAG
```

Where `$DOCKER_TAG` is latest for master branch, as for the release branches, this coincides with its release name, such as brahmaputra.1.0.

Run the Docker image:

```
docker run \
 --privileged=true \
  --rm \
  -t \
  -e "INSTALLER_TYPE=${INSTALLER_TYPE}" \
  -e "INSTALLER_IP=${INSTALLER_IP}" \
  opnfv/yardstick \
  exec_tests.sh ${YARDSTICK_DB_BACKEND} ${YARDSTICK_SUITE_NAME}
```

Where `${INSTALLER_TYPE}` can be apex, compass, fuel or joid, `${INSTALLER_IP}` is the installer master node IP address (i.e. 10.20.0.2 is default for fuel). `${YARDSTICK_DB_BACKEND}` is the IP and port number of DB, `${YARDSTICK_SUITE_NAME}` is the test suite you want to run. For more details, please refer to the Jenkins job defined in Releng project, labconfig information and sshkey are required. See the link https://git.opnfv.org/cgit/releng/tree/jjb/yardstick/yardstick-ci-jobs.yml.

Note: exec_tests.sh is used for executing test suite here, furthermore, if someone wants to execute the test suite manually, it can be used as long as the parameters are configured correct. Another script called run_tests.sh is used for unittest in Jenkins verify job, in local manaul environment, it is recommended to run before test suite execuation.

Basic steps performed by the **Yardstick-CI** container:

1. clone yardstick and releng repos
2. setup OS credentials (releng scripts)
3. install yardstick and dependencies
4. build yardstick cloud image and upload it to glance
5. upload cirros-0.3.3 cloud image to glance

6. run yardstick test scenarios
7. cleanup

OpenStack parameters and credentials

Yardstick-flavor Most of the sample test cases in Yardstick are using an OpenStack flavor called *yardstick-flavor* which deviates from the OpenStack standard m1.tiny flavor by the disk size - instead of 1GB it has 3GB. Other parameters are the same as in m1.tiny.

Environment variables Before running Yardstick it is necessary to export OpenStack environment variables from the OpenStack *openrc* file (using the `source` command) and export the external network name `export EXTERNAL_NETWORK="external-network-name"`, the default name for the external network is `net04_ext`.

Credential environment variables in the *openrc* file have to include at least:

- OS_AUTH_URL
- OS_USERNAME
- OS_PASSWORD
- OS_TENANT_NAME

Yardstick default key pair Yardstick uses a SSH key pair to connect to the guest image. This key pair can be found in the `resources/files` directory. To run the `ping-hot.yaml` test sample, this key pair needs to be imported to the OpenStack environment.

Examples and verifying the install

It is recommended to verify that Yardstick was installed successfully by executing some simple commands and test samples. Below is an example invocation of yardstick help command and ping.py test sample:

```
yardstick -h
yardstick task start samples/ping.yaml
```

Each testing tool supported by Yardstick has a sample configuration file. These configuration files can be found in the **samples** directory.

Example invocation of `yardstick-plot` tool:

```
yardstick-plot -i /tmp/yardstick.out -o /tmp/plots/
```

Default location for the output is `/tmp/yardstick.out`.

More info about the tool can be found by executing:

```
yardstick-plot -h
```