Intel® Open Network Platform Server Release 1.5 Reference Architecture Guide

SDN/NFV Solutions with Intel® Open Network Platform Server

Document Revision 1.0
September 2015

Revision History

Date	Revision	Comments
September 30, 2015	1.0	Initial release of Intel® Open Network Platform Server Release 1.5

Contents

Figures

Tables

1.0 Audience and Purpose

The primary audiences for this document are architects and engineers implementing the Intel® Open Network Platform Server Reference Architecture using open-source software. Software ingredients include the following:

- DevStack*
- OpenStack*
- OpenDaylight*
- Data Plane Development Kit (DPDK)*
- Regular Open vSwitch*
- Open vSwitch* with DPDK-netdev
- Fedora*

This document provides a guide for integration using the Intel® Open Network Platform Server (Intel ONP Server). Content includes high-level architecture, setup and configuration procedures, integration learnings, and a set of baseline performance data. This information is intended to help architects and engineers evaluate Network Functions Virtualization (NFV) and Software Defined Networking (SDN) solutions.

The purpose of documenting configurations is not to imply any preferred methods. Providing a baseline configuration of well-tested procedures, however, can help achieve optimal system performance on an IA Platform when developing an NFV/SDN solution.

2.0 Summary

The Intel ONP Server uses open-source software to help accelerate SDN and NFV commercialization with the latest Intel Architecture Communications Platform.

This document describes how to set up and configure the controller and compute nodes for evaluating and developing NFV/SDN solutions using Intel® Open Network Platform ingredients.

Platform hardware is based on an Intel® Xeon® DP Server with the following hardware:

- Intel® dual Xeon® Processor Series E5-2600 V3
- Intel® X710 4x10 GbE and XL710 2x40 GbE adapters

The host operating system is Fedora* 21 with QEMU-KVM virtualization technology. Software ingredients include DPDK, Open vSwitch, Open vSwitch with DPDK-netdev, OpenStack, and OpenDaylight (ODL). Figure 2–1 shows the corresponding version information for the components involved. For the list of new features and improvements, see *Intel® ONP Server Release 1.5 Release Notes*, section 3, available on 01.org.

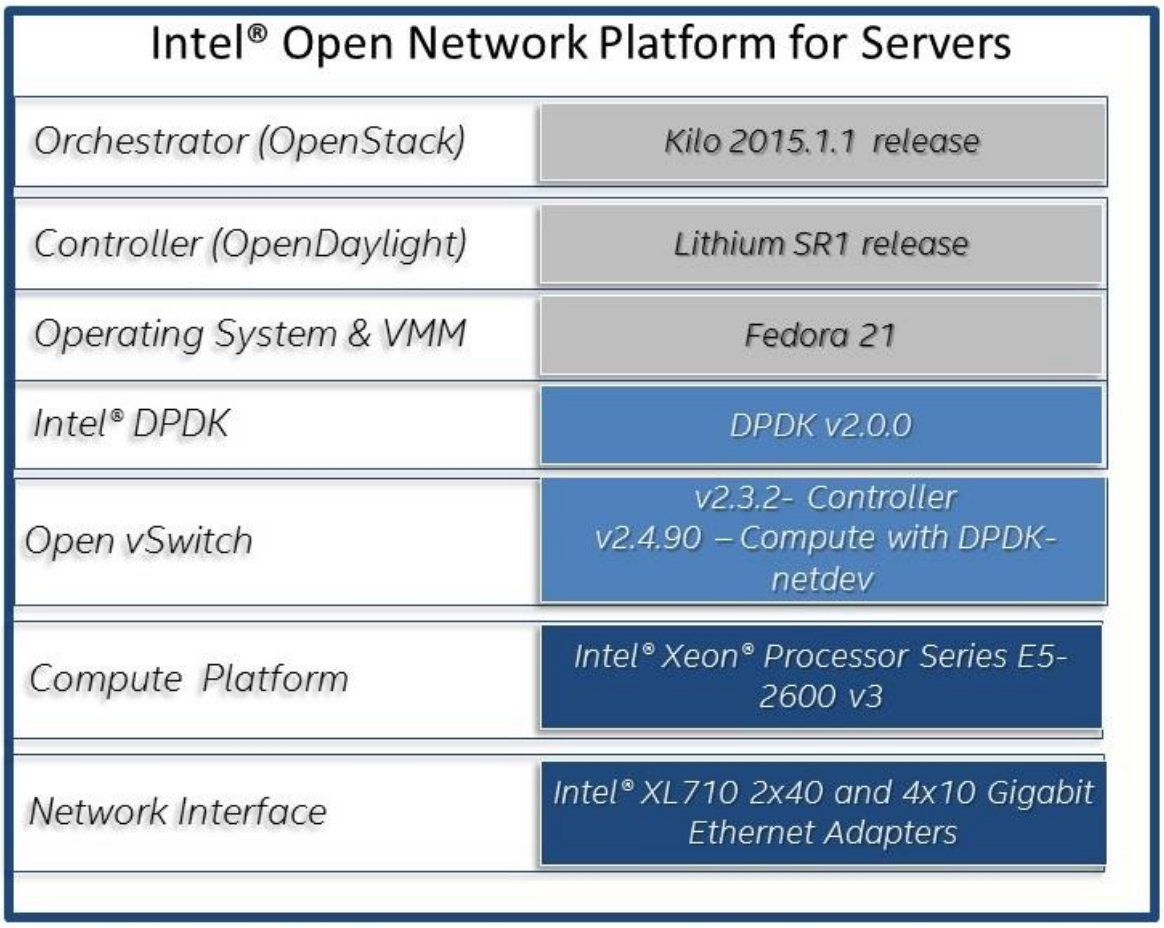


Figure 2–1. Intel ONP Server — Hardware and Software Ingredients

The test cases described in this document are designed to illustrate functionality using the specified ingredients, configurations, and test methodology.

The test cases documented are designed to:

- Verify communication between the controller and compute nodes
- Validate basic controller functionality
- Demonstrate how to deploy virtualized network functions (VNFs) and showcase traffic processed with in them.

By doing this, we hope to show users how they can use the "plumbing" that is put in place with great efficiency and optimized performance for Intel Architecture to deploy their VMs and VNFs in a way that enables more CPU cycles to be dedicated to run these VMs and VNFs, instead of pushing network packets to and from physical and virtual networks.

Figure 2–2 shows a generic SDN/NFV setup. In this configuration, the orchestrator and controller (management and control plane) run on one server, and the two compute nodes (data plane) run on two individual server nodes. The differences in network configuration to enable this setup are shown with the management and data ports. Note that many variations of this setup can be deployed.

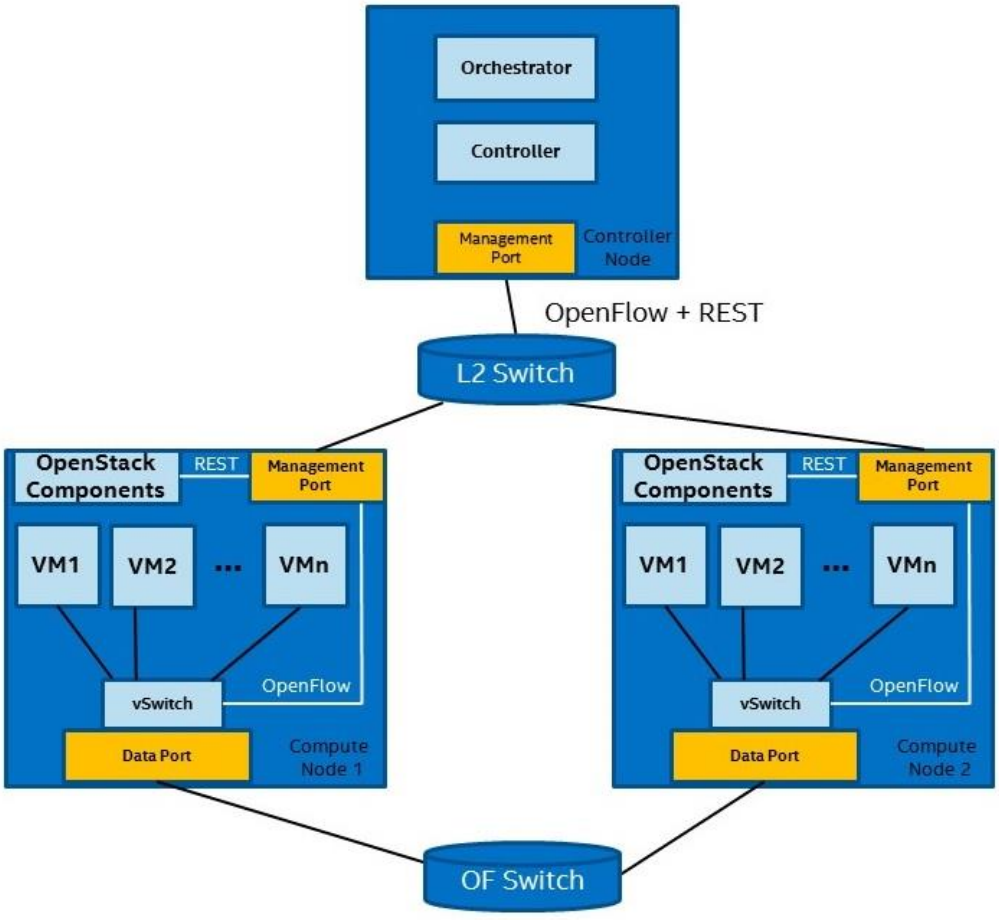


Figure 2–2. Generic Setup with Controller and Two Compute Nodes

2.1 Network Services Examples

The following network services in this section are included as examples that have been tested with the Intel® ONP Server Reference Architecture. They are demonstrated as use cases running as virtualized instances deployed and controlled by OpenStack.

2.1.1 Suricata (Next Generation IDS/IPS Engine)

Suricata is a high-performance network intrusion detection system (IDS), intrusion prevention system (IPS), and network security monitoring engine developed by the Open Information Security Foundation, its supporting vendors, and the community. Refer to the following URL: http://suricata-ids.org/.

2.1.2 vBNG (Broadband Network Gateway)

Intel Data Plane Performance Demonstrators — Broadband (or Border) Network Gateway (BNG) using DPDK can be found at: https://01.org/intel-data-plane-performance-demonstrators/downloads/bng-application-v017.

A BNG may also be known as a Broadband Remote Access Server (BRAS) and routes traffic to and from broadband remote access devices, such as digital subscriber line access multiplexers. This network function is included as an example of a workload that can be virtualized on the Intel ONP Server.

Additional information on the performance characterization of this vBNG implementation can be found at https://networkbuilders.intel.com/docs/Network_Builders_RA_vBRAS_Final.pdf.

Refer to section 7.3, Installation and Configuration of the vBNG, or Appendix B, Configuring the Proxy, for more information on running BNG as an appliance.

3.0 Hardware Components

Table 3–1 provides details of the platform hardware components used for testing purposes. The "Notes" column details some of the fine tunings enabled on the hardware.

Table 3–1. Hardware Ingredients (Code-named Wildcat Pass)

Item	Description	Notes
Platform	Intel® Server Board S2600WTT 1100 W power supply	Intel® Xeon® processor-based DP server (Formerly code-named Wildcat Pass) 120 GB SSD 2.5in SATA 6GB/s Intel Wolfsville SSDSC2BB120G4; supports SR-IOV.
Processors	Intel® Dual Xeon® Processor E5-2697 V3, 2.6 GHz, 35 MB, 145 W, 14 cores	(Formerly code-named Haswell) 14 cores, 2.60 GHz, 145 W, 35 MB total cache per processor, 9.6 GT/s QPI, DDR4-1600/1866/2133, 28 hyper-threaded cores per CPU for 56 total cores.
	Intel® Dual Xeon® Processor Series E5-2699 v3 2.3 GHz, 45 MB, 145 W, 18 cores	(Formerly code-named Haswell) 18 cores, 2.3 GHz, 145 W, 45 MB total cache per processor, 9.6 GT/s QPI, DDR4-1600/1866/2133, 36 hyper-threaded cores per CPU for 72 total cores.
Memory	8 GB DDR4 RDIMM Crucial CT8G4RFS423	64 GB RAM (8 x 8 GB)
NICs (XL710)	Intel® Ethernet Controller XL710-AM1 4x10GbE, Firmware v.f4.33 a1.2 n04.42	(Code-named Fortville) NICs are on socket zero.
	Intel® Ethernet Controller XL710-AM2 2x40GbE, Firmware v.f4.33 a1.2 n04.42	(Code-named Fortville) NICs are on socket zero.
BIOS	SE5C610.86B.01.01.0008.031920151331 Release Date: 03/19/2015	Intel® Virtualization Technology for direct I/O (Intel® VT-d), hyper-threading enabled.

4.0 Software Versions

Table 4–1 describes the function of the software ingredients involved along with their version or configuration. For open-source components, a specific commit ID set is used for this integration. Note that the commit IDs used are the latest working set at the time of this release.

Table 4–1. Software Versions

Software Component	Function	Version/Configuration
Fedora 21 x86_64	Host OS	Kernel version: 3.18.8-201.fc21.x86_64
Real-Time Kernel	Targeted towards the Telco environment, which is sensitive to low latency	Real-Time Kernel v3.18.16-rt13
QEMU-KVM	Virtualization technology	QEMU-KVM v2.3.0.5.fc21.x86_64 Libvirt 1.2.9.3-2.fc21.x86_64 (non-DPDK nodes) Libvirt 1.2.13.1-2.fc21.x86_64 (DPDK nodes)
DPDK	Network stack bypass and libraries for packet processing; includes user space vhost drivers	2.0.0
Open vSwitch	vSwitch	OVS: Open vSwitch v.2.3.2 (non-DPDK nodes) OVS with DPDK-netdev: Open vSwitch v.2.4.90
OpenStack	SDN orchestrator	Stable/Kilo 2015.1.1 Release
DevStack	Tool for OpenStack deployment	https://github.com/openstack-dev/devstack.git tag: stable/kilo
ODL	SDN controller	Lithium SR1
Suricata	IPS application	Suricata v2.0.2-1.fc20.x86_64

4.1 Obtaining Software Ingredients

All of the open-source software ingredients involved are downloaded from the source repositories shown in Table 4–2. Due to the number of ingredients involved, you will need to follow the instructions for each of the software components to be able to deploy them. Commit IDs are shown in Table 4–3.

Table 4–2. Software Ingredients

Software Component	Software Subcomponents	Patches	Location	Comments
Fedora 21			http://download.fedoraproject.org/pub/fedora/linux/releases/21/Server/x86_64/iso/Fedora-Server-DVD-x86_64-21.iso	Standard Fedora 21 Server ISO image.
Real-Time Kernel			git clone https://www.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git/ checkoutv3.18.16-rt13	Latest Real-Time Kernel
DPDK	DPDK poll mode driver, sample apps (bundled)		git clone http://dpdk.org/git/dpdk checkout v2.0.0	DPDK download will be done through the DevStack script during installation.
OpenvSwitch			git clone https://github.com/openvswitch.ovs.git checkout 1e77bbe565bbf5ae7f4c47f481a4097d666d3d68	OVS download will be done through the DevStack script during installation.
OpenStack			Kilo release: To be deployed using DevStack (see next row). The commit ids for the various Openstack components are provided in the Table 4-3.	
DevStack			DevStack: git clone https://github.com/openstack-dev/devstack.git checkout stable/kilo 4935abbcbb658b5a0f059fc8f2f2480b26e40554	
ODL			https://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/org/opendaylight/integration/distribution-karaf/0.3.1-SNAPSHOT/distribution-karaf-0.3.1-20150823.042817-365.zip	ODL download will be done through the DevStack script during installation.
Intel® ONP Server Release 1.5 Script	Helper scripts to set up SRT 1.4 using DevStack		https://download.01.org/packet-processing/ONPS1.5/onps_server_1_5.tar.gz	
Suricata			Suricata	yum install Suricata

Table 4–3. Commit IDs for OpenStack Components

OpenStack Component	Commit ID, Tag, or Branch
OpenStack Nova	stable/kilo 2015.1.1
OpenStack Neutron	stable/kilo 2015.1.1
OpenStack Keystone	stable/kilo 2015.1.1
OpenStack Glance	stable/kilo 2015.1.1
OpenStack Horizon	stable/kilo 2015.1.1
OpenStack Cinder	stable/kilo 2015.1.1
OpenStack Requirements	stable/kilo d9c926f831c869d3516d9a34bc54a538df20f158
OpenStack Tempest	master 5
OpenStack noVNC	master 8f3c0f6b9b5e5c23a7dc7e90bd22901017ab4fc7
OpenStack Networking-odl	stable/kilo 4f9f667774eb773e43cb71e6bc96c10951a4354
OpenStack Networking-ovs-dpdk	stable/kilo 2015.1.1

5.0 Installation and Configuration Guide

This section describes the installation and configuration instructions to prepare the controller and compute nodes.

5.1 Automated Installation Using Scripts

In order to go with the base set of Intel-recommended settings, using the scripts provided for installation is optimal. Many of the manual operating system and OpenStack installations with DevStack are automated. The bulk of this procedure is condensed into a few steps that can be executed using these scripts. All the ingredients listed in Table 4–1 can be installed using this method.

Before continuing with the scripts, update the BIOS settings using Table 5–1. You can download the scripts by clicking onps_server_1_5.tar.gz tarball. Follow the instructions listed in the README file for detailed execution steps of the scripts. Once completed, jump to section 6.0, VM Deployment Using OpenStack, for the next steps.

The option of enabling Real-Time kernel on the compute node is also provided in the scripts used for automated installation.

Note: The automation scripts are only tested on Fedora 21 OS and will not necessarily work on other flavors of Linux.

5.2 Manual Installation Procedure

This section provides instructions on how to manually prepare both the controller and compute nodes. Although it is considered relatively easy to use this solutions guide for other Linux distributions, the preferred operating system is Fedora 21. Details on the procedure common to the compute and controller nodes can be found in section 5.2.2.1, OpenStack (Kilo), to help provide a good start.

Note: It is assumed that commands requiring root privileges are listed beginning with '#' and commands requiring only user privileges are listed beginning with the '$' sign.

5.2.1 Instructions Common to the Controller and Compute Nodes

If you are looking to fine-tune a specific set of components either in the operating system or OpenStack, you can choose to follow the manual installation procedure. Update the BIOS settings as described in section 5.2.1.1, BIOS Settings.

5.2.1.1 BIOS Settings

During boot time, enter the BIOS menu and update the following configuration as described in Table 5–1. These settings are common to both the controller and compute nodes.

Table 5–1. BIOS Settings

Configuration	Controller Node Setting	Compute Node Setting
Intel® Virtualization Technology	Enabled	Enabled
Intel® Hyper-Threading Technology (HTT)	Enabled	Enabled
Intel® VT for Directed I/O (VT-d)	Enabled	Enabled

5.2.1.2 Operating System Installation and Configuration

The following are some generic instructions for installing and configuring the operating system. Other methods of installing the operating system, such as network installation, PXE boot installation, USB key installation, etc., are not described in this solutions guide.

Get the Fedora DVD

1. Download the 64-bit Fedora 21 DVD from the following site:

   https://getfedora.org/en/server/

   or directly from the URL:

   http://download.fedoraproject.org/pub/fedora/linux/releases/21/Server/x86_64/iso/Fedora-Server-DVD-x86_64-21.iso

2. Burn the ISO file to a DVD and create an installation disk.

Fedora 21 OS Installation

Use the DVD to install Fedora 21. During the installation, click **Software selection**, then choose the following:

1. C Development Tool and Libraries
2. Development Tools
3. Virtualization

Create a user named "stack" and check the box **Make this user administrator** during the installation. The user "stack" is also used in the OpenStack installation. Reboot the system after completing the installation.

Proxy Configuration

If your infrastructure requires you to configure the proxy server, follow the instructions in Appendix B, Configuring the Proxy.

Additional Packages Installation

Some packages are not installed with the standard Fedora 21 installation, but are required by Intel® ONPS components. These packages should be installed by the user:

```
# yum –y install git ntp patch socat python-passlib libxslt-devel libffi-devel fuse-devel gluster
```

Fedora 21 Kernel Upgrade

The ONP Server supports the upgraded kernel on Fedora 21 that comes with the 3.17.4 kernel.

Kernel 3.18.8-201 is newer than native Fedora 21 kernel 3.17.4. To upgrade from 3.17.4 to a newer kernel, download the target kernel package files from a repository, and then install the RPM packages. Below is an example of upgrading to the 3.18.8-201 kernel:

Note: If the Linux Real-Time kernel is preferred, you can skip this section and go to the Real-Time Kernel Compute Node Enablement section below.

1. Download the following kernel packages:

```
# wget
https://kojipkgs.fedoraproject.org/packages/kernel/3.18.8/201.fc21/x86_64/kernel-core-3.18.8-201.fc21.x86_64.rpm

# wget
https://kojipkgs.fedoraproject.org/packages/kernel/3.18.8/201.fc21/x86_64/kernel-modules-3.18.8-201.fc21.x86_64.rpm

# wget
https://kojipkgs.fedoraproject.org/packages/kernel/3.18.8/201.fc21/x86_64/kernel-3.18.8-201.fc21.x86_64.rpm

# wget
https://kojipkgs.fedoraproject.org/packages/kernel/3.18.8/201.fc21/x86_64/kernel-devel-3.18.8-201.fc21.x86_64.rpm
# wget
https://kojipkgs.fedoraproject.org/packages/kernel/3.18.8/201.fc21/x86_64/kernel-modules-extra-3.18.8-201.fc21.x86_64.rpm
```

2. Install the kernel packages:

```
# rpm -ihv kernel-core-3.18.8-201.fc21.x86_64.rpm
# rpm -ihv kernel-modules-3.18.8-201.fc21.x86_64.rpm
# rpm -ihv kernel-3.18.8-201.fc21.x86_64.rpm
# rpm -ihv kernel-devel-3.18.8-201.fc21.x86_64.rpm
# rpm -ihv kernel-modules-extra-3.18.8-201.fc21.x86_64.rpm
```

3. Reboot system to allow booting into the 3.18.8 kernel.

Note: ONPS depends on libraries provided by your Linux distribution. As such, it is recommended that you regularly update your Linux distribution with the latest bug fixes and security patches to reduce the risk of security vulnerabilities in your systems.

The following command upgrades to the latest kernel that Fedora supports. In order to maintain the specific kernel version (e.g., 3.18.8), the yum configuration file needs to be modified with this command prior to running the yum update:

```
# echo "exclude=kernel*" >> /etc/yum.conf
```

After installing the required kernel packages, the operating system should be updated with the following command:

```
# yum update -y
```

Reboot the system once the update is done.

Special Consideration for the Compute Node with OVS-DPDK

Fedora 21 comes with libvirt v1.2.9 even after the system updates to the latest packages through the yum repository. DPDK v2.0.0, however, requires a minimum version of 1.2.12 for libvirt. Two solutions can bridge this gap: download the higher version of the libvirt source files and then compile on the target system or use an available repository for a direct upgrade. We recommend using the Fedora virt-preview repository for this purpose.

To do this, run the following command:

```
# cd /etc/yum.repos.d/
# wget http://fedorapeople.org/groups/virt/virt-preview/fedora-virt-preview.repo
```

This adds a new repository, fedora-virt-preview, to the system. When performing a system update (i.e., yum update), libvirt upgrades to v1.2.13 (i.e., the current latest version from this repository).

Real-Time Kernel Compute Node Enablement

In real-time use cases, like the Telco environment, applications like media are sensitive to low latency and jitter, it makes sense to install the Linux Real-Time stable kernel on a compute node, instead of the standard Fedora kernel. This section provides the procedure to do this.

Note: The option of enabling the Real-Time kernel on the compute node is provided in the scripts used for automated installation.

In case automated installation scripts are not used, the procedure for manually enabling the Real-Time kernel on the compute node is as follows:

1. Select the following software to install the Fedora 21 operating system:
   - Virtualization
   - C Development Tools
   - Development Tools
2. After the installation is done, reboot the system.
3. The inbox kernel is 3.17.4-301.fc21.x86_64. Prepare the system to define the network interface by disabling the firewall and Network Manager (refer to the "Disable and Enable Services" section below for details), and then perform the following steps:
   a. Run `yum update -y` to update the system with the latest libraries.
   b. Reboot the system.
4. Install the stable Real-Time kernel:
   a. Get the Real-Time kernel sources:

```
# cd /usr/src/kernels
# git clone https://www.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git
```

   Note: It may take a while to complete the download.

   b. Find the latest rt version from the git tag and then check out the version:

```
# cd linux-stable-rt
# git checkout v3.18.16-rt13
```

   Note: v3.18.16-rt13 is the latest version when this document was written.

   c. Compile the Real-Time kernel:

   # NOTE: Reference this link:

   https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

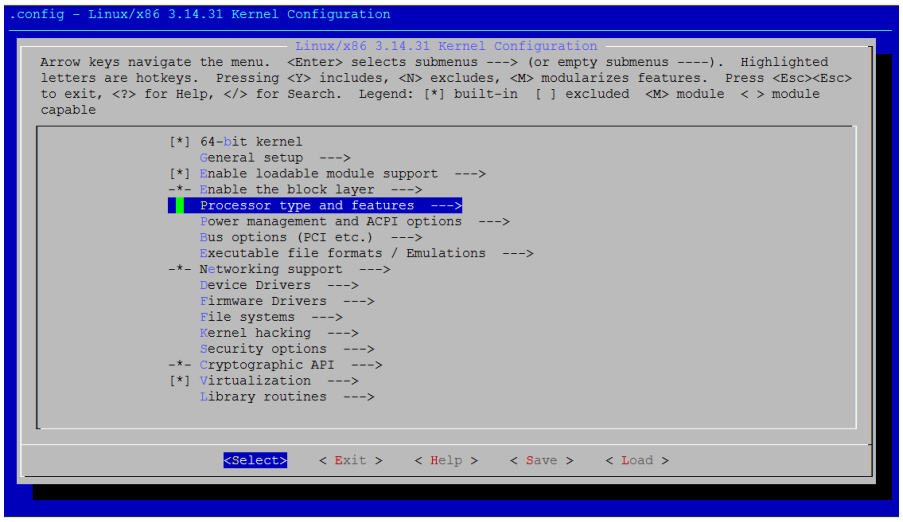
   d. Install the necessary package:

```
# yum install ncurses-devel
```

   e. Copy the kernel configuration file to the kernel source:

```
# cp /usr/src/kernel/3.17.4-301.f21.x86_64/.config /usr/src/kernel/linux-stable-rt/
# cd /usr/src/kernel/linux-stable-rt
# make menuconfig
```

The resulting configuration interface is shown below.



f. Select the following:

   i. Enable the high resolution timer:

      General Setup > Timers Subsystem > High Resolution Timer Support
      (This option is selected by default.)

   ii. Enable the max number SMP:

      Processor type and features > Enable Maximum Number of SMP Processor and NUMA Nodes

   iii. Enable PREEMPT-RT:

      Processor type and features > Preemption Model > Fully Pre-emptible Kernel (RT)

   iv. Set the high-timer frequency:

      Processor type and features > Timer frequency > 1000 HZ
      (This option is selected by default.)

   v. Exit and save.

   vi. Compile the kernel:

```
# make –j `grep –c processor /proc/cpuinfo` && make modules_install &&
make install
```

g. Make changes to the boot sequence:

   i. To show all menu entries:

```
# grep ^menuentry /boot/grub2/grub.cfg
```

   ii. To set the default menu entry:

```
# grub2-set-default "the desired default menu entry"
```

iii. To verify:

```
# grub2-editenv list
```

iv. Reboot and log to the new Real-Time kernel version: v3.18.16-rt13.

Disable and Enable Services

For OpenStack, the default settings for a set of services need to be enabled/disabled or changed. The following services need to be changed or disabled: SELinux, firewall, and NetworkManager.

To do this, run the following commands:

```
# sed –i 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/selinux/config
# systemctl disable firewalld.service
# systemctl disable NetworkManager.service
```

The following services should be enabled: ntp, sshd, and network.

To do this, run the following commands:

```
# systemctl enable ntpd.service
# systemctl enable ntpdate.service
# systemctl enable sshd.service
# chkconfig network on
```

It is necessary to keep the timing synchronized between all nodes and to use a known Network Time Protocol (NTP) server for of them. If this is not done, there will be synchronization issues between the OpenStack controller and compute nodes. You can use either the standard NTP server provided by Fedora or edit `etc/ntp.conf` to add a new server and remove the default ones.

The following example replaces a default NTP server with a local NTP server `10.166.45.16` and comments out other default servers:

```
# sed –i 's/server 0.fedora.pool.ntp.org iburst/server 10.166.45.16/g' /etc/ntp.conf
# sed –i 's/server 1.fedora.pool.ntp.org iburst/# server 1.fedora.pool.ntp.org
iburst
/ g' /etc/ntp.conf
# sed –i 's/server 2.fedora.pool.ntp.org iburst/# server 2.fedora.pool.ntp.org
iburst
/ g' /etc/ntp.conf
# sed –i 's/server 3.fedora.pool.ntp.org iburst/# server 3.fedora.pool.ntp.org
iburst
/ g' /etc/ntp.conf
```

Now that the base operating system is installed and the right services are configured, you can proceed to the individual node setup for either the controller or compute node.

5.2.2 Controller Node Setup

This section describes the controller node setup. It is assumed that the user successfully followed the operating system installation and configuration sections.

5.2.2.1 OpenStack (Kilo)

This section documents the configurations that need to be made and the installation of OpenStack on the controller node.

Network Requirements

If your infrastructure requires you to configure the proxy server, follow the instructions in Appendix B, Configuring the Proxy.

General

At least two networks are required to build the OpenStack infrastructure in a lab environment. One network is used to connect all nodes for OpenStack management (management network); the other is a private network exclusively for an OpenStack internal connection (private or tenant network) between instances (or VMs).

Some users might want to have Internet and/or external connectivity for OpenStack instances (VMs). In this case, an optional network (public network) can be used.

One additional network is required for Internet connectivity, because installing OpenStack requires pulling packages from various sources/repositories on the Internet.

The assumption is that the targeting OpenStack infrastructure contains multiple nodes: one is a controller node and one or more are compute nodes.

Network Configuration Example

The following is an example of how to configure networks for the OpenStack infrastructure. The example uses four network interfaces. Note that the names of these network interfaces (ports) are used throughout this document:

- **enp3s0f1:** For the Internet network — used to pull all necessary packages/patches from repositories on the Internet, configured to obtain a Dynamic Host Configuration Protocol (DHCP) address.
- **enp3s0f0:** For the management network — used to connect all nodes for OpenStack management, configured to use network `10.11.0.0/16`.
- **ens786f0:** For the tenant network — used for OpenStack internal connections for VMs. Configuration of the tenant network interface becomes more complicated with the introduction of the ODL controller. Depending on whether ODL is used for network control, the configuration of this interface varies:
    - If ODL is used, configure this interface with an IP address. This address should be from a different network than the management network.
    - If ODL is not used, configure this interface with no IP address.
- **ens786f1:** For the optional external network — Used for VM Internet/external connectivity, configured with no IP address. This interface is only used in the controller node, if the external network is configured. For the compute node, this interface is not required.

Notes:
1. Among these interfaces, the interface for the tenant network (in this example, **ens786f0**) is an X710 10 Gb port or XL710 40 Gb port (Fortville); it is used for DPDK and OVS with DPDK-netdev.
2. Also note that a static IP address should be used for the interface of the management network.

In Fedora, the network configuration files are located at: */etc/sysconfig/network-scripts/.*

To configure a network interface on the host system, edit the following network configuration files:

```
ifcfg-enp3s0f1
DEVICE=enp3s0f1
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=dhcp

ifcfg-enp3s0f0
DEVICE=enp3s0f0
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=static
IPADDR=10.11.12.11
NETMASK=255.255.0.0

ifcfg-ens786f0
DEVICE=ens786f0
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=none

ifcfg-ens786f1
DEVICE=ens786f1
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=none
```

Notes:
1. The IP address and netmask in the configuration files above are examples only.
2. The example of tenant network interface **ens786f0** assumes that ODL is not used; therefore, it is configured with no address. For the ODL configuration, refer to section 6.2, Using ODL.
3. Do not configure the IP address for the tenant network interface for the DPDK compute node, which will fail binding of the driver for DPDK during the OpenStack Neutron installation.

5.2.2.2 OpenStack Installation Procedures

General

DevStack provides and maintains tools for the installation of OpenStack from upstream sources. Its main purpose is for OpenStack development and testing of the components involved. Due to its evolving nature, DevStack does not provide production-level stability. The Intel ONP Server chooses DevStack for quick deployment of OpenStack in order to use its latest features, including, at times, its experimental ones. Given this, volatility may be introduced.

The following procedure uses actual examples of an OpenStack (DevStack) installation performed in an Intel test lab. It consists of one controller node (controller) and one compute node (compute).

Controller Node Installation Procedures

The following example uses a host for the controller node installation with the following:

- Hostname: sdnlab-k01
- Internet network IP address: Obtained from the DHCP server
- OpenStack Management port and IP address: `enp3s0f0, 10.11.12.11`
- Tenant (Data plane) network port: `ens786f0`
- User/password: 'stack/stack'

Root User Actions

Log in as **root** user and, if not already added, add the stack user to the sudoers list:

```
# echo "stack ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

Stack User Actions

1. Log in as **stack** user.
2. Configure the appropriate proxies (yum, http, https, and git) for the package installation and make sure these proxies are functional. Note that the controller node, localhost, and its IP address should be included in the `no_proxy setup` (e.g., `export no_proxy=localhost, 10.11.12.11`). For detailed instructions on how to set up your proxy, refer to Appendix B, Configuring the Proxy.
3. Download the DevStack source:

   ```
   $ git clone https://github.com/openstack-dev/devstack
   ```

4. Use OpenStack with the stable/kilo branch (or specific commit ID):

   ```
   $ cd /home/stack/devstack/
   $ git checkout stable/kilo
   ```

   or use a specific commit ID, for example:

   ```
   $ git checkout –b stable/kilo 4935abbcbb658b5a0f059fc8f2f2480b26e40554
   ```

5. Create the local.conf file in /home/stack/devstack/.

6. Pay attention to the following in the *local.conf* file:
   a. Explicitly disable the Nova compute service and Nova network service on the controller. By default these services are enabled:

```
disable_service n-cpu
disable_service n-net
```

   b. Explicitly enable VNC and related services. These services were previously enabled by default, but are now disabled by default:

```
enable_service n-novnc
enable_service n-xvnc
enable_service n-crt
enable_service n-cauth
```

   c. When the DPDK compute nodes are present in the OpenStack setup, an additional ML2 plugin mechnism driver, ovsdpdk, is required, even though no DPDK is installed on the controller. The OpenStack plugin, networking-ovs-dkdk, also should be installed as shown below:

```
$(sudo -E -H pip install -e git+https://github.com/stackforge/networking-ovs-dpdk.git@stable/kilo#egg=networking-ovs-dpdk )
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,ovsdpdk
```

   d. Two tenant network types, VLAN and VXLAN, are supported that require different settings:
      i. For VLAN tenant network, configure the following:

```
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
Q_ML2_TENANT_NETWORK_TYPE=vlanML2_VLAN_RANGES=<network name><VLAN range>
```

      Note: A network name is a name given to a network. The VLAN range should match the configuration in the switch the tenant network interface is connected to. For example, the following uses a network named "physnet1" with a VLAN range of 1000 to 1010:

```
ML2_VLAN_RANGES=physnet1:1000:1010
```

      In this case, the matching VLANs 1000 to 1010 should also be configured in the switch.

      ii. For the VXLAN tenant network, configure the following:

```
ENABLE_TENANT_TUNNELS=True
TUNNEL_ENDPOINT_IP=<IP addrees for VXLAN tunnel point>
```

      Note: If ODL is used, do not configure `TUNNEL_ENDPOINT_IP`, but remove it, if it exists because ODL will directly use the IP address of the network interface for the data plane to forward tenant network traffic.

   e. If the ODL controller is to install:
      i. The OpenStack plugin, networking-odl, and the following ODL parameters should be added:

```
enable_plugin networking-odl
http://git.openstack.org/openstack/networking-odl

Q_HOST=$HOST_IP
Q_PLUGIN=ml2
```

```
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight
ODL_MGR_IP=<IP address of management network interface>
ODL_PROVIDER_MAPPINGS=<name of the network>:<tenant network interface>
ODL_LOCAL_IP=<IP address of tenant network interface>
ODL_MODE=allinone
ODL_RELEASE=lithium-snapshot-0.3.1
```

ii. If they exist, these parameters should be removed:

```
Q_AGENT
TUNNEL_ENDPOINT_IP
OVS_PHYSICAL_BRIDGE
```

A sample *local.conf* file for the controller node with VXLAN tenant network is shown below. An example for the ODL controller is given in section 6.2, Using ODL.

```
# Controller node
#
[[local|localrc]]
FORCE=yes

HOST_NAME=$(hostname)
HOST_IP=10.11.12.16
HOST_IP_IFACE=enp3s0f0

PUBLIC_INTERFACE=ens786f1

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_service n-net
disable_service n-cpu

enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron
enable_service n-novnc
enable_service n-xvnc
enable_service n-crt
enable_service n-cauth
```

```
$(sudo -E -H pip install -e
git+https://github.com/stackforge/networking-ovs-
dpdk.git@stable/kilo#egg=networking-ovs-dpdk )

SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/xstack.sh.log
SYSLOG=True
LOGDAYS=1
MULTI_HOST=True

Q_AGENT=openvswitch
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,ovsdpdk
Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan,vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vxlan

ENABLE_TENANT_TUNNELS=True
TUNNEL_ENDPOINT_IP=192.168.12.16

PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-ens786f0

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
novncproxy_host=0.0.0.0
novncproxy_port=6080
```

7. Install DevStack:

```
$ cd /home/stack/devstack/
$ ./stack.sh
```

8. The following appears at the end of the screen output to indicate a successful installation:

```
"This is your host ip: 10.11.12.11"
"Horizon is now available at http://10.11.12.11.
```

9. After a successful installation, post-stacking work will need to be done by performing the following steps:

   a. Add physical port(s) to the bridge(s) created by the DevStack installation. The following example can be used to configure the two bridges: br-ens786f0 (for the tenant network) and br-ex (for the public network):

```
$ sudo ovs-vsctl add-port br-ens786f0 ens786f0
$ sudo ovs-vsctl add-port br-ex ens786f1
```

   b. If the VXLAN tenant network is used, set the tunnel endpoint IP address (i.e., the same address set in local.conf for `TUNNEL_ENDPOINT_IP`) for the bridge with the physical port and set tag for bridge br-int. The examples below configure address `192.168.12.16` for bridge br-ens786f0 and the set tag=1 for bridge br-int:

```
$ sudo ifconfig br-ens786f0 192.168.12.16 netmask 255.255.255.0
$ sudo ovs-vsctl set port br-int tag=1
```

   Note: If ODL is used, do not perform steps 9a and b. Refer instead to section 6.2.3, Additional Configurations and Operations, for post-stacking activity.

5.2.3 Compute Node Setup

This section describes how to complete the compute node setup. It is assumed that the user has successfully completed the BIOS settings and the operating system installation and configuration sections.

5.2.3.1 Host Configuration

Using DevStack to deploy vSwitch and OpenStack Components

General

Deploying OpenStack and OpenvSwitch using DevStack on a compute node follows the same procedures as on the controller node. Refer to section 5.2.2.2, OpenStack Installation Procedures, for controller node deployment to help with compute node setup. Differences include:

- Required services are Nova compute and Neutron agent.
- OVS with DPDK-netdev may be used in place of OVS for the Neutron agent.

Configure Qemu.conf:

1. Edit `/etc/libvirt/qemu.conf`, add or modify with the following lines:

```
cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuset",”cpuacct” ]
cgroup_device_acl = [ "/dev/null", "/dev/full", "/dev/zero", "/dev/random",
"/dev/urandom", "/dev/ptmx", "/dev/kvm", "/dev/kqemu", "/dev/rtc", "/dev/hpet",
"/dev/net/tun" ]
```

2. If DPDK is used, add the following to the end of `cgroup_device_acl`:

```
"/mnt/huge","/dev/vhost-net"
```

   Also add this line to the file:

```
hugetlbs_mount = “/mnt/huge”
```

3. Restart libvirt service and make sure libvirtd is active:

```
systemctl restart libvirtd.service
```

Compute Node Installation Example

The example below uses a host for the compute node installation with the following:

- Hostname: sdnlab-k02
- Intenet network IP address: Obtained from the DHCP server
- OpenStack Management IP address: `ens2f0, 10.11.12.12`
- Data plane network port: `ens786f0`
- User/password: ‘stack/stack’

Note the following:

- `No_proxy` setup: Localhost and its IP address should be included in the `no_proxy` setup. The hostname and IP address of the controller node should also be included. For example:

```
export no_proxy=localhost,10.11.12.12,sdnlab-k01,10.11.12.11
```

  Refer to Appendix B, Configuring the Proxy, for more details.

- Differences in the *local.conf* file:
    - The service host is the controller, as well as other OpenStack services (e.g., database, messaging, authentication, and image). The service host should be spelled out in the local.conf of the compute node. Using the controller node example in the previous section, the service host and its IP address should be:

```
SERVICE_HOST_NAME=sdnlab-k01
SERVICE_HOST=10.11.12.11
```

    - The only OpenStack services required in compute nodes are Nova compute and Neutron agent, so the local.conf might look like:

```
disable_all_services
enable_service n-cpu
enable_service q-agt
```

    - OVS is used for the Neutron agent and Neutron ML2 plugin driver:

```
Q_AGENT=openvswitch
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
```

    - For the OVS with DPDK-netdev setting, the differences are:
        - In place of `Q_AGENT=openvswitch`, use `OVS_AGENT_TYPE=ovsdpdk`
        - In place of `Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch`, use `Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,ovsdpdk`
    - OVS-DPDK repository for OpenStack, use (or not use) vHost-user, DPDK version, OVS git tag; hugepages settings should be added:

```
enable_plugin networking-ovs-dpdk https://github.com/stackforge/networking-
ovs-dpdk 2015.1.1
OVS_DPDK_RTE_LIBRTE_VHOST=y
OVS_DATAPATH_TYPE=netdev
OVS_NUM_HUGEPAGES=8192
OVS_DPDK_MEM_SEGMENTS=8192
OVS_HUGEPAGE_MOUNT_PAGESIZE=2M
OVS_DPDK_GIT_TAG=v2.0.0
OVS_GIT_TAG=1e77bbe565bbf5ae7f4c47f481a4097d666d3d68
```

  If ODL is used in place of `ODL_MODE=allinone` in the controller node, use `ODL_MODE=compute` for the compute node. In addition, `Q_HOST` should be the IP address of the controller. Note that in the controller this value is `Q_HOST=$HOST_IP`, but in the compute node, it should be `Q_HOST=$SERVICE_HOST`.

  Note: Currently, if ODL is used, OVS with DPDK-netdev is not supported. Therefore, use regular OVS when ODL is in place.

- A sample local.conf file for compute node with OVS with DPDK-netdev agent is as follows:

```
# Compute node
# OVS_TYPE=ovs-dpdk
[[local|localrc]]

FORCE=yes
MULTI_HOST=True

HOST_NAME=$(hostname)
HOST_IP=10.11.12.17
```

```
HOST_IP_IFACE=enp3s0f0
SERVICE_HOST_NAME=sdnlab-wp16
SERVICE_HOST=10.11.12.16

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST

GLANCE_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_all_services
enable_service n-cpu
enable_service q-agt
enable_plugin networking-ovs-dpdk https://github.com/stackforge/networking-ovs-dpdk
2015.1.1
OVS_DPDK_RTE_LIBRTE_VHOST=y

DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/stack.sh.log
SYSLOG=True
LOGDAYS=1

OVS_AGENT_TYPE=ovsdpdk
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,ovsdpdk
Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan,vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vxlan
OVS_GIT_TAG=1e77bbe565bbf5ae7f4c47f481a4097d666d3d68
OVS_DPDK_GIT_TAG=v2.0.0
OVS_DATAPATH_TYPE=netdev
OVS_NUM_HUGEPAGES=8192
OVS_DPDK_MEM_SEGMENTS=8192
OVS_HUGEPAGE_MOUNT_PAGESIZE=2M

ENABLE_TENANT_TUNNELS=True
TUNNEL_ENDPOINT_IP=192.168.12.17
```

```
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-ens786f0

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```

- A sample local.conf file for compute node with the regular OVS agent follows:

```
# Compute node
# OVS_TYPE=ovs
[[local|localrc]]

FORCE=yes
MULTI_HOST=True

HOST_NAME=$(hostname)
HOST_IP=10.11.12.15
HOST_IP_IFACE=enp3s0f0
SERVICE_HOST_NAME=sdnlab-wp16
SERVICE_HOST=10.11.12.16

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST

GLANCE_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_all_services
enable_service n-cpu
enable_service q-agt

DEST=/opt/stack
```

```
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/stack.sh.log
SYSLOG=True
LOGDAYS=1

Q_AGENT=openvswitch
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan,vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vxlan

ENABLE_TENANT_TUNNELS=True
TUNNEL_ENDPOINT_IP=192.168.12.15

PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-ens786f0

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```

- Perform the same post-stacking procedure as with the controller. Note that if ODL is used, do not perform the next two steps.
- For the regular OVS compute node, add the physical port(s) to the bridge(s) created by the DevStack installation. The following example can be used to configure the bridges: br-ens786f0 for the virtual network:

```
$ sudo ovs-vsctl add-port br-ens786f0 ens786f0
```

Note: For the compute node with DPDK, the physical port is bound to port dpdk0; and dpdk0 will be automatically added to the bridge.

- If the VXLAN tenant network is used, set the tunnel endpoint IP address for the bridge with the physical port. The examples below configure address `192.168.12.16` for bridge br-ens786f0 for bridge br-int:

```
$ sudo ifconfig br-ens786f0 192.168.12.16 netmask 255.255.255.0
```

6.0 VM Deployment Using OpenStack

This section describes how to bring up the VMs in the OpenStack environment, deploy various advanced features of OpenStack, and integrate them with the network controller, ODL.

Note: It is assumed that commands that require root privileges are listed beginning with '`#`' and those that require only user privileges are listed beginning with the '`$`' sign.

6.1 Preparation with OpenStack

6.1.1 Deploying VMs

6.1.1.1 Default Settings

OpenStack deployed with DevStack comes with the following default settings:

- Tenant (Project): admin and demo
- Network:
  - Private network (virtual network): `10.0.0.0/24`
  - Public network (external network): `172.24.4.0/24`
- Image: cirros-0.3.2-x86_64
- Flavor: nano, micro, tiny, small, medium, large, and xlarge

To deploy new instances (VMs) with different setups (e.g., different VM image, flavor, or network), users must create custom configurations. OpenStack provides a command-line interface and graphic interface (Horizon) for this purpose. In this section, you will be shown how to use OpenStack commands to create VMs with custom settings. For the same functionalities using a graphic interface, refer to Appendix C, Configuring Horizon UI to Deploy VMs.

To access the OpenStack dashboard, use a web browser (Firefox, Internet Explorer, etc.) and controller's IP address (management network), for example, http://10.11.12.11/.

Login information is defined in the *local.conf* file. In the examples that follow, "password" is the password for both admin and demo users.

6.1.1.2 Manual Deployment with Custom Settings

The following examples describe how to create a VM image with custom options like flavor, and aggregate/availability zone using OpenStack commands. The examples assume the IP address of the controller is `10.11.12.11`.

1. Log in as **stack** user.

   Notes:
   a. Some OpenStack commands (e.g., keystone and aggregate-related) can only be used by the admin user, while others (e.g., Glance, Nova, and those that are Neutron-related) can be used by other users, but with limited visibility.
   b. DevStack provides a built-in tool, `openrc`, located at */home/stack/DevStack/* to source an OpenStack user credential to the shell environment.

2. Source the admin credential:

   ```
   $ source /home/stack/DevStack/openrc admin demo
   ```

   Note: OpenStack commands will thereafter use the admin credential.

3. Create an OpenStack Glance image. (Glance is the OpenStack component that manages VM images.) A VM image file should be ready in a location accessible by OpenStack. The following command creates an OpenStack image from an existing image file. The image is used as a template for creating new VMs:

   ```
   $ glance image-create --name <image-name-to-create> --is-public=true --
   container-format=bare --disk-format=<format> --file=<image-file-path-name>
   ```

   The following example shows the image file, fedora20-x86_64-basic.qcow2, and is located in a NFS share and mounted at */mnt/nfs/openstack/images/* to the controller host. The command creates a Glance image named "fedora-basic" with a qcow2 format for the public that any tenant can use:

   ```
   $ glance image-create --name fedora-basic --is-public=true –container-
   format=bare --disk-format=qcow2 --file=/mnt/nfs/openstack/images/fedora20-
   x86_64-basic.qcow2
   ```

4. Create the host aggregate and availability zone:

   First find out the available hypervisors, and then use this information to create an aggregate/ availability zone:

   ```
   $ nova hypervisor-list
   $ nova aggregate-create <aggregate-name> <zone-name>
   $ nova aggregate-add-host <aggregate-name> <hypervisor-name>
   ```

   The following example creates an aggregate named "aggr-g06" with one availability zone named 'zone-g06' and the aggregate contains one hypervisor named "sdnlab-g06":

   ```
   $ nova aggregate-create aggr-g06 zone-g06
   $ nova aggregate-add-host aggr-g06 sdnlab-g06
   ```

5. Create flavor. (Flavor is a virtual hardware configuration for the VMs; it defines the number of virtual CPUs, size of the virtual memory, disk space, etc.)

   The following command creates a flavor named "onps-flavor" with an ID of 1001, 1024 Mb virtual memory, 4 Gb virtual disk space, and 1 virtual CPU:

   ```
   $ nova flavor-create onps-flavor 1001 1024 4 1
   ```

6. Source the demo user credential. Note that OpenStack commands will continue to use this demo credential:

```
$ source /home/stack/DevStack/openrc demo demo
```

7. Create a network for the tenant demo as follows:
   a. Create a tenant network:

```
$ neutron net-create <network-name>
```

   The following creates a network with a name of "net-demo" for the tenant demo (because a demo credential is used):

```
$neutron net-create net-demo
```

   b. Create a subnet:

```
$ neutron subnet-create --name <subnet_name> <network-name> <net-ip-range>
```

   The following creates a subnet with a name of "sub-demo" and CIDR address `192.168.2.0/24` for the network net-demo:

```
$ neutron subnet-create --name sub-demo net-demo 192.168.2.0/24
```

8. Create an instance (VM) for the tenant demo as follows:
   a. Get the name and/or ID of the image, flavor, and availability zone to be used for creating the instance:

```
glance image-list
nova flavor-list
nova availability-zone-list
neutron net-list
```

   b. Launch the instance (VM) using information from the previous step:

```
$ nova boot --image <image-id or image-name> --flavor <flavor-id or flavor-
name> --availability-zone <zone-name> --nic net-id=<network-id or network-
name> <instance-name>
```

   The new VM should be up and running in a few minutes.

Special Consideration for OVS-DPDK Compute Node with a VHost-User

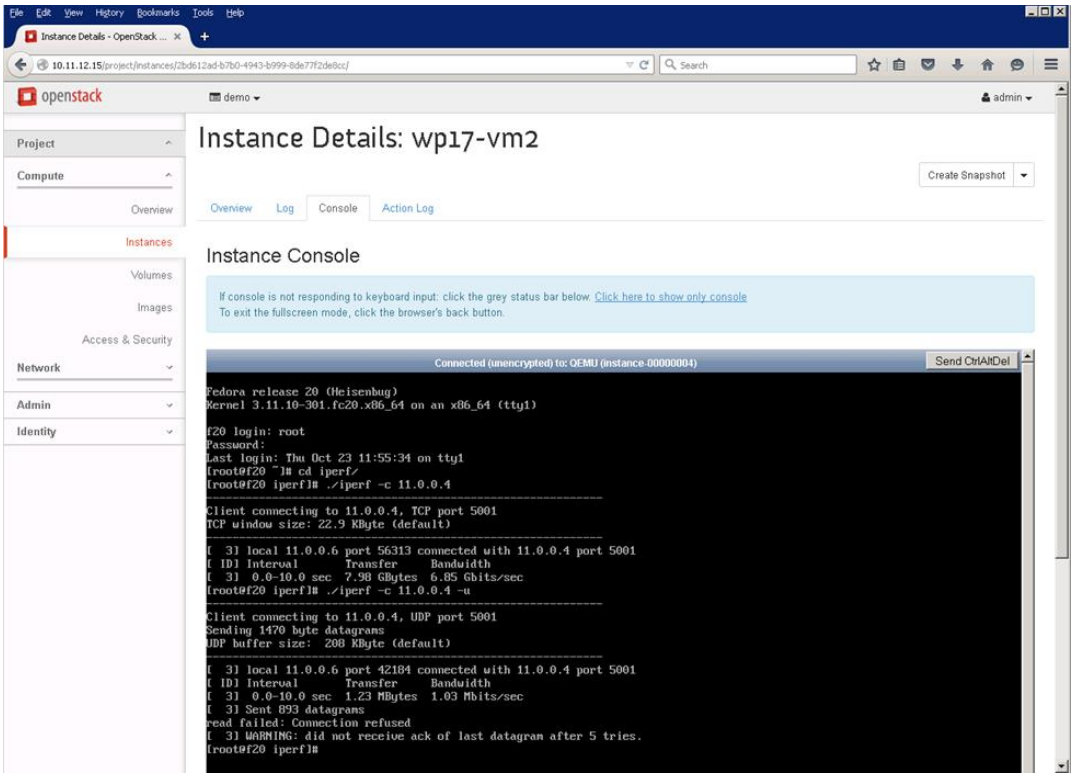
With the OVS-DPDK compute node with a vhost-user, a large memory page size should be configured for the OpenStack flavor for creating instances. This can be done in two steps: first create a flavor, and then modify it to allow a large memory page size.

The following commands create a flavor named "largepage-flavor" with an ID of 1002, 1024 Mb virtual memory, 4 Gb virtual disk space, 1 virtual CPU, and large memory page size:

```
$ nova flavor-create largepage-flavor 1002 1024 4 1
$ nova flavor-key 1002 set "hw:mem_page_size=large"
```

Use this flavor to create instances hosted by OVS-DPDK compute node with a vhost-user.

9. Log in to the OpenStack dashboard using the demo user credential; click **Instances** under "Project" in the left pane; the new VM should show in the right pane. Click the instance name to open the **Instance Details** view, and then click **Console** in the top menu to access the VM as show below.



Special Consideration for Tenant Network with VXLAN Tunnelling

With the introduction of VXLAN tunnelling to tenant network, the user must consider the maximum trasmission unit (MTU) associated with the network interface in order to have the tenant network function as expected. One method to do this is to decrease the MTU of all virtual network devices (i.e., network interfaces in all VMs) by 50 bytes to 1450 byte. This is because the VXLAN adds an extra 50-byte header to the MTU of 1500 bytes for normal Ethernet packets. The example below changes the MTU eth0 to 1450 bytes:

```
# ip link set eth0 mtu 1450
```

6.2 Using ODL

6.2.1 Preparing the ODL Controller

Networking-odl is an OpenStack project that develops a plugin library to integrate the ODL controller with Neutron. This plugin allows easy deployment of ODL in the OpenStack environment. In DevStack, the ODL installation is through networking-odl plugin specified in the *local.conf* file.

Note: Currently OVS with DPDK-netdev is not supported by ODL; therefore, use regular OVS for both controller and compute nodes when ODL is in place.

For Fedora 21, the native Java version from the yum repository is Java v1.8. The current ODL version Lithium SR1, however, only supports Java v1.7. It is necessary, therefore, to manually install Java 1.7 on Fedora 21 system and make it the default Java version:

1. Download the RPM file from the Oracle website:
2. # curl -v -j -k -L -H "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/7u75-b13/jdk-7u75-linux-x64.rpm > jdk-7u75-linux-x64.rpmInstall Java v1.7:

   ```
   # rpm -ivh jdk-7u75-linux-x64.rpm
   ```

3. Set Java v1.7 as the default version and verify. Note that Java v1.7 was installed at */usr/java/*:

   ```
   # alternatives --install /usr/bin/java java /usr/java/default/jre/bin/java 2
   # echo 2 | alternatives --config java
   # java -version
   ```

4. Configure the environment variable `JAVA_HOME`:

   a. Find the Java path:

      ```
      # ls -l /etc/alternatives/java
      ```

      Note: The output of the above command points to */usr/java/default/jre/bin/java*.

   b. Set `JAVA_HOME` for the "root" user:

      ```
      # echo "export JAVA_HOME=/usr/java/default/jre" >> /root/.bashrc
      # source /root/.bashrc
      ```

      Note: `jre/java` should not be included in the path for `JAVA_HOME`.

   c. Repeat above for the "stack" user.

      ```
      # echo "export JAVA_HOME=/usr/java/default/jre" >> /home/stack/.bashrc
      # source /home/stack/.bashrc
      ```

5. If your infrastructure requires a proxy server to access the Internet, follow the maven-specific instructions in Appendix B, Configuring the Proxy.

6.2.2 Preparing for DevStack

Controller Node Configuration

1. Update the existing *local.conf* file in order for ODL to be functional with DevStack. The example below assumes the following:
   - The controller management IP address is `10.11.13.35`.
   - Port ens786f0 is used for the data plane network.
2. On the controller (refer to section 5.2.2.2, OpenStack Installation Procedures, for the original local.conf settings on the controller), modify local.conf:

   Remove or comment out these lines:

   ```
   enable_service q-agt
   Q_AGENT=openvswitch
   Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
   OVS_PHYSICAL_BRIDGE=br-ens786f0
   ```

   Add these lines in the middle of file, anywhere before [[post-config|$NOVA_CONF]]:

   ```
   enable_plugin networking-odl http://git.openstack.org/stackforge/networking-odl
   stable/kilo
   Q_HOST=$HOST_IP
   Q_PLUGIN=ml2
   Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight
   ODL_MGR_IP=10.11.13.35
   ODL_PROVIDER_MAPPINGS=physnet1:ens786f0
   ODL_LOCAL_IP=192.168.13.35
   ODL_MODE=allinone
   ODL_RELEASE=lithium-snapshot-0.3.1
   ```

   Note: Pay attention to the parameter `ODL_LOCAL_IP`, which is is the IP address of the data plane port. In the example above, this is port `ens786f0`.
3. On the ODL controller node, an issue was found in */opt/stack/networking-odl/DevStack/plugin.py* that tries to do a yum install of java-1.7 that is unavailable in the Fedora 21 repository. (For this reason, Java 1.7 is manually installed as the first step in this solutions guide.) With this bug, stacking fails and the error points to the installation of Java. To avoid the stacking failure, run the following commands before stacking:

   ```
   # mkdir /opt/stack
   # chown -R stack:stack /opt/stack
   # git clone http://git.openstack.org/openstack/networking-odl
   /opt/stack/networking-odl
   # chown -R stack:stack /opt/stack/networking-odl
   # cd /opt/stack/networking-odl
   # git checkout stable/kilo
   # sed -i 's/yum_install maven java-1.7.0-openjdk/yum_install maven/g'
   /opt/stack/networking-odl/devstack/plugin.sh
   ```

4. Run `stack.sh` to bring up OpenStack. If you missed step 3 and experienced a stacking failure, run the following to recover:

```
$ cd /home/stack/DevStack
$ ./unstack.sh
$ sed -i 's/yum_install maven java-1.7.0-openjdk/yum_install maven/g'
/opt/stack/networking-odl/devstack/plugin.sh
S ./stack.sh
```

Compute Node Configuration

1. Modify local.conf similar to that on the controller node, except that `Q_HOST` points to the controller (`SERVICE_HOST`) and `ODL_MODE` is compute, as shown below:

```
Q_HOST=$SERVICE_HOST
ODL_MODE=compute
```

2. Edit `/etc/libvirt/qemu.conf`, add or modify with the following lines :

```
cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuset","”cpuacct” ]
cgroup_device_acl = [ "/dev/null", "/dev/full", "/dev/zero", "/dev/random",
"/dev/urandom", "/dev/ptmx", "/dev/kvm", "/dev/kqemu", "/dev/rtc", "/dev/hpet",
"/dev/net/tun" ]
```

   Note: If you have performed step 1 in section 5.2.3.1, Host Configuration, skip this step. Also make sure that the commit ID is 4f9f667774eb773e43cb71e6bc96c10951a43544 for the networking-odl and the tag is 2015.1.1 for Nova and Neutron.

3. Run `stack.sh`.

Sample local.conf files for both the controller and compute nodes are listed in Appendix A, Sample local.conf Files for OpenStack with ODL Controller.

6.2.3 Additional Configurations and Operations

With the ODL configuration in OpenStack, the data plane bridge (e.g., br-enp786f0) is no longer used. Instead, ODL directly controls the bridge br-int for data flow. In a perfect scenario, after successful stacking no additional configurations are required. Several issues, however, are present in the current version of ODL:

1. On the controller, it does not add the manager IP address to the bridges br-int and br-ex. The workaround is to add this address to the bridges. The examples below assume the management IP address of the controller is `10.11.12.35`:

```
$ sudo ovs-vsctl set controller br-int tcp:10.11.12.35:6653
$ sudo ovs-vsctl set controller br-ex tcp:10.11.12.35:6653
```

2. On the controller, it does not create a VXLAN port in bridge br-int for remote peers. This issue is particularly problematic, because this port should be created dynamically as a remote peer added into the OpenStack configuration.

   The workaround is to add the port manually before the peer (compute node) is created. A precondition is that the IP address of the data port (e.g., ens786f0) of both the controller and compute nodes are known. The example below creates a VXLAN port in bridge br-int for the compute node with the IP address of `192.168.12.38` for the data port. Note that the IP address of the controller data port is `192.168.12.35`.

```
$ sudo ovs-vsctl add-port br-int vxlan-192.168.12.38 -- set interface vxlan-
192.168.12.38 type=vxlan options:remote_ip=192.168.12.38
options:local_ip=192.168.12.35 options:key=flow
```

3. On the controller, the default tenant network, private network, does not pass the traffic. Users are adviced to create their own tenant network in order to create a fully functional VM. Refer to section 6.1.1.2, Manual Deployment with Custom Settings, for how to create a tenant network.

You can use the same methods to create a VM as those described in section 6.1, Preparation with OpenStack.

6.2.4 Monitor Network Flow with ODL

DLUX (OpenDaylight User eXperience) is a web-based graphic interface for ODL. After successfully installing as described above, however, users still cannot access the DLUX. This is because some features are not installed by default.

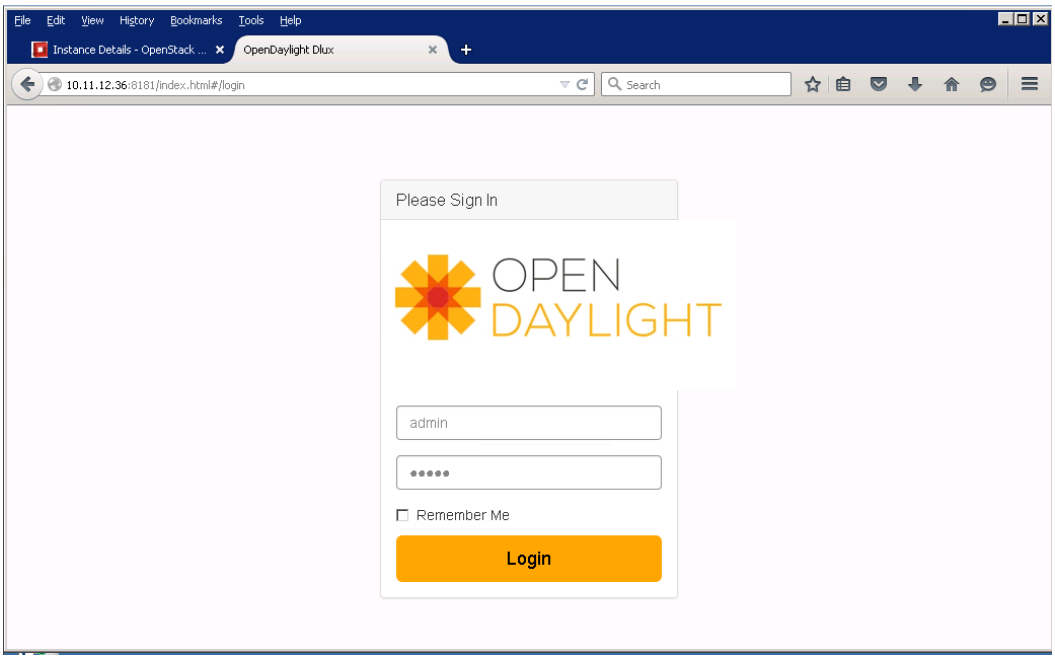
1. To install these features, do the following:

```
$ cd /opt/stack/opendaylight/distribution-karaf-0.3.1-SNAPSHOT/
$ ./bin/client -u karaf feature:install odl-base-all odl-nsf-all odl-adsal-
northbound odl-mdsal-apidocs odl-ovsdb-openstack odl-ovsdb-northbound odl-dlux-
core odl-dlux-all
```

   Note: The features above can be selectively chosen and installed one at a time depending on user requirements.

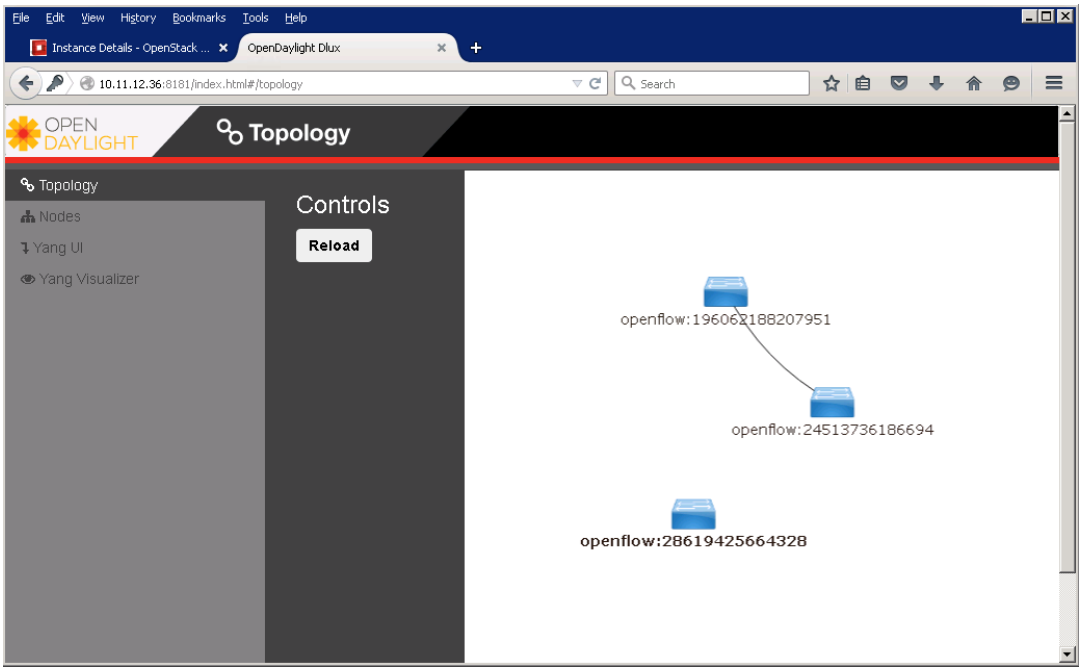
2. Access DLUX through a Web browser. The following is an example of an ODL login page from the controller running at IP address `10.11.12.36`:

   http://10.11.12.36:8181/index.html

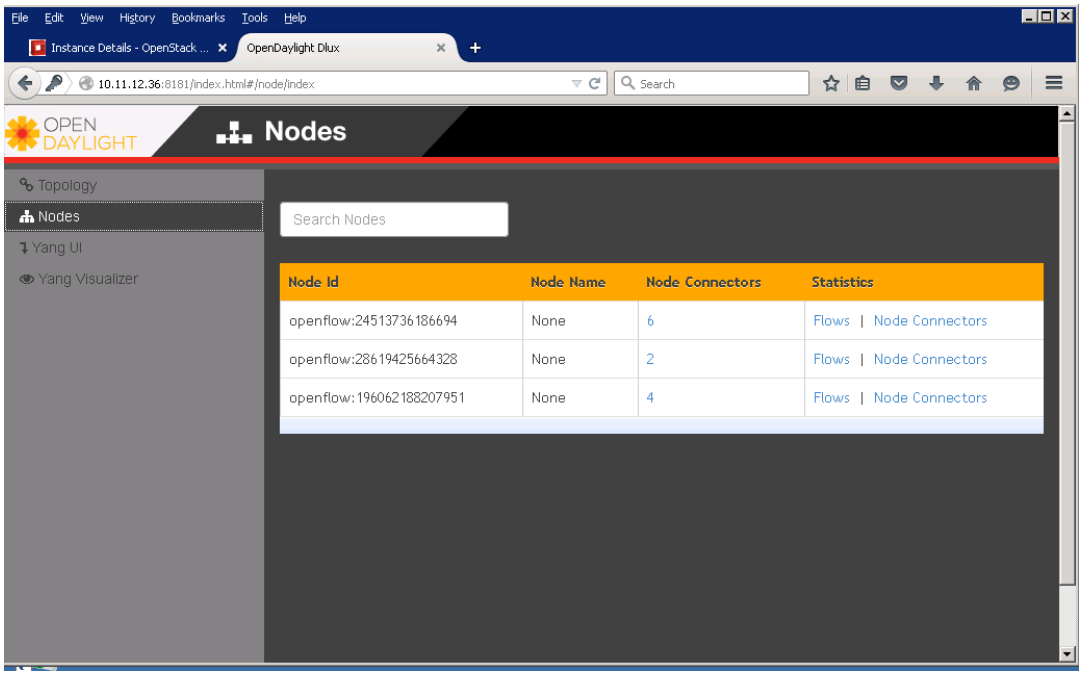


3. Log in with the default login credential `admin/admin`.

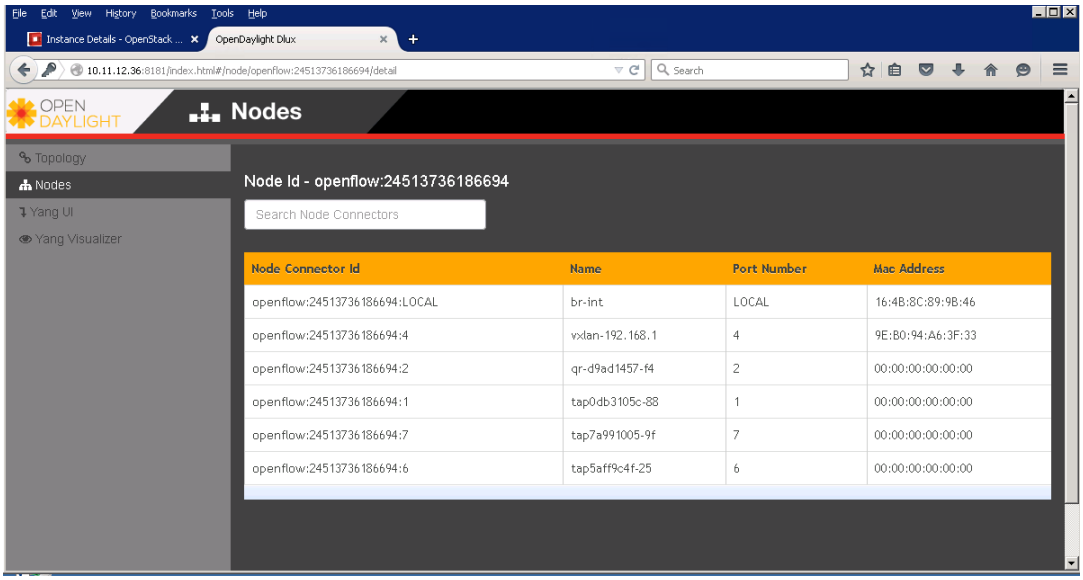
4. After login, the user can browse to "Topology" for a visual view of the openflow topology. The example below shows three openflows, one for bridge br-int on controller (openflow:24513736186694), one for br-int on compute node (openflow:196062188207951), and one for br-ex on controller (openflow:28619425664328).



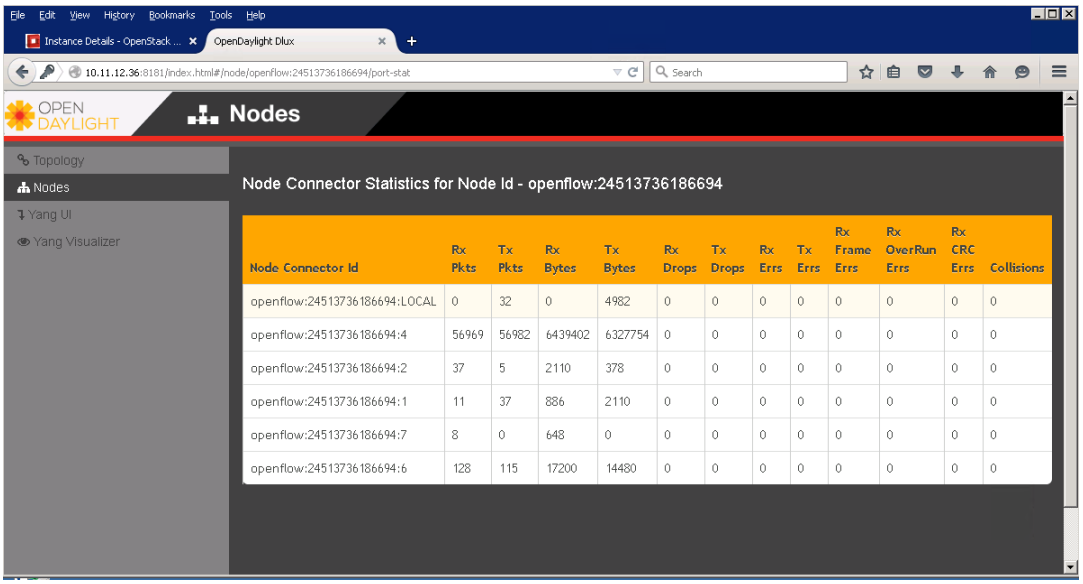
5. Browse to "Nodes" with the following view.



For more details on each flow, click **Node Connectors**. The screenshot below shows the connector with a VXLAN port, a router port, and three VM ports (port name starting with tap) and br-int port.



6. Click **Flows** to show the Rx and Tx statistics of each port.



7.0 Use Cases with Virtual Network Functions

This section describes the Virtual Network Functions (VNFs) that have been used in the ONP for servers. These functions assume VMs that have been prepared in a way similar to compute nodes. Refer to section 6.1, Preparation with OpenStack, for the test setup.

7.1 Generic VNF Configurations

Any VNF like a virtual firewall or virtual router can be implemented on either the same compute node like other VMs or a remote compute node. The following two sections provide examples of how traffic can be routed for these two different scenarios.

7.1.1 Local VNF

Figure 7–1 describes a simple network flow between two VMs (a source and sink) with a VNF between them. All three are locally configured on the same compute node.

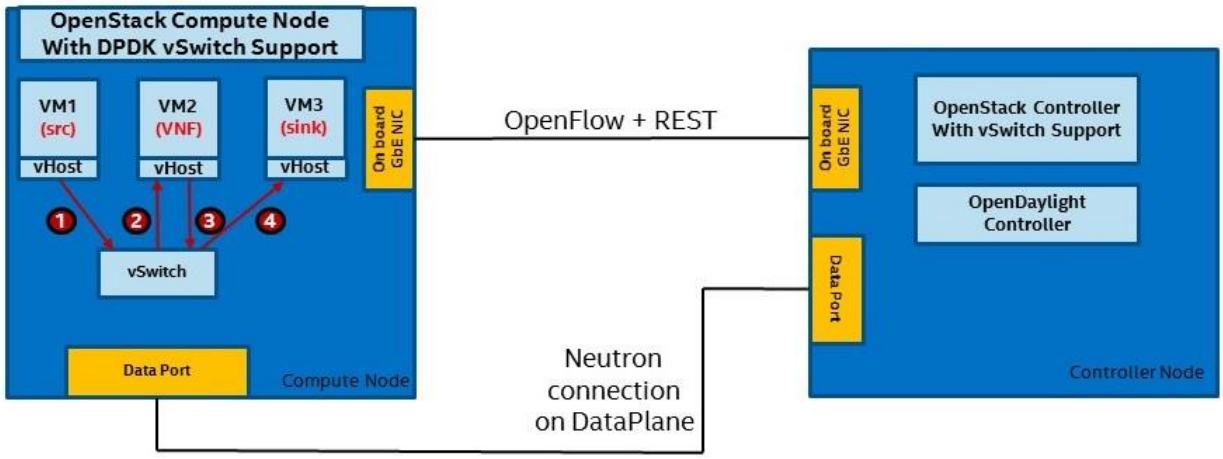


Figure 7–1. Local VNF Configuration

Configuration

1. OpenStack brings up the VMs and connects them to the vSwitch.
2. The VMs obtained IP addresses from OpenStack built-in DHCP servers. VM1 belongs to one subnet and VM3 to a different one. VM2 has ports on both subnets.
3. Flows get programmed to the vSwitch by either OpenStack Neutron or ODL controller (refer to section 6.2, Using ODL).

Data Path (Numbers Matching Red Circles)

1. VM1 sends a flow to VM3 through the vSwitch.
2. The vSwitch forwards the flow to the first vPort of VM2.
3. The VM2 VNF receives the flow, processes it as per its functionality and sends it out through its second vPort.
4. The vSwitch receives the flow and sends it out to VM3's vPort.

7.1.2 Remote VNF

Figure 7–2 shows a simple network flow between two VMs (source and sink) with the VNF configured on a separate compute node.

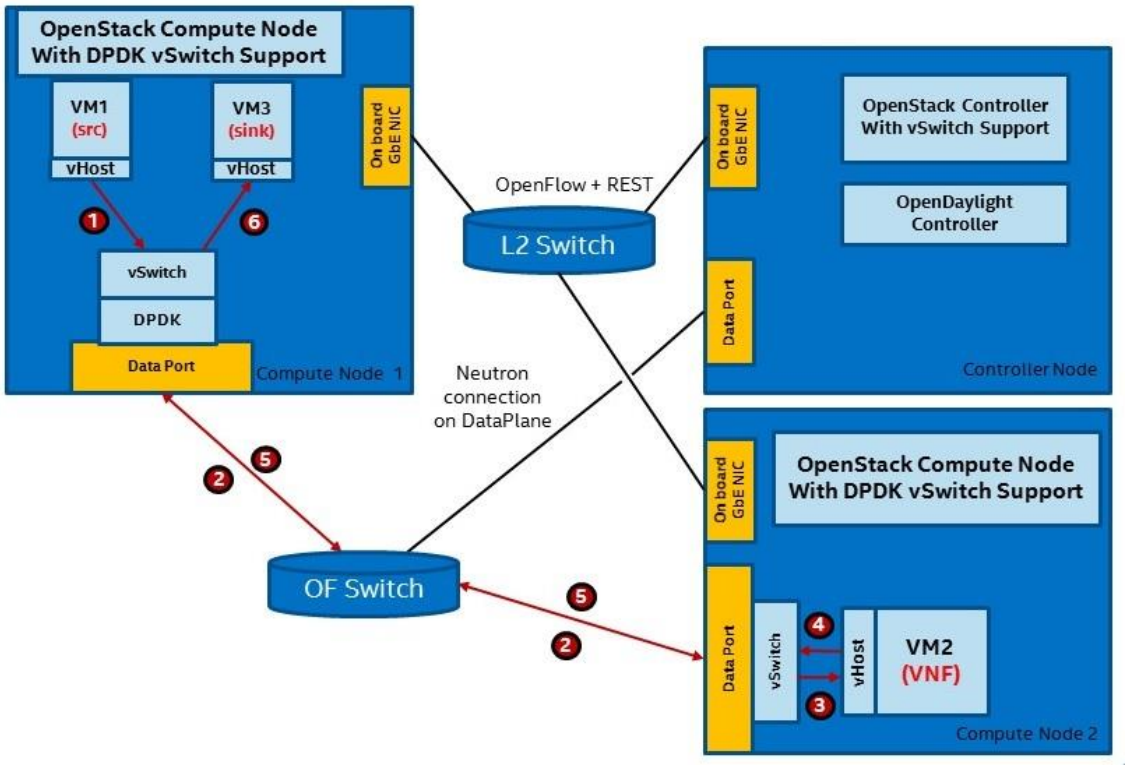


Figure 7–2. Remote VNF Configuration

Configuration

1. OpenStack brings up the VMs and connects them to the vSwitch.
2. The VMs obtained IP addresses from OpenStack built-in DHCP servers.

Data Path (Numbers Matching Red Circles)

1. VM1 sends a flow to VM3 through the vSwitch inside compute node 1.
2. The vSwitch forwards the flow out of the first port to the first port of compute node 2.
3. The vSwitch of compute node 2 forwards the flow to the first port of the vHost, where the traffic gets consumed by VM1.
4. The VNF receives the flow and sends it out through its second port of the vHost into the vSwitch of compute node 2.
5. The vSwitch forwards the flow out of the second XL710 port of compute node 2 into the second port of the XL710 in compute node 1.
6. The vSwitch of compute node 1 forwards the flow into the port of the vHost of VM3 where the flow gets terminated.

7.1.3 Network Configuration with Source and Sink VM

Sink and source are two Fedora VMs that are used only to generate or receive network traffic. The examples below assume two OpenStack tenant networks, `10.0.0.0/24` and `11.0.0.0/24`, were created for use with a VNF:

1. Install iperf:

```
# yum install -y iperf
```

2. Turn on IP forwarding:

```
# sysctl -w net.ipv4.ip_forward=1
```

3. In the source, add the route to the sink:

```
# ip route add 11.0.0.0/24 dev eth0
```

4. At the sink, add the route to the source:

```
# ip route add 10.0.0.0/24 dev eth0
```

7.2 Installation and Configuration of vIPS

7.2.1 Setup

Controller and compute nodes have been set up with the following ingredients:

- DPDK (only for compute node)
- OVS
- OpenStack Kilo Release

As a prerequisite, it is assumed that OpenStack is installed and properly configured on the controller and compute nodes as described in section 6.1, Preparation with OpenStack.

7.2.2 Local vIPS Test

1. From the OpenStack UI (<controller ip address>/project/instances), the "demo" user navigates as follows: **Compute > Project > Instances > Access VMs: vm01, vm03 & vm02.** It is assumed in this test that vm01 is on network 1, vm03 is on network 2, and vm02 is on both networks; vm01 is the source and vm03, the sink.
2. From each console, log in as **root**: Check that vm has loopback and the IP address is assigned by DHCP:

```
$ ip addr | grep inet
inet 127.0.0.1/8 scopt host lo
inet 10.0.0.3/24 scope global dynamic eth0
```

3. From vm02:
   a. Test ping the other vms:

```
$ ping <vm01-ip>
$ ping <vm03-ip>
```

   b. Turn on IP forwarding, or run `suricata.sh`:

```
# sysctl net.ipv4.ip_forward=1
# iptables -I FORWARD -i eth0 -o eth1 -j NFQUEUE
# iptables -I FORWARD -i eth1 -o eth0 -j NFQUEUE
# echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp
# echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
$ suricata -c /etc/suricata/suricata.yaml -q 0
```

4. From vm01:
   a. Test ping vm02:

```
$ ping <vm02-net1-ip>
```

   b. Turn on IP forwarding or run `add_router.sh`:

```
# sysctl -w net.ipv4.ip_forward=1
# ip route add 11.0.0.0/24 dev eth0
```

c. Test ping vm02 and vm03:

```
$ ping <vm02-net2-ip>
$ ping <vm03-ip>
```

5. From vm03:

a. Test ping vm02:

```
$ ping <vm02-net2-ip>
```

b. Turn on IP forwarding or run `add_router.sh`:

```
# sysctl -w net.ipv4.ip_forward=1
# ip route add 10.0.0.0/24 dev eth0
```

c. Test ping vm02 and vm03:

```
$ ping <vm02-net1-ip>
$ ping <vm01-ip>
```

6. Ensure the test pings from step 3 to step 5 are successful. Then proceed as follows:

a. From vm03:

```
$ cd <path to iperf>
$ ./iperf –s
```

b. From vm01:

```
$ cd <path to iperf>
$ ./iperf -c <vm03-ip> -l 1000 -t 60
$ ./iperf -c <vm03-ip> -l 64 -t 60
$ ./iperf -c <vm03-ip> -l 1450 -t 60
```

Configuration:

Note: Refer to Figure 7–1.

1. OpenStack brings up the VMs and connects them to the vSwitch.
2. IP addresses of the VMs get configured using the DHCP server. VM1 belongs to one subnet and VM3 to a different one. VM2 has ports on both subnets.

Data Path (Numbers Matching Red Circles):

1. VM1 sends a flow to VM3 through the vSwitch.
2. The vSwitch forwards the flow to the first vPort of VM2 (active IPS).
3. The IPS receives the flow, inspects it and (if not malicious) sends it out through its second vPort.
4. The vSwitch forwards it to VM3.

7.2.3 Remote vIPS Test

1. After completing local vIPS test, delete vm02 (from the previous setup) from the the control node:

```
source demo-cred
nova delete <vm02>
```

2. Return to step 8 of section 6.1.1.2, Manual Deployment with Custom Settings:
   a. Create vm02 in a different availability zone (and, therefore, different aggregate and compute node) from the one vm01 and vm03 are on.
   b. Proceed with all steps as in the local vIPS test (refer to section 7.2.2, Local vIPS Test).

Configuration

Note: Refer to Figure 7–2.

1. OpenStack brings up the VMs and connects them to the vSwitch.
2. The IP addresses of the VMs get configured using the DHCP server.

Data Path (Numbers Matching Red Circles)

1. VM1 sends a flow to VM3 through the vSwitch inside compute node 1.
2. The vSwitch forwards the flow out of the first 710 port (used for DPDK and OVS) to the first data port of compute node 2.
3. The vSwitch of compute node 2 forwards the flow to the first port of the vHost, where the traffic gets consumed by VM2.
4. The IPS receives the flow, inspects it, and (provided it is not malicious) sends it out through its second port of the vHost into the vSwitch of compute node 2.
5. The vSwitch forwards the flow out of the second data port of compute node 2 into the second data port in compute node 1.
6. The vSwitch of compute node 1 forwards the flow into the port of the vHost of VM3 where the flow gets terminated.

7.3 Installation and Configuration of vBNG

A virtual Border Network Gateway (vBNG) application can be configured and installed using the set of commands below. It is assumed that you have successfully set up a controller node and a compute node as described in section 6.1, Preparation with OpenStack.

1. Execute the following command in a Fedora VM with two Virtio interfaces:

```
# yum -y update
```

2. Disable SELINUX:

```
# setenforce 0
# vi /etc/selinux/config
```

   And change so SELINUX=disabled.

3. Disable the firewall:

```
# systemctl disable firewalld.service
# reboot
```

4. Edit the grub default configuration:

```
# vi /etc/default/grub
```

   and add hugepages to it:

```
… noirqbalance intel_idle.max_cstate=0 processor.max_cstate=0 ipv6.disable=1
default_hugepagesz=1G hugepages=4 2
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19"
```

5. Rebuild grub config and then reboot the system:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
# reboot
```

6. Set the number of hugepages:

```
# echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
```

7. Verify that hugepages are available in the VM:

```
# cat /proc/meminfo
...
HugePages_Total:1024
HugePages_Free:1024
Hugepagesize:2048 kB
...
```

8. Add the following to the end of ~/.bashrc file:

```
# -----------------------------------------
export RTE_SDK=/root/dpdk
export RTE_TARGET=x86_64-native-linuxapp-gcc
export OVS_DIR=/root/ovs
export RTE_UNBIND=$RTE_SDK/tools/dpdk_nic_bind.py
export DPDK_DIR=$RTE_SDK;
export DPDK_BUILD=$DPDK_DIR/$RTE_TARGET
# -----------------------------------------
```

9. Re-log in or source that file:

```
# . .bashrc
```

10. Install DPDK:

```
# cd /root
# git clone http://dpdk.org/git/dpdk
# cd dpdk
# git checkout v2.0.0
# make install T=$RTE_TARGET
# modprobe uio
# insmod $RTE_SDK/$RTE_TARGET/kmod/igb_uio.ko
```

11. Check the PCI addresses of the vVirtio vNICs:

```
# lspci | grep Ethernet
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 Ethernet controller: Red Hat, Inc Virtio network device
```

12. Use the DPDK binding scripts to bind the interfaces to DPDK instead of the kernel:

```
# $RTE_SDK/tools/dpdk_nic_bind.py –b igb_uio 00:03.0
# $RTE_SDK/tools/dpdk_nic_bind.py –b igb_uio 00:04.0
```

13. Download the BNG packages:

```
# wget https://01.org/sites/default/files/downloads/intel-data-plane-
performance-demonstrators/dppd-bng-v017.zip
```

14. Extract the DPPD BNG sources:

```
# unzip dppd-bng-v017.zip
```

15. Build the BNG DPPD application:

```
# yum -y install ncurses-devel
# cd dppd-BNG-v017
# make
```

16. Change the settings in `handle_none.cfg`:

```
# sed –i ‘s/s0]/]/g’ config/handle_none.cfg
```

17. Delete the following lines in `handle_none.cfg`:

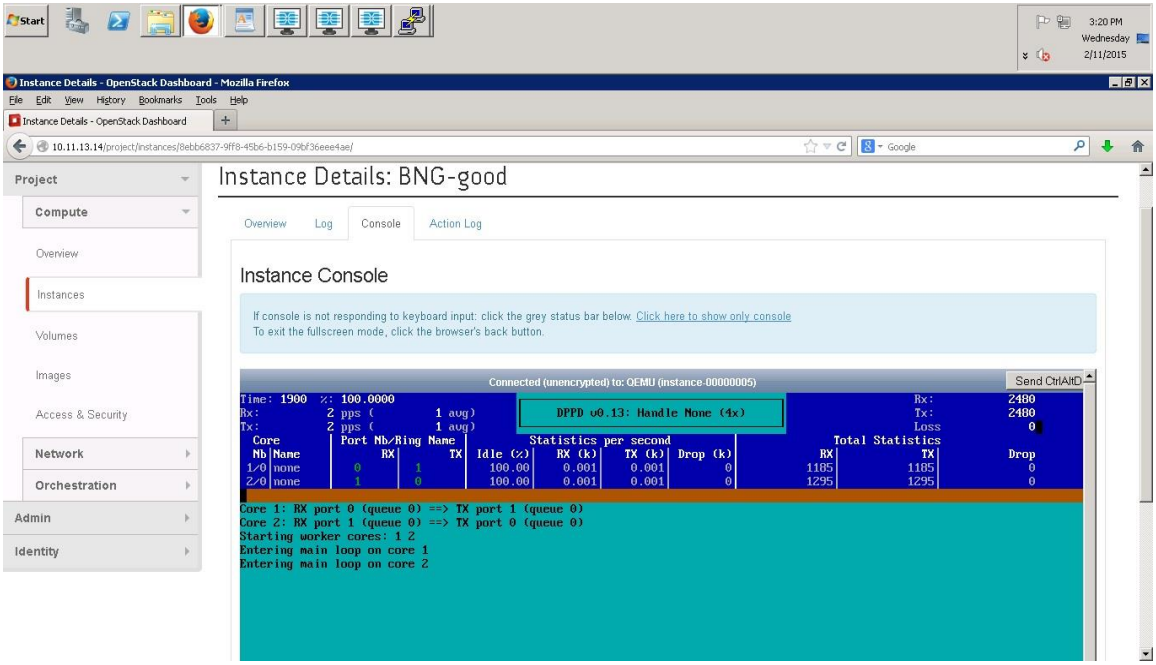
```
[port 2]
name=if2
mac=00:00:00:00:00:03
[port 3]
name=if3
mac=00:00:00:00:00:04

[core 3s0]
name=none
task=0
mode=none
rx port=if2
tx port=if3
drop=no
```

```
[core 4s0]
name=none
task=0
mode=none
rx port=if3
tx port=if2
drop=no
```

The application starts like this:

```
# ./build/dppd -f config/handle_none.cfg
```

When run under OpenStack, it should look like this.



Appendix A. Sample local.conf Files for OpenStack with ODL Controller

The following is a sample *local.conf* for the controller node:

```
# Controller node
#
[[local|localrc]]
FORCE=yes

HOST_NAME=$(hostname)
HOST_IP=10.11.12.16
HOST_IP_IFACE=enp3s0f0

PUBLIC_INTERFACE=ens786f1

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_service n-net
disable_service n-cpu

enable_service q-svc
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron
enable_service n-novnc
enable_service n-xvnc
enable_service n-crt
enable_service n-cauth

DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/xstack.sh.log
SYSLOG=True
LOGDAYS=1

MULTI_HOST=True
```

```
Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan,vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vxlan

ENABLE_TENANT_TUNNELS=True
PHYSICAL_NETWORK=physnet1

# ODL specific
enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
4f9f667774eb773e43cb71e6bc96c10951a43544
Q_HOST=$HOST_IP
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight

ODL_MGR_IP=10.11.12.16
ODL_PROVIDER_MAPPINGS=physnet1:ens786f0
ODL_LOCAL_IP=172.16.12.16
ODL_RELEASE=lithium-snapshot-0.3.1
ODL_MODE=allinone

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
novncproxy_host=0.0.0.0
novncproxy_port=6080
```

Here is a sample *local.conf* for the compute node:

```
# Compute node
# OVS_TYPE=ovs
[[local|localrc]]

FORCE=yes
MULTI_HOST=True

HOST_NAME=$(hostname)
HOST_IP=10.11.12.17
HOST_IP_IFACE=enp3s0f0
SERVICE_HOST_NAME=sdnlab-wp16
SERVICE_HOST=10.11.12.16

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST

GLANCE_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST
```

```
ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_all_services
enable_service n-cpu

DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/stack.sh.log
SYSLOG=True
LOGDAYS=1

Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan
Q_ML2_TENANT_NETWORK_TYPE=vxlan

ENABLE_TENANT_TUNNELS=True
PHYSICAL_NETWORK=physnet1

# ODL Specific
enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
4f9f667774eb773e43cb71e6bc96c10951a43544

Q_HOST=$SERVICE_HOST
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight

ODL_MGR_IP=10.11.12.16
ODL_PROVIDER_MAPPINGS=physnet1:ens786f0
ODL_LOCAL_IP=172.16.12.17
ODL_RELEASE=lithium-snapshot-0.3.1
ODL_MODE=compute

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```

Appendix B. Configuring the Proxy

This appendix describes how to configure the proxy in case the infrastructure requires it. Proxy settings are generally set as environment variables in the user's `.bashrc`:

```
$ vi ~/.bashrc
```

And add:

```
export http_proxy=<your http proxy server>:<your http proxy port>
export https_proxy=<your https proxy server>:<your http proxy port>
```

Also add the no proxy settings (i.e., the hosts and/or subnets that you do not want to use the proxy server to access), for example:

```
export no_proxy=192.168.122.1,<intranet subnets>
```

If you want to make the change across all users instead of just yours, make the above additions with */etc/profile* as root:

```
# vi /etc/profile
```

This allows most shell commands (e.g., `wget` or `curl`) to access your proxy server first.

Because yum does not read the proxy settings from your shell, you must also edit */etc/yum.conf* as root and add the following line:

```
# vi /etc/yum.conf
proxy=http://<your http proxy server>:<your http proxy port>
```

In order for git to also use your proxy servers, execute the following command:

```
$ git config --global http.proxy <your http proxy server>:<your http proxy port>
$ git config --global https.proxy <your https proxy server>:<your https proxy port>
```

If you want to make the git proxy settings available to all users as root, run the following commands instead:

```
# git config --system http.proxy <your http proxy server>:<your http proxy port>
# git config --system https.proxy <your https proxy server>:<your https proxy port>
```

For ODL deployments, the proxy needs to be defined as part of the XML settings file of Maven:

1. Create settings.xml to .m2/ directory, if it does not already exist:

   ```
   $ mkdir ~/.m2
   ```

2. Edit the ~/.m2/settings.xml file:

   ```
   $ vi ~/.m2/settings.xml
   ```

3. Add the following:

   ```
   <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
   http://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository/>
   <interactiveMode/>
   <usePluginRegistry/>
   <offline/>
   ```

```
<pluginGroups/>
<servers/>
<mirrors/>
<proxies>
  <proxy>
    <id>intel</id>
    <active>true</active>
    <protocol>http</protocol>
    <host>your http proxy host</host>
    <port>your http proxy port no</port>
    <nonProxyHosts>localhost,127.0.0.1</nonProxyHosts>
  </proxy>
</proxies>
<profiles/>
<activeProfiles/>
</settings>
```

Appendix C. Configuring Horizon UI to Deploy VMs

C.1 Custom VM Image, Availability Zone, and Flavor

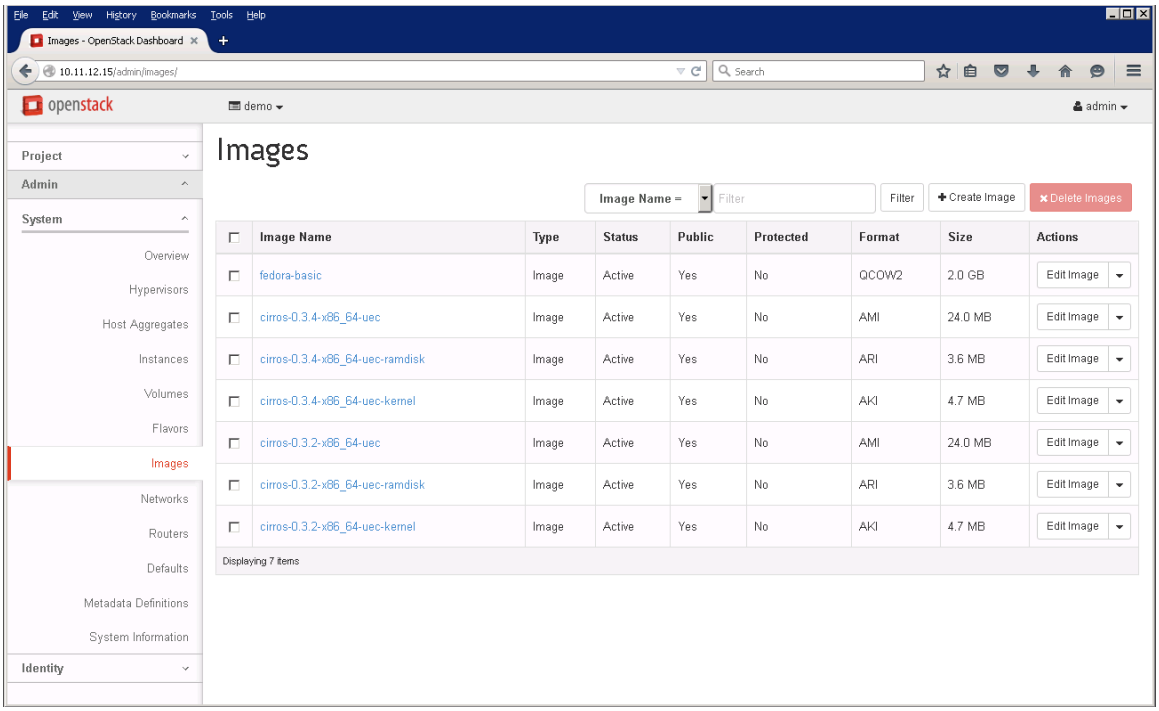
Users can create their own custom VM image to deploy. Horizon UI provides a few choices to accomplish this. These include the choice of flavor, format, architecture, etc. The following example provides instructions on how to use a customer VM image and create a host aggregate and an availability zone.

1. Use **admin** user to log in from the OpenStack dashboard with the IP address of the management port of the controller:

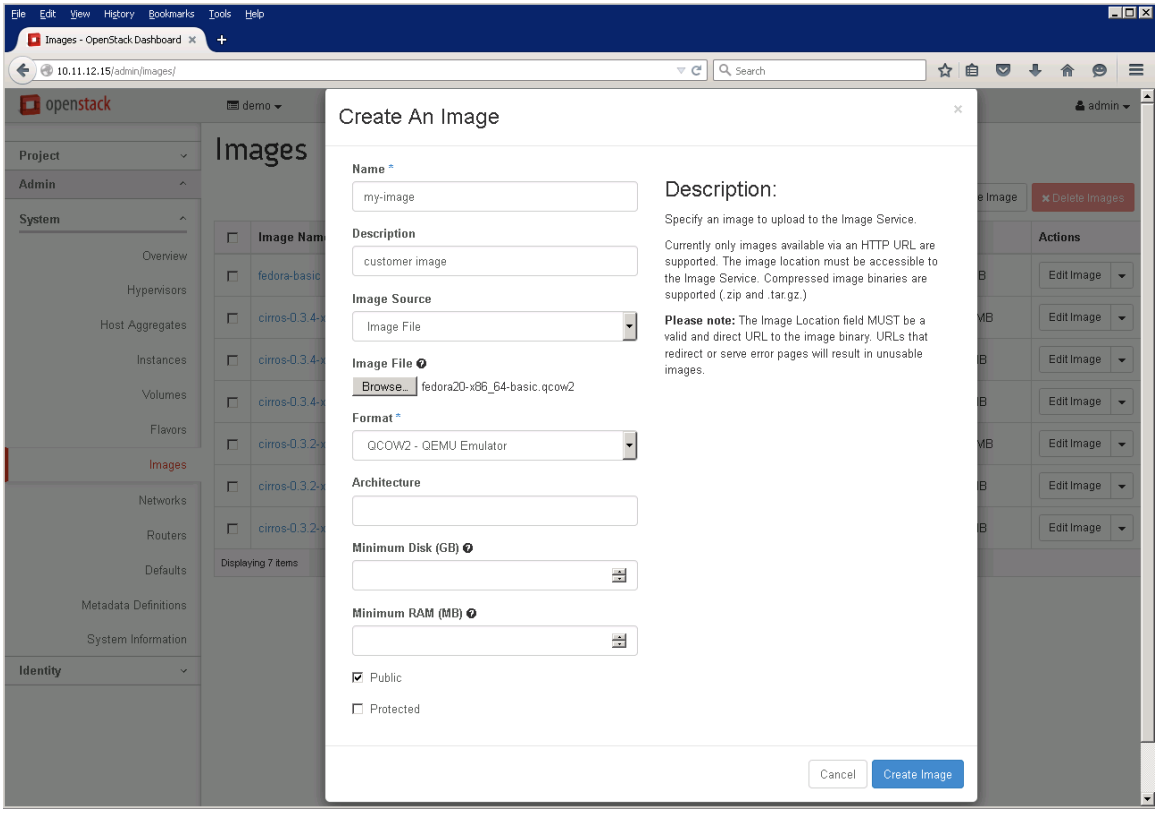
   http://10.11.12.15/

2. To create a VM image after logging in, click **Images** under **System** in the left pane and then the **Create Image** tab in the upper-right corner.

   Note: Ensure the project name at the top left of the page is "demo."

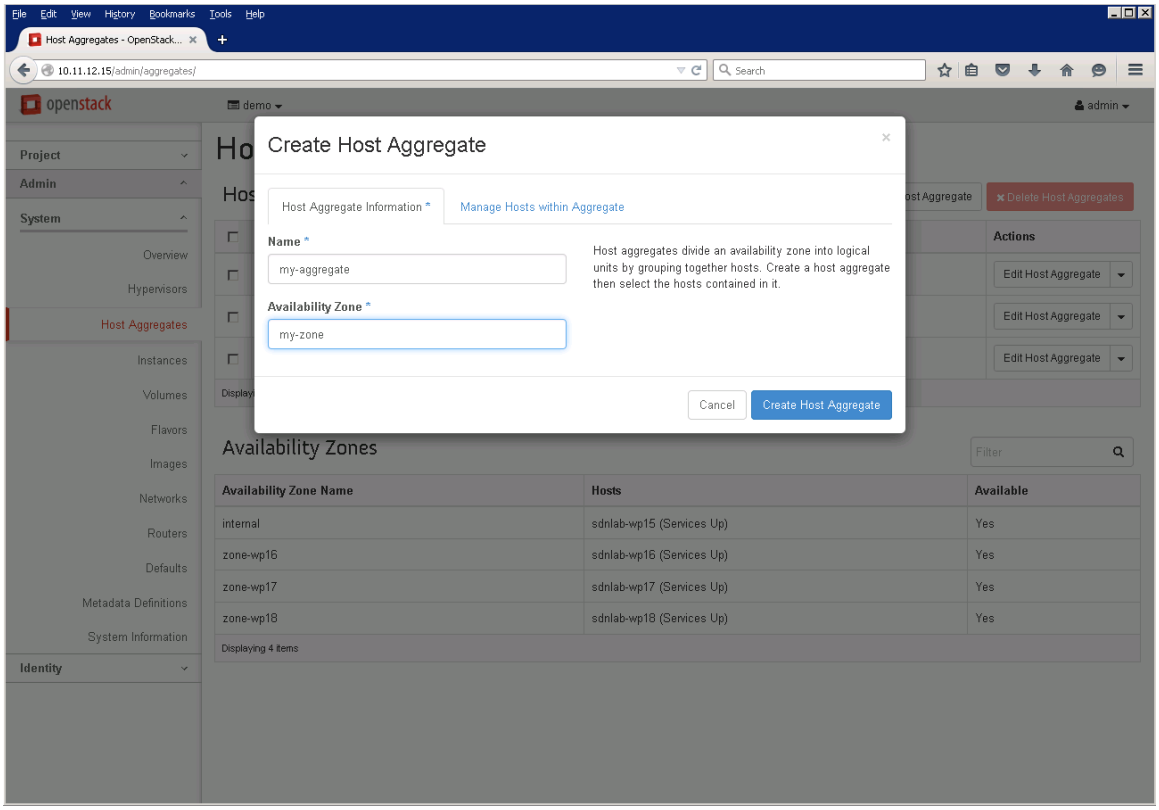


3. In the "Create An Image" window, enter the image name and description, select the image source (the source should be accessible by OpenStack), and check "Public" from the respective boxes. Click **Create Image** at the bottom right to load the image file to the Glance image server for OpenStack consumption.

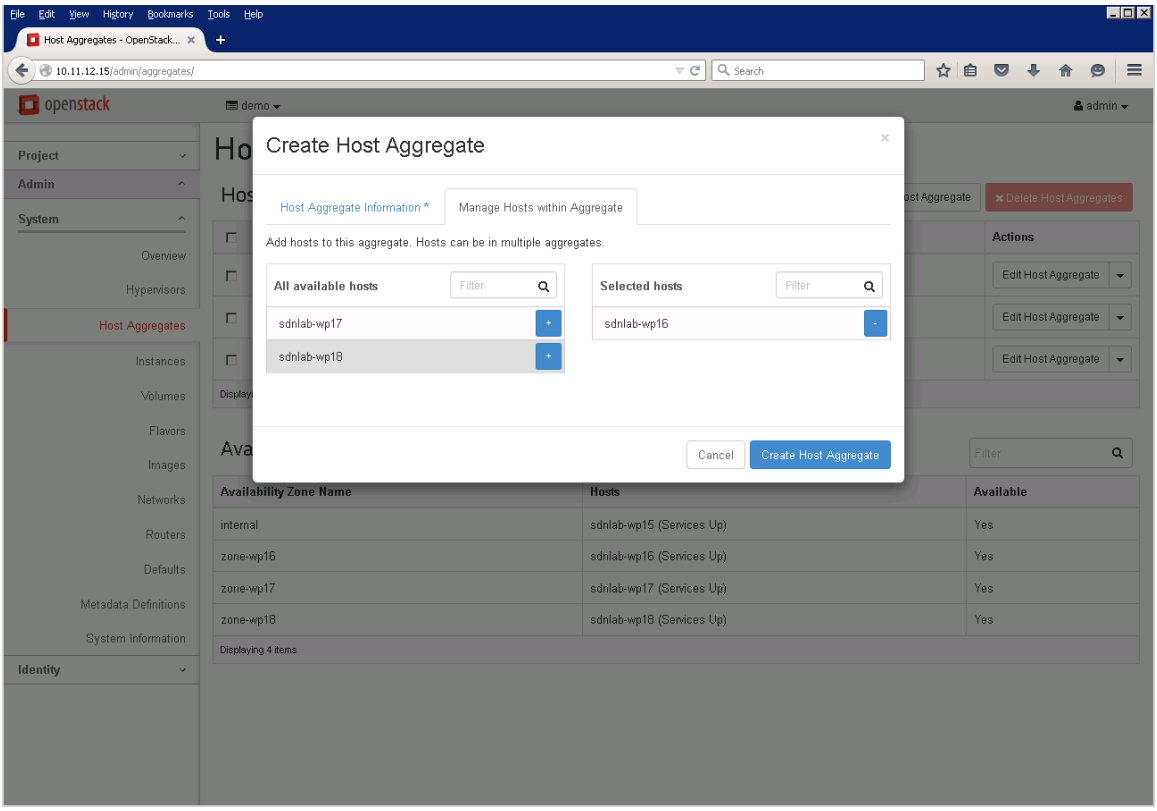


4. Create the availability zone and host aggregate by clicking **Host Aggregates** under **System Panel** on the left pane and then **Create Host Aggregate** in the upper-right corner.

5. In the **Create Host Aggregate** window, enter the names of the aggregate and availability zone.

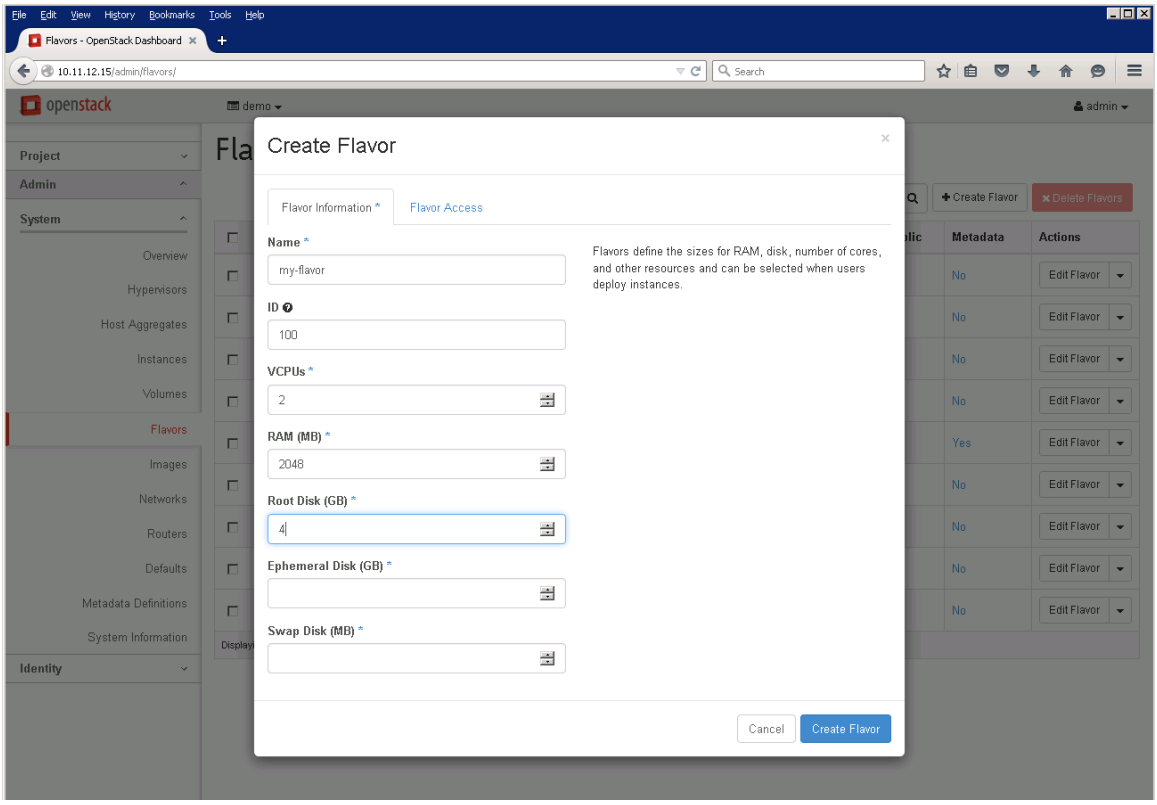


6. Click the **Manage Hosts within aggregate** tab (all available hosts are listed), and then select host(s) to add into the aggregate.



7. Click Create Host Aggregate to finish.
8. Create flavor by clicking **Flavors** under **System Panel** on the left pane and then **Create Flavor** in the upper-right corner.

9. In the **Create Flavor** window, enter the names of the flavor, ID (or select "auto" to let OpenStack generate one for you), number of VCPU, memory and disk sizes, and then click **Create Flavor** to complete.

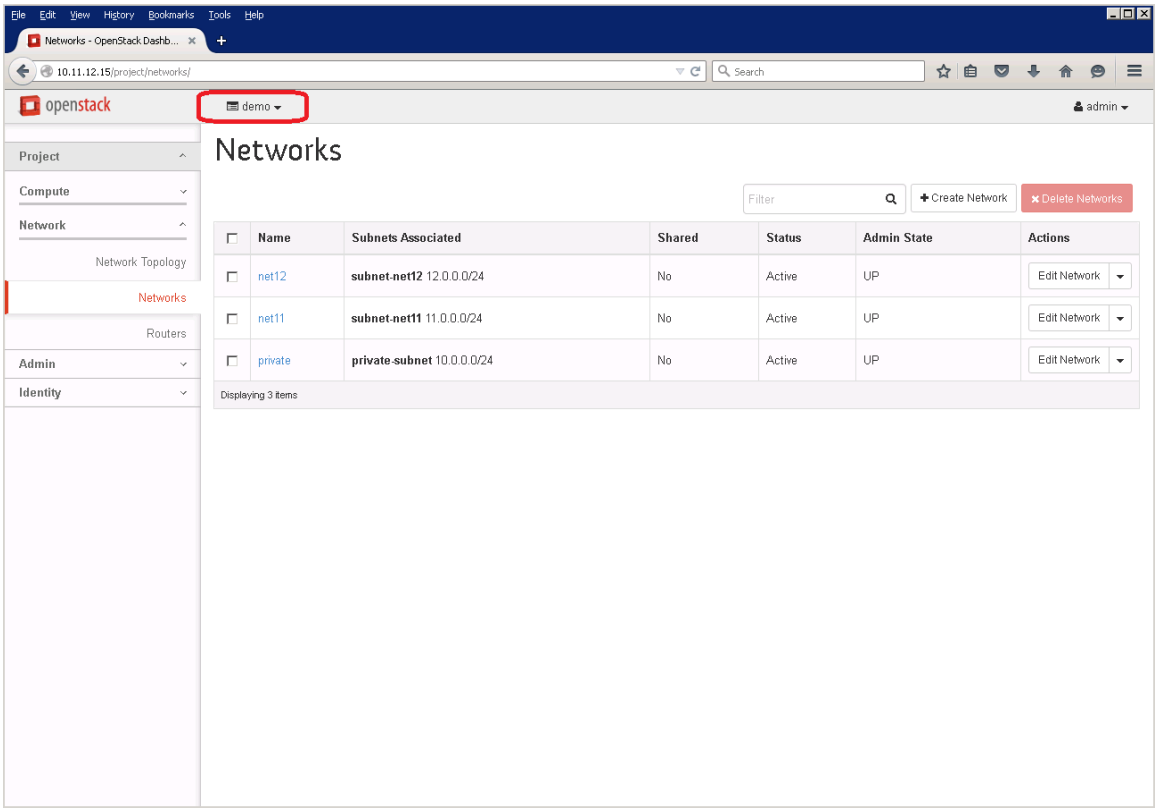


C.2 Creating Additional Networks

The following example describes how to create a new network in an OpenStack environment and apply it to a VM to enable the VM to have multiple network interfaces. To do this, follow these steps:

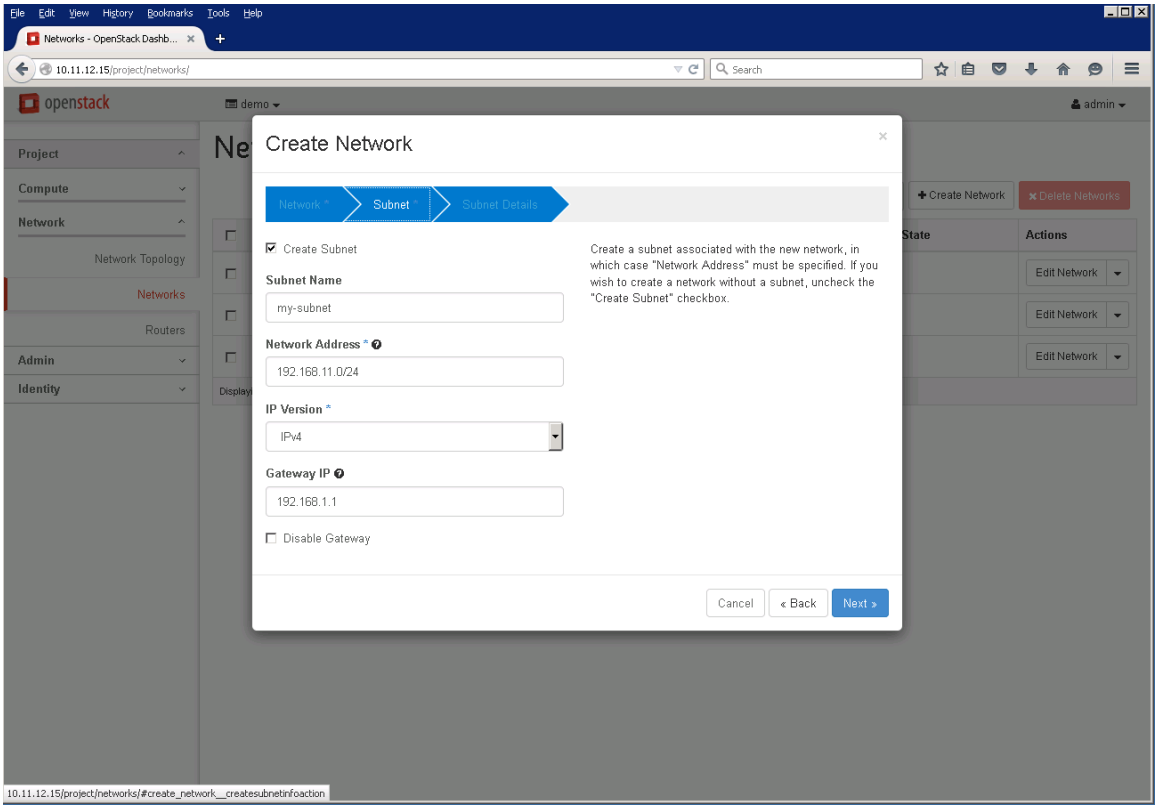
1. Use **demo** user to log in to Horizon.
2. Click the **Network/Networks** tab in the left pane.
3. Click **Create Network** in the upper-right corner.

   Note: If you log in as **admin** user, make sure the user is "demo" (highlighted in red below) from the project drop-down box at the top left of the screen.



4. In the **Create Network** window, click the **Network** tab, then enter the network name. Click the **Subnet** tab, enter a subnet name, network address range, and gateway, and then click **Next** to continue.

   Note: Users can ignore DNS and the router setup, and complete creating the network.



5. Click **Finish** to complete creating the network.

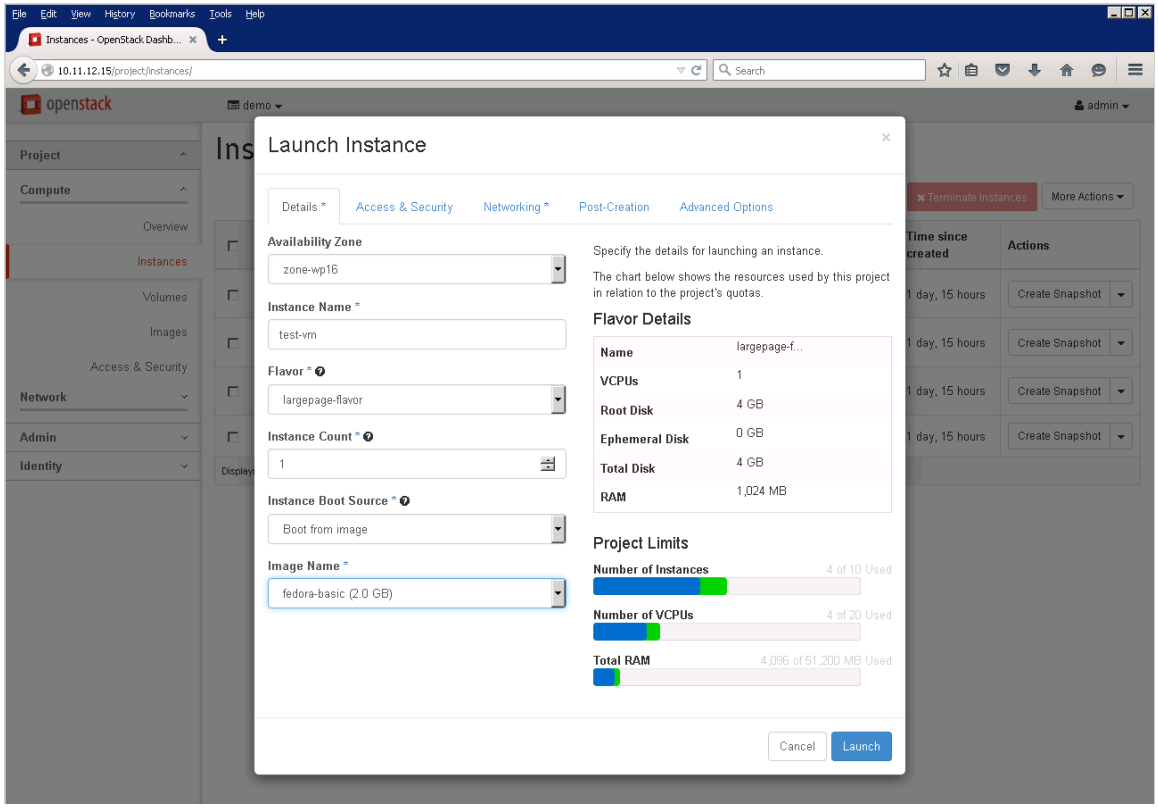
C.3 VM Deployment

The following example describes how to use customer VM image, flavor, availability zone, and networks to launch a VM in an OpenStack environment.

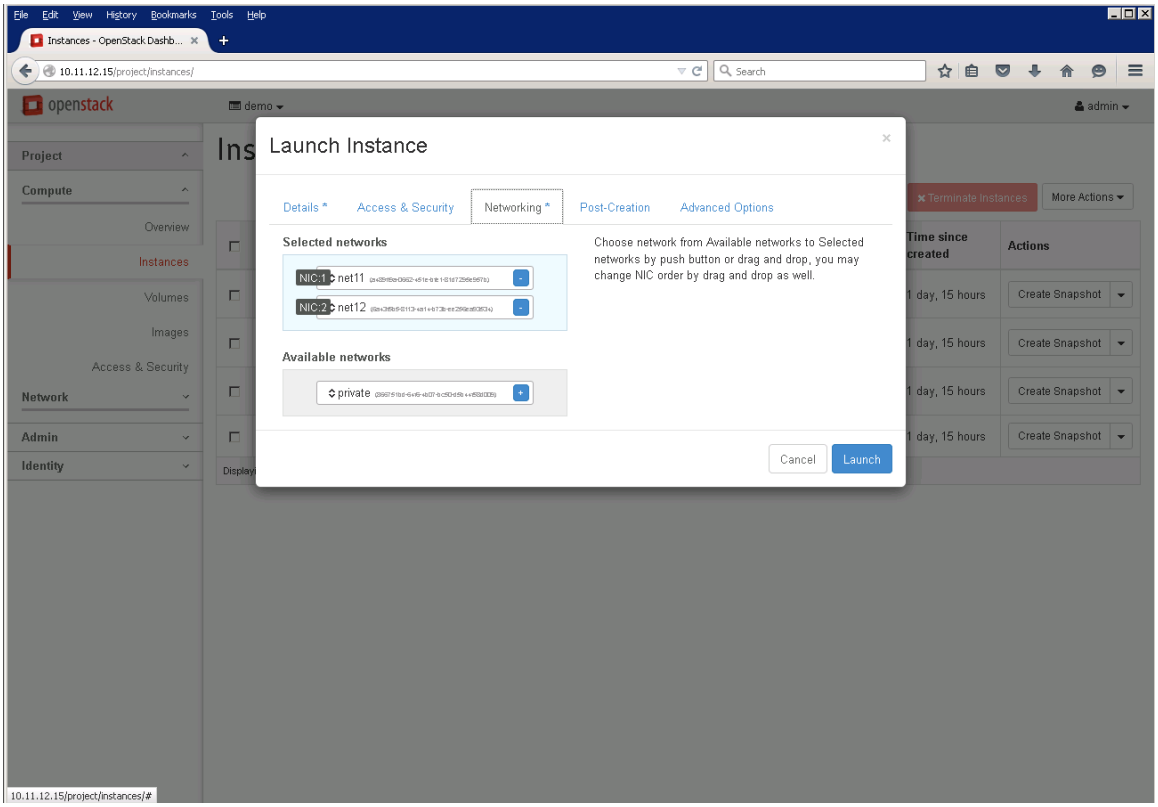
1. Log in as **demo** user.

   Note: If you log in as **admin** user, make sure the user is "demo" from the project drop-down box on top left of the page. Refer to C.2, Creating Additional Networks.

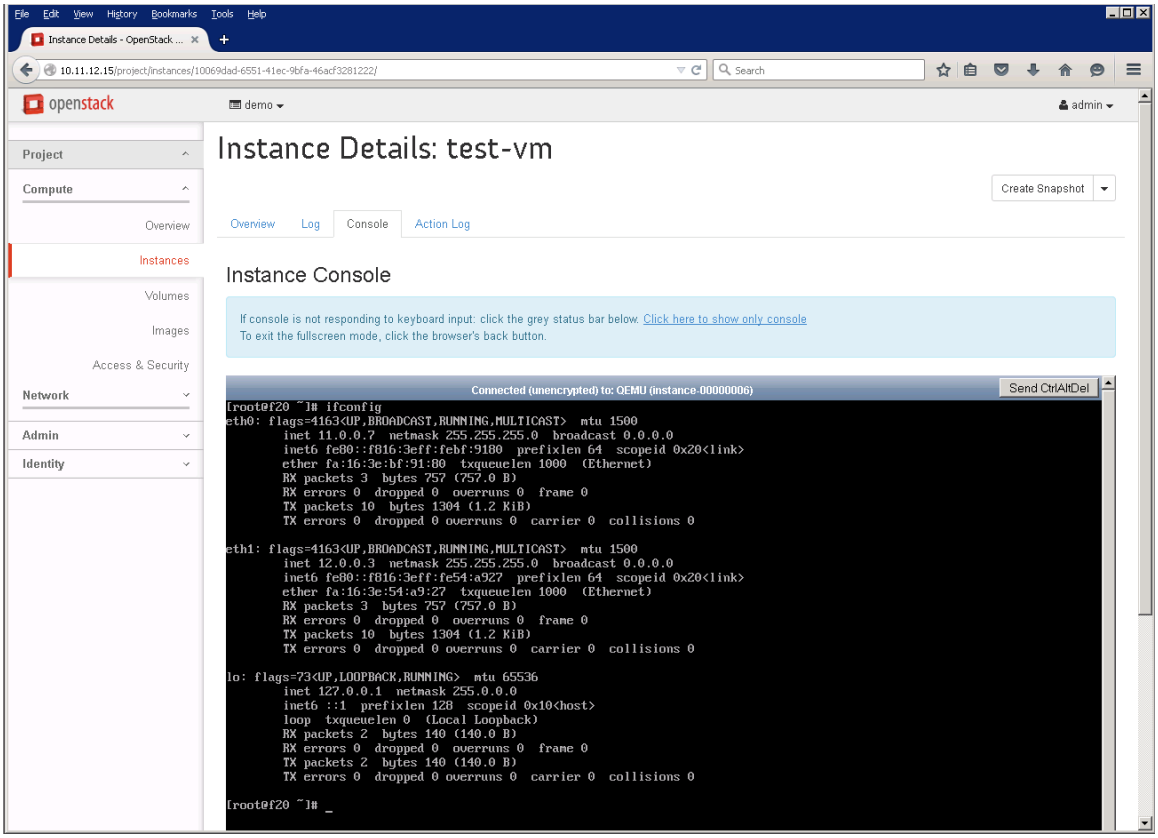
2. Click the **Instances** tab under **Project > Compute** in the left pane. Click the **Launch Instance** tab at the upper-right corner in the new window, and then select the desired availability zone, instance name, flavor, and instance boot source from the respective drop-down boxes.



3. Click the **Networking** tab, and then select one or more networks for the instance. All networks, including the default network, private, and newly created ones, are available for selection. A virtual network interface (NIC) will be formed for the VM for each network selected. Therefore, if the VM needs two or more NICs, you should select two or more networks accordingly.



4. Click **Launch** to finish. The instance has two network interfaces, eth0 and eth1, belonging to two different networks.



5. The new VM should be up and running in a minutes or so. Click the new name of the VM from the list, and then click **Console** in the top menu to access the VM.

Appendix D. Acronyms and Abbreviations

Abbreviation	Description
ATR	Application Targeted Routing
BIOS	Basic Input/Output System
BNG	Broadband (or Border) Network Gateway
BRAS	Broadband Remote Access Server
DHCP	Dynamic Host Configuration Protocol
DLUX	OpenDaylight User eXperience
DPDK	Data Plane Development Kit
HTT	Hyper-Threading Technology
IDS	Intrusion Detection System
IOMMU	Input/Output Memory Management Unit
IPS	Intrusion Prevention System
KVM	Kernel-based Virtual Machine
MTU	Maximum Transmission Unit
NFV	Network Functions Virtualization
NIC	Network Interface Card
NTP	Network Time Protocol
NUMA	Non-Uniform Memory Access
ODL	OpenDaylight
ONP	Open Network Platform
ONPS	Open Network Platform Server
OVS	Open vSwitch
RT Kernel	Real-Time Kernel
SDN	Software Defined Networking
SR-IOV	Single Root I/O Virtualization
vBNG	Virtual Broadband (or Border) Network Gateway
VM	Virtual Machine
VNF	Virtualized Network Function

Appendix E. References

Document Name	Source
Intel® Xeon® Processor E5-2699 v3	http://ark.intel.com/products/81061/Intel-Xeon-Processor-E5-2699-v3-45M-Cache-2_30-GHz
Intel® Xeon® Processor E5-2697 v3	http://ark.intel.com/products/81059/Intel-Xeon-Processor-E5-2697-v3-35M-Cache-2_60-GHz
Intel® Ethernet Converged Network Adapter XL710-QDA2 2 x 40 GbE	http://ark.intel.com/products/83967/Intel-Ethernet-Converged-Network-Adapter-XL710-QDA2
Intel® Ethernet Converged Network Adapter X710-DA4 4 x 10 GbE	http://ark.intel.com/products/83965/Intel-Ethernet-Converged-Network-Adapter-X710-DA4
DPDK	http://www.intel.com/go/dpdk
OpenvSwitch with DPDK-netdev	https://01.org/packet-processing
OpenDaylight Lithium	https://www.opendaylight.org/lithium
OpenStack	https://www.openstack.org/
DevStack	http://docs.openstack.org/developer/DevStack/
Suricata	http://suricata-ids.org/

Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer. Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

Intel does not control or audit third-party websites referenced in this document. You should visit the referenced website and confirm whether referenced data are accurate.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.