Intel® Open Network Platform Release 2.0 Reference Architecture Guide

SDN/NFV Solutions with Intel® Open Network Platform

Document Revision 1.1
April 2016

Revision History

Date	Revision	Comments
January 22, 2016	1.0	Initial release of Intel® Open Network Platform Release 2.0
April 6, 2016	1.1	Minor clarifications and typo corrections in hardware and software specifications

Contents

Figures

Tables

1.0 Audience and Purpose

Intel® Open Network Platform (Intel® ONP) is a Reference Architecture that provides engineering guidance and ecosystem-enablement support to encourage widespread adoption of Software Defined Networking (SDN) and Network Functions Virtualization (NFV) solutions in Telco, Enterprise, and Cloud.

The primary audiences for this document are architects and engineers developing or testing SDN/NFV solutions and looking to use the Intel® Open Network Platform Reference Architecture for reference. This document provides step by step instructions on how to build, configure and operate the system using open-source software. Ingredients include the following:

- OpenStack*
- OpenDaylight*
- Data Plane Development Kit (DPDK)*
- Open vSwitch* (OvS)
- Fedora 22*
- CentOS-7.1*

This document provides a guide for integration of these software elements on the Intel® Architecture (IA) platform for the Intel® Open Network Platform (Intel® ONP) reference architecture. The content includes high-level architecture, setup, configuration and provisioning 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 with Intel® ONP.

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

Please note that Intel® offers a scripts available on 01.org to facilitate the installation of Intel® ONP reference software stack.

2.0 Summary

The Intel® ONP 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.

Supported processor families include:

- Intel® Xeon® processor E5-2600 v3 product family
- Intel® Xeon® processor D-1500 family
- Intel® Atom™ processor C2000 product family

Supported networking adapters include:

- Intel® Ethernet Server Adapter X520 Series
- Intel® Ethernet Converged Network Adapter X540-T2
- Intel® Ethernet Converged Network Adapters XL710-QDA2 and X710-DA4.

Supported host operating systems include:

- Fedora 22 with QEMU-KVM virtualization technology
- CentOS-7.1 with QEMU-KVM virtualization technology.

Additional software ingredients include Intel® DPDK, Open vSwitch, Open vSwitch with DPDK, 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 Release 2.0 Release Notes, section 3, available on 01.org.

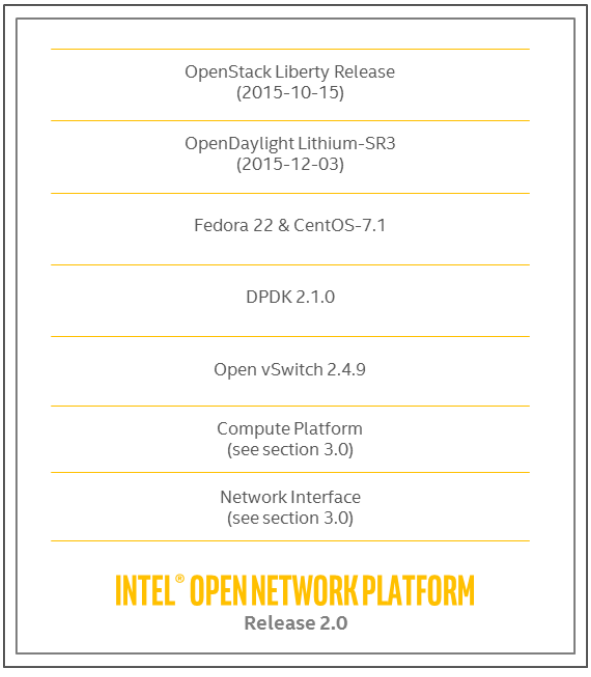

Figure 2-1 Intel® ONP — Hardware and Software Ingredients

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

- Verify communication between the Intel® ONP Controller and Intel® ONP Compute Nodes.
- Validate basic controller functionality.
- Demonstrate how to deploy virtualized network functions (VNFs) and showcase traffic processed within them. Two open source VNFs, virtual Intrusion Prevention System (vIPS) and virtual Border Network Gateway (vBNG), are used as examples to demonstrate the usage.

This way, Intel® aims to show users how to apply the "plumbing" tailored for great efficiency and optimized performance for IA that enables more CPU cycles to be dedicated to run users' 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 Planes) run on one server, and the two Compute Nodes (Data Plane) run on two individual server nodes. The differences in the network configuration to enable this setup are shown with the management and data ports. Note that many variations of this setup can be deployed.

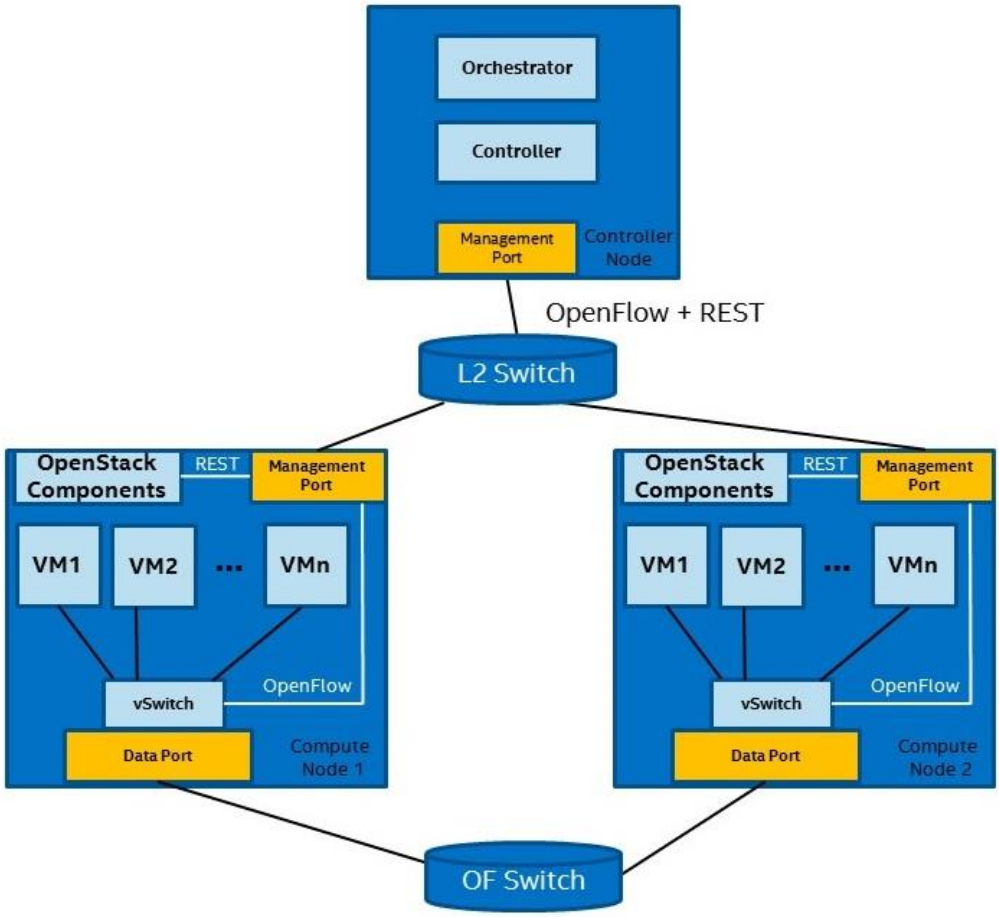

Figure 2-2 Generic Setup with Controller and two Compute Nodes

2.1 Network Services Examples

The network services presented in this section are included as examples that have been tested with Intel® ONP 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 http://suricata-ids.org.

With the ONP2.0 deployment we showcase how the traffic across 2 VNFs can be controlled using an intermediate Suricata based IPS VNF. This intermediate VNF can be either on same Compute Node or a Remote Compute node. Refer to the section 7.2 Installation and Configuration of vIPS for more information on running IPS as a VNF.

2.1.2 vBNG (Broadband Network Gateway)

A BNG, may also be known as a Broadband Remote Access Server (BRAS), 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.

Intel® PROX (Packet pROcessing eXecution engine) tool is used to demonstrate workload functionality of a virtual BNG as a VNF. It is based on DPDK libraries to accelerate data plane packet processing. The PROX tool is configured to perform Layer 3 forwarding in a VNF acting as an intermediate node across two VNFs. With the options provided by the PROX tool the input and output characteristics of the traffic and packet processing across multiple cores with in the VNF can be configured among its different options. This way we emulate the functionality of a BNG using a PROX based VNF. Refer to 7.3 Installation and Configuration of vBNG, or Appendix A: Configuring Proxy for OpenDaylight for more information on running BNG as an appliance.

Intel® PROX using DPDK can be found at https://01.org/intel-data-plane-performance-demonstrators/downloads/prox-application-v021.

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.

3.0 Hardware Components

Table 3-1, Table 3-2 and Table 3-3 provide details of platform hardware components used for testing purposes. The Notes column describes some of the fine tunings enabled on the hardware.

Table 3-1 Intel® Xeon® processor E5-2600 v3 product family-based platforms - hardware ingredients used in integration tests

Item	Description	Notes
Platform	Intel® Server Board S2600WTT	Intel® Xeon® processor-based DP server (Formerly Wildcat Pass) 2 x 10GbE integrated LAN ports, 120 GB SSD 2.5in SATA 6GB/s Intel® Wolfsville SSDSC2BB120G4 Supports SR-IOV
Processors	Dual Intel® Xeon® processor E5-2697 v3	Formerly Haswell 14 cores, 28 threads, 2.6-3.6 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.
	Dual Intel® Xeon® processor E5-2699 v3	Formerly Haswell 18 cores, 36 threads, 2.3-3.6 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	64 GB total; Crucial CT8G4RFS4213	8x DDR4 RDIMM 2133 MHz, 8 GB
NICs	Intel® Ethernet Converged Network Adapter X710-DA4	Intel® Ethernet Controller XL710-AM1 (Formerly Fortville) 4 x 10 GbE ports Firmware version f4.33 a1.2 n04.42 Tested with Intel® FTLX8571D3BCV-IT and AFBR-703sDZ-IN2 transceivers
	Intel® Ethernet Converged Network Adapter XL710-QDA2	Intel® Ethernet Controller XL710-AM2 (Formerly Fortville) 2 x 40 GbE ports Firmware version f4.33 a1.2 n04.42 Tested with Intel® E40QSFPSR transceiver
Intel® QuickAssist Technology	Intel® QuickAssist Adapter 8950	Intel® QuickAssist Adapter 8950 (Formerly Walnut Hill) Provides IPSec, SSL Acceleration and Compression services Support for SR-IOV PCIE Gen 3 (8GT/s)
BIOS	SE5C610.86B.01.01.0009.060120151350 Release Date: 03/19/2015	Hyper-Threading enabled Intel® Virtualization Technology (Intel® VT-x) enabled Intel® VT for Directed I/O (Intel® VT-d) enabled

Table 3-2 Intel® Xeon® processor D-1500 family-based SoC platforms - hardware ingredients used in integration tests

Item	Description	Notes
Platform	SuperMicro SuperServer 5018D-FN4T	Intel® Xeon® processor-based SOC server Motherboard: SuperMicro X10SDV-8C-TLN4F Dual LAN via Intel® i350-AM2 Gigabit Ethernet Dual LAN via SoC 10GBase-T 500 GB HDD 3.5in SATA 6GB/s 7200RPM 16MB Seagate Barracuda ST500DM002
Processors	Intel® Xeon® processor D-1540	Formerly Broadwell-DE 8 cores, 16 threads, 2-2.6 GHz, 12 MB cache Single Socket FCBGA 1667 supported CPU TDP 45W System-on-Chip
	Intel® Xeon® processor D-1520	Formerly Broadwell-DE 4 cores, 8 threads, 2.2-2.6 GHz, 6 MB cache Single Socket FCBGA 1667 CPU TDP 45W System-on-Chip
Memory	32 GB total; Kingston KVR21R15S4/8	4x DDR4 RDIMM 2133 MHz, 8 GB
BIOS	AMIBIOS Version: 1.0a Release Date: 05/27/2015	Hyper-Threading enabled Intel® Virtualization Technology (Intel® VT-x) enabled Intel® VT for Directed I/O (Intel® VT-d) enabled

Table 3-3 Intel® Atom™ processor C2000 product family-based SoC platforms - hardware ingredients used in integration tests

Item	Description	Notes
Platform	SuperMicro SuperServer 5018A-FTN4	Intel® Atom™ processor-based SoC server Motherboard: SuperMicro A1Sri-2758F 4 x 1GbE integrated Intel® Ethernet C2000 SoC I354 Quad GbE LAN ports, 120 GB SSD 2.5in SATA 6GB/s Intel® Wolfsville SSDSC2BB120G4
Processor	Intel® Atom™ processor C2758	8 core, 8 threads, 2.4 GHz, 4 MB cache CPU TDP 20W (8-Core) FCBGA 1283 System-on-Chip
Memory	32 GB Total; 1600MHZ DDR3L ECC CL11 SODIMM 1.35V	4x 204-pin DDR3 SO-DIMM slots
BIOS	AMIBIOS Version: 1.1 Release Date: 01/09/2015	

4.0 Software Versions

The Table 4-1 describes functions of the software ingredients 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 22	Host Operating System	Fedora 22 Server x86_64 Kernel version: 4.1.10-200.fc22.x86_64
CentOS-7.1	Host Operating System	CentOS-7 (1503) x86_64 DVD ISO Kernel version: 3.10.0-229.el7.x86_64
Real-Time Kernel	Targeted towards the Telco environment, which is sensitive to low latency	Fedora 22 Real-Time Kernel version: 3.18.24-rt22 CentOS-7.1 Real-Time Kernel version: 3.10.93-rt101-rebase
QEMU-KVM	Virtualization technology	QEMU-KVM version: 2.3.1-7.fc22.x86_64 libvirt version: 1.2.13.1-3.fc22.x86_64
DPDK	Network stack bypass and libraries for packet processing; includes user space vhost drivers	2.1.0
Open vSwitch	vSwitch	Open vSwitch 2.4.9 Commit ID 88058f19ed9aadb1b22d26d93e46b3fd5eb1ad32 used for: Open vSwitch (non-DPDK nodes) Open vSwitch with DPDK
OpenStack	SDN orchestrator	OpenStack Liberty Release (2015-10-15)
OpenDaylight	SDN controller	OpenDaylight Lithium-SR3 (2015-12-03)
Intel® Ethernet Drivers	Ethernet drivers	ixgbe-4.3.9 (Intel® Xeon® processor D-1500 family deployments) Inbox i40e driver (Intel® Ethernet Converged Network Adapters X710-DA4 and XL710-QDA2) igb-5.2.15-k (Intel® Atom™ processor C2000 family deployments)

4.1 Obtaining Software Ingredients

All of the open-source software ingredients involved are downloaded from the source repositories shown in the Table 4-2. Commit IDs for major OpenStack components are shown in the Table 4-3.

Table 4-2 Sources for Software Ingredients

Software Component	Location	Comments
Fedora 22	http://mirror.us.leaseweb.net/fedora/linux/releases/22/Server/x86_64/iso/	https://getfedora.org/en/server/ Fedora-Server-DVD-x86_64-22.iso
CentOS-7.1	http://vault.centos.org/7.1.1503/isos/x86_64/	https://www.CentOS.org/download/ CentOS-7-x86_64-DVD-1503-01.iso
Real-Time Kernel	git clone https://www.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git/ checkout v3.18.24-rt22 (for Fedora) or checkout v3.10.93-rt101-rebase (for CentOS)	
DPDK	git clone http://dpdk.org/git/dpdk checkout 7173acefc7cfdfbbb9b91fcba1c9a67adb4c07c9, v2.1 from master	Includes DPDK Poll Mode Driver, sample apps (bundled) DPDK download will be done through the DevStack script during installation.
Open vSwitch	git clone https://github.com/openvswitch.ovs.git checkout 88058f19ed9aadb1b22d26d93e46b3fd5eb1ad32	OvS download will be done through the DevStack script during installation.
OpenStack	OpenStack Liberty Release (2015-10-15)	Deployed using DevStack (see next row). The commit IDs for the various OpenStack components are provided in the Table 4-3.
DevStack	git clone https://github.com/openstack-dev/devstack.git checkout stable/liberty 6ff54a3936d3b1999d625019889c2e5894be396d	
networking-ovs-dpdk ML2 Plugin	git clone https://git.openstack.org/cgit/openstack/networking-ovs-dpdk checkout stable/liberty 6a8821c69b9154bc432f86c62e478d14f1c6fcb5	
OpenDaylight	Lithium-SR3 (2015-12-03)	OpenDaylight download will be done through the DevStack script during installation.
Intel® ONP Release 2.0 Scripts	https://01.org/packet-processing/intel®-onp-servers	Includes helper scripts to set up ONP 2.0 using DevStack
Suricata	`# yum install suricata`	Suricata version 2.0.2-1.fc20.x86_64 was used in integration tests
vBNG using PROX	https://01.org/intel-data-plane-performance-demonstrators/downloads/prox-application-v021	
Intel® Ethernet Drivers	http://sourceforge.net/projects/e1000/files/ixgbe%20stable/4.3.9/ixgbe-4.3.9.tar.gz	ixgbe-4.3.9 for Intel® Xeon® processor D-1500 family deployments igb-5.2.15-k (kernel in-built driver) for Intel® Atom™ processor C2000 family deployments

Table 4-3 Commit IDs for Major OpenStack Components

OpenStack Component	Commit ID, Tag, or Branch
OpenStack Nova	stable/liberty 6df6ad3ff32f2b1fe2978df1032002548ad8eb66
OpenStack Neutron	stable/liberty 6dcfe3a9362ae5fcf18e5cfb59663e43446cd59c
OpenStack Keystone	stable/liberty 8dcd82fb9c76d43f26338bee293b32f4af473ad9
OpenStack Glance	stable/liberty 69516fad5f651a085a047a337a05c58b39023c1b
OpenStack Horizon	stable/liberty 593f0b78eea8efbb6d833d66acc7ab4dc852159b
OpenStack Cinder	stable/liberty 61026d4e4f2a58dd84ffb2e4e40ab99860b9316a
OpenStack Requirements	stable/liberty 55c6058d302919fc6c435e9b7791fb8c5e680ba1
OpenStack Tempest	stable/liberty a1edb75d7901a9e338ab397d208a40c99c5fd9a1
OpenStack noVNC	stable/liberty 6a90803feb124791960e3962e328aa3cfb729aeb
OpenStack networking-odl	stable/liberty fec09899a610d565baadbf33713b57e760aa27a5
OpenStack networking-ovs-dpdk	6a8821c69b9154bc432f86c62e478d14f1c6fcb5

Note: See Intel® ONP Release 2.0 Scripts for commit IDs of minor components.

Note: Due to the number of ingredients involved, follow the instructions provided by the scripts and execute them in order to deploy and provision each of the software components.

Note: For the sake of simplicity, this document uses `yum` command for installing packages. As of Fedora 22, `yum` has been replaced by `dnf`.

5.0 Installation and Configuration Guide

This section describes the installation and configuration instructions to prepare the Controller and Compute Nodes.

5.1 BIOS Settings

For Intel® Xeon® processor based platforms, during boot time, enter the BIOS menu and update the following configuration as described in the 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 Operating System Installation and Configuration

Current Intel® ONP scripts support two operating systems – Fedora 22 and CentOS-7.1. Details below provide generic instructions for installing and configuring the operating system of choice. 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.

1. Download the image of Linux distribution supported by Intel® ONP from the source given in the Table 4-2.
2. Burn the ISO file to a DVD and create an installation disk.
3. Use the DVD to install the operating system. During the installation, click **Software selection**, then choose the following:
   - Development Tools
   - Virtualization
4. 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.

5.3 Automated Installation Using Scripts

The Intel® ONP chooses DevStack for quick deployment of OpenStack in order to use its latest features, including, at times, its experimental ones. DevStack provides and maintains tools for the installation of OpenStack from upstream sources. Its main purpose is to support OpenStack development and testing of the components involved.

Due to its evolving nature, DevStack does not provide production-level stability. Since DevStack depends on the upstream sources in the internet to install the packages, there may be instability in the downloading process (for Ex. SSL certificates updates, servers maintenance works etc.). If you see a failure in installation of DevStack due to difficulty in downloading the packages from internet, please wait for some time and try again.

In order to have a predictable and validated setup using the scripts provided for the installation, it is optimal to proceed with the base set of Intel-recommended settings. 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 and have been validated with hardware described in the section 3.0 Hardware Components.

Before continuing with the scripts, update the BIOS with the settings presented in Table 5-1. Installation scripts can be obtained from the download link in the section 4.1 Obtaining Software Ingredients.

Note: The automation scripts are only tested on Fedora 22 and CentOS-7.1 operating systems and will not necessarily work on other distributions of Linux.

5.3.1 Intel® ONP Scripts

Intel® ONP scripts provide a simplified procedure for installing and configuring OpenStack using DevStack. Based on the type of deployment preferred, the user will need to update a configuration file. The scripts will take care of preparing the system and deploying the software ingredients, including:

- Configuring, provisioning and updating the required operating system kernel, services and network settings.
- Installing OpenStack using DevStack installer.
- Configuring the server as either a Controller or Compute node, according to the configuration file.

The installation script tarball contains the following files to accelerate DevStack deployment:

README

This file provides instructions on how to update the onps_config configuration file. It is highly recommended to read this file before attempting installation.

onps_config

A configuration file which determines the behavior of the installation scripts. The Intel® ONP deployment is based on the details provided in this file by user. Following are the current options provided to the user:

- the choice to configure the system as a Controller or Compute Node
- the choice to provision the Controller or Compute Node with vanilla Open vSwitch or accelerated Open vSwitch using the Data Plane Development Kit (DPDK).
- the choice of the overlay network type to configure: VXLAN or VLAN
- the choice to install the OpenDaylight (ODL) SDN controller
- the hostname of the node
- the network interfaces to use for each of the OpenStack networks
- any proxy information
- the kernel to use.

Note: The selections in this file will determine the content of the local.conf configuration file required for DevStack installation.

prepare_system.sh

Using the user provided hostname, proxy, kernel and network interface information, this script:

- Configures network interfaces and services.
- Configures (and optionally updates) the kernel.
- Enables or disables system services.
- Pulls required updates for Fedora or CentOS operating systems.
- Creates the DevStack configuration file local.conf.

prepare_stack.sh

Once the system is configured with prepare_system.sh, it is ready for OpenStack deployment. This script:

- Executes DevStack installer script, stack.sh.
- Configures OvS and DPDK parameters based on the overlay network.
- Finalizes network settings.

onps_commit_ids

Intel® ONP uses software components from multiple open source repositories. In order to have a predictable installation and working setup specific commit IDs are used to ensure validated components are installed. This file contains list of frozen commit ids including, but not limited to OpenStack, OvS and DPDK components.

create_local_conf.sh

Based on the user's choices updated in onps_config file this script is used to create the DevStack configuration file local.conf. This script detects the options given in the onps_config file, creates a local.conf file under "/home/<username>" directory and populates the parameters required for DevStack to deploy OpenStack as per user's requirements.

settings.xml

This file is used by OpenDaylight SDN controller to configure proxy related information. Initially as a template, this file will be copied to "/root/.m2" and modified accordingly via prepare_system.sh.

samples/*

DevStack uses a configuration file, local.conf, to setup and configure OpenStack services. The local.conf file is generated by create_local_conf.sh based on the user choices provided in the onps_config file. As a point of reference, a set of sample local.conf configuration files are provided in the "samples" directory with various combinations of deployment. These files can be used as example configuration files for deployments of choice.

5.4 Controller and Compute Node Setup

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). It is assumed that the user has successfully followed the hardware and software installation and configuration detailed in sections above.

Note: This procedure must be repeated on each node in the environment.

5.4.1 Network Configuration and Requirements

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.

5.4.2 Network Configuration Example

The following is an example on 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 OpenDaylight SDN 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.

Note: Among these interfaces, the interface for the tenant network (in this example, ens786f0) is used for DPDK and OvS with DPDK.

Note: Static IP address should be used for the interface of the management network.

Note: The Intel® ONP scripts do not guarantee kernel drivers are installed for network interfaces. The user must ensure kernel drivers are loaded and interfaces are present before stacking. See Table 4-2 for additional details.

5.4.3 Controller and Compute Node Installation Procedure

Follow the steps below to configure the host to be an OpenStack Controller or Compute Node:

1. Plan ahead to decide what interfaces of your hosts will belong to management and/or the data plane network.

2. Manually edit the `onps_config` configuration file on the host to be a Controller or Compute Node, hostname, type of interfaces, type of vSwitch desired vanilla vs. DPDK based Open vSwitch, etc.

3. Execute `prepare_system.sh` and reboot. The script will parse `onps_config` and prepare the system accordingly.

Note: Intel® ONP scripts assume SOCKS proxy is available. Otherwise, git will not be able to clone repositories using git:// protocol. To change the git settings, use https:// protocol instead:

```
# git config --system url."https://".insteadOf git://
```

Note: DHCP is assumed on the OpenStack outbound interface. If DHCP is not present, add DNS1 & DNS2 to `/etc/hosts`. A reboot will be needed to fix `/etc/resolv.conf`.

4. Update the system with following command:

```
# yum update -y
```

5. Reboot again.

6. Execute `prepare_stack.sh`. The script will kick off DevStack installation.

Note: If prepare_stack.sh execution run returns an error and fails the installation process, follow these instructions below and rerun the `prepare_stack.sh`:

```
$ cd /home/<username>/devstack
$ ./unstack.sh
$ ./clean.sh
$ sudo rm -rf /opt/stack/
$ rm -rf /home/<username>/devstack
$ sudo reboot
```

DPDK driver runs on a separate, dedicated CPU. ONP scripts assume CPU no. 2 should be used. The CPU used should belong to the same NUMA node as NIC used for tenant network.

You can check NUMA node of NIC by, e.g.:

```
$ cat /sys/class/net/ens6f1/device/numa_node
1
```

You can check which CPUs belong to NUMA node by, e.g.:

```
$ cat /sys/devices/system/node/node1/cpulist
```

```
12-23,36-47
```

You can change CPU used by changing OVS_PMD_CORE_MASK in local.conf generated by `prepare_system.sh`.

For example, to use CPU no. 14 set OVS_PMD_CORE_MASK=0x4000

More details in 6.2.1 OvS Core Pinning.

5.4.4 Real-Time Kernel Compute Node Enablement

Some use cases, such as Telco media applications, which are sensitive to low latency and jitter require a real time kernel.

During configuration of the `onps_config` file, the user is presented with a multiple kernel options. To deploy a Compute Node using a real time kernel, specify:

- kernel_to_use=realtime
- version=<according to the Table 4-1>
- kernelURL="https://www.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git"

To enable the Real-Time Kernel you may need to ensure the `bc` is installed, which is the component required to generate the keys to sign the kernel modules during compilation. To install the `bc` run the following command.

```
# yum install bc
```

6.0 VNF Deployment Using OpenStack

This section describes how to bring up the Virtual Network Functions (VNFs) using Virtual Machines (VMs) in the OpenStack environment, deploy various advanced features of OpenStack, utilize Enhanced Platform Awareness (EPA) Features and integrate them with the OpenDaylight (ODL) network controller

Note: It is assumed that user has successfully installed Intel® ONP software stack using instructions from 5.3 Automated Installation Using Scripts.

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 Using Horizon Graphical Interface

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 graphical interface, refer to Appendix B: 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.

6.1.2 Manual Deployment of VNFs 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.

Note: 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.

Note: 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> --visibility=public \
          --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 --visibility=public \
          -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 a 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:

```
$ 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
```

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:

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
```

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> <instance-name>
```

9. For Suricata and vBNG use cases we need to create instances (VMs) on security disabled ports to allow packets go through particular instance (VM) between two or more networks.

```
$ neutron port-create <1st network_name> --port-security-enabled=False \
          --name <port_name-1>
$ neutron port-create <2nd network_name> --port-security-enabled=False \
          --name <port_name-2>
```

10. Deploy instance (VM) on security disabled ports

```
$ nova boot --flavor=<flavor-id or flavor-name> \
          --image=<image-id or image-name> --nic port-id=$(neutron port-show \
          -f value -F id <port_name-1>) --nic port-id=$(neutron port-show \
          -f value -F id <port_name-2>) <instance-name>
```

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

11. 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.

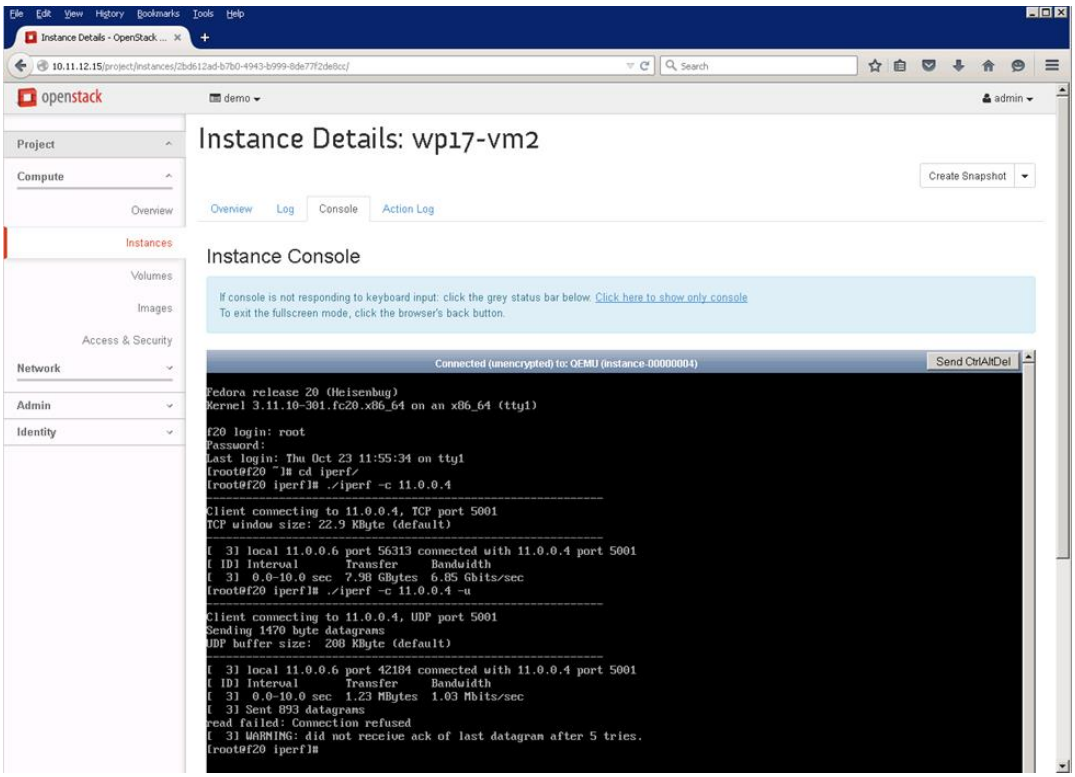

Figure 6-1 OpenStack Instance Console

6.1.3 Special Consideration for OvS-DPDK Compute Node with a vhost-user

With the OvS with 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 with DPDK compute node with a vhost-user.

6.1.4 Special Consideration for Tenant Network with DPDK and/or VXLAN Tunneling

With the introduction of DPDK and/or VXLAN tunneling to tenant network, the user must consider the maximum transmission 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 bytes. 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
```

6.1.5 Special Consideration for Two Physical Networks with VLAN on Tenant Network

The default Intel® ONP scripts configure one physical network on Tenant network with one physical port. If the host requires two physical networks on tenant network to be configured for tenant network it requires neutron networks to be created with additional parameters. This configuration is used to enable more than one VLAN so that neutron can map it a specific VLAN.

The following commands help choose the physical network on a specific VLAN for a tenant network:

```
$ neutron net-create <network name> --provider:network_type vlan --
provider:physical_network <physical network name> --provider:segmentation_id
<VLAN ID>
$ neutron subnet-create --name <subnet_name> <network-name> <net-ip-range>
```

6.2 networking-ovs-dpdk ML2 Plugin

The networking-ovs-dpdk Mechanism Layer 2 (ML2) plugin is a collection of agents and drivers to support managing DPDK accelerated Open vSwitch with Neutron. It provides a set of configurations for accelerated OvS with DPDK to take advantage of platform hardware features. Intel® ONP scripts use this plugin to help configure and expose its various features.

This plugin is loaded at runtime by Neutron service, processes all API calls and stores the resulting logical network data model and associated network mappings in a database backend. The agent associated with the plugin gathers the configuration and mappings from the central MySQL database and communicates directly with the local Open vSwitch instance to configure flows to implement the logical data model.

Intel® ONP scripts will use this plugin when the `ovsdpdk` option is enabled in the `onps_config` file. Few sample `local.conf` configuration file using this plugin can be found under `samples` directory

```
$ cd ONP_2.0/samples/
$ ls sample-local.conf-compute-ovs-dpdk*
```

6.2.1 OvS Core Pinning

The Intel® ONP scripts by default set the below options to better pin OvS processes to physical CPUs:

- **OVS_CORE_MASK**: CPU cores are selected according to this hex value to pin OvS processes. The default value used in scripts is 0x02.
- **OVS_PMD_CORE_MASK:** The mask in hex format for the Poll Mode Drivers (PMD) threads of OvS set in its database. The default value used in the scripts is 0x04.

More details about these options are detailed in Networking-OVS-DPDK ML2 Plugin documentation available at https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/usage.rst.

6.2.2 VXLAN Overlay Network Considerations

Intel® ONP scripts will deploy the VXLAN overlay network on the Compute node by choosing the `tenant_network_type` as VXLAN. The `create_local_conf.sh` script will update `local.conf` to provision and configure VXLAN, an example `local.conf` file can be found in `samples/sample-local.conf-compute-ovs-dpdk-vxlan-odl-no`.

OVS_TUNNEL_CIDR_MAPPING

Starting with OpenStack Liberty, Intel® ONP scripts use this option enables automatic assignment of the tunnel endpoint IP to a specific interface, e.g. `OVS_TUNNEL_CIDR_MAPPING=br-phy:192.168.50.1/24` assigns the IP of 192.168.50.1 with subnet mask 255.255.255.0 to the br-phy local port.

6.2.3 VLAN Overlay Network Considerations

Intel® ONP scripts will deploy the VLAN overlay network on the Compute node by choosing the `tenant_network_type` as VLAN. The `create_local_conf.sh` will update `local.conf` to provision and configure VLAN, an example `local.conf` file can be found in `samples/sample-local.conf-compute-ovs-dpdk-vlan-odl-no`.

OVS_BRIDGE_MAPPINGS

User can provide a list of comma separated pairs of "physical network:bridge name" used by DPDK/OvS while using the VLAN overlay network. Example: `OVS_BRIDGE_MAPPINGS=physnet1:br-eth0,physnet2:br-eth1`.

6.2.4 Huge Page Considerations

Virtual Machines using vhost-user need to be backed by hugepages. The following settings are used to configure huge pages:

OVS_NUM_HUGEPAGES

Specifies number of hugepages to mount, the default option used in the scripts is 8192.

OVS_HUGEPAGE_MOUNT_PAGESIZE

Specifies the preferred huge page size, as either 1GB or 2MB. 2MB huge page size is default in Intel® ONP scripts. If not set, the OS default is used. If '1G' value is used, hugepages should be allocated before starting OvS, i.e. at kernel boot command line.

6.3 Enhanced Platform Awareness

6.3.1 What is Enhanced Platform Awareness

Enhanced Platform Awareness (EPA) contributions from Intel® and others to the OpenStack cloud operating environment enable fine-grained matching of workload requirements to platform capabilities, prior to launching a VM. For workloads requiring particular CPU and/or I/O capabilities, EPA helps OpenStack assign VMs to run on the optimal platforms. EPA can benefit VM performance and operation for SDN and NFV environments. More details can be found at https://networkbuilders.intel.com/docs/OpenStack_EPA.pdf.

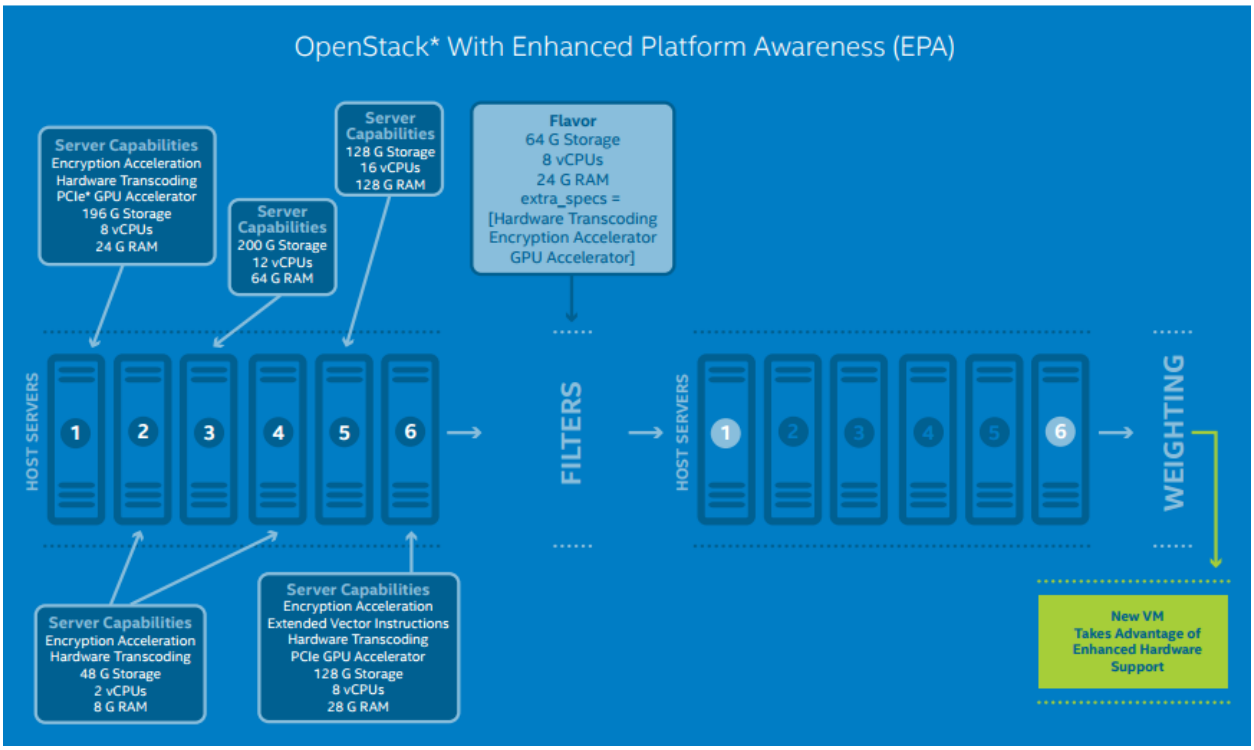

Figure 6-2 OpenStack with Enhanced Platform Awareness

There are many features provided by OpenStack to take advantage of platform features using EPA. The following set of features were validated for this release and can be used to enhance performance.

6.3.2 Dedicated Physical CPUs

Depending on the workload being executed, the end user may wish to have control over how the guest VM uses hyper-threads. To maximize cache efficiency, the guest may wish to be pinned to thread siblings. Conversely, the guest may wish to avoid thread siblings (i.e. only pin to 1 sibling) or even avoid hosts with threads entirely. This level of control is of particular importance to NFV deployments which care about maximizing cache efficiency of vCPUs.

OpenStack flavor provides extra specs below to provide this support.

```
$ nova flavor-key <flavor id> set "hw:cpu_policy=dedicated"
```

If the policy is set to `dedicated` then the guest virtual CPUs will be strictly pinned to a set of host physical CPUs.

6.3.3 Scheduler Filters

The Filter Scheduler supports filtering and weighting to make informed decisions on where a new instance should be created. Filter Scheduler iterates over all found Compute Nodes, evaluating each against a set of filters. The list of resulting hosts is ordered by weighers. The Scheduler then chooses hosts for the requested number of instances, choosing the most weighted hosts. The following filters are used in Intel® ONP scripts:

- **RamFilter** filters hosts by their RAM. Only hosts with sufficient RAM to host the instance are passed.
- **ComputeFilter** passes all hosts that are operational and enabled
- **AvailabilityZoneFilter** passes hosts matching the availability zone specified in the instance properties. Use a comma to specify multiple zones. The filter will then ensure it matches any zone specified.
- **ComputeCapabilitiesFilter** checks that the capabilities provided by the host compute service satisfy any extra specifications associated with the instance type. It passes hosts that can create the specified instance type.
- **ImagePropertiesFilter** filters hosts based on properties defined on the instance's image. It passes hosts that can support the properties specified on the image used by the instance.
- **NUMATopologyFilter** filters hosts based on the NUMA topology requested by the instance, if any.
- **PciPassthroughFilter** schedules instances on a host if the host has devices to meet the device requests in the 'extra_specs' for the flavor.

More details on scheduler filters can be found at http://docs.openstack.org/developer/nova/filter_scheduler.html.

6.4 Using OpenDaylight

6.4.1 Installing OpenDaylight

The 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.

For Controllers, The networking-odl plugin will handle the Java installation and configuration:

- First, networking-odl plugin will attempt to find supported Java version already installed.
- If a supported version is not found, it will then attempt to install from OS repository or Oracle.
- Once installed, networking-odl plugin will then configure JAVA_HOME system wide.

Note: When configuring the networking-odl with networking-ovs-dpdk, networking-odl plugin must be specified prior to newtorking-ovs-dpdk.

Note: If your infrastructure requires a proxy server to access the Internet, follow instructions in Appendix A: Configuring Proxy for OpenDaylight.

6.4.2 Additional OpenDaylight Considerations

Note: OpenDaylight configuration in OpenStack:

- If DPDK is configured, OpenDaylight will use the data plane bridge (e.g., `br-enp786f0`).
- Without DPDK, the data plane bridge is not used. Instead, OpenDaylight directly controls the bridge `br-int` for data flow.

After stacking, check and ensure the manage connections are configured

```
$ sudo ovs-vsctl show
```

Manager connections must be present and connected (connected: true) before network creation can occur, otherwise instances may not receive IP addresses. If manager connections are not present:

1. On the Controller and Compute Nodes, Intel® ONP scripts will add the manager IP address to bridge `br-int`. If not configured, add the manager IP address to bridge `br-int`. 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
   ```

2. Additionally, on the Controller check the manager IP address for bridge `br-ex`. If not configured, add the manager IP address to bridge `br-ex`. The examples below assume the management IP address of the controller is `10.11.12.35`.

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

The VXLAN port configuration will automatically be created between `br-int` for remote peers once a network subnet is created.

On the controller, the default tenant network, private network, does not pass the traffic. Users are advised to create their own tenant network in order to create a fully functional VM. Refer to the section 6.1.2 Manual Deployment of VNFs with Custom Settings on how to create a tenant network.

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

6.4.3 Monitor Network Flow with OpenDaylight

To monitor via command line, use ovs-ofctl to display the tables:

```
$ sudo ovs-ofctl -O OpenFlow13 dump-flows br-int
```

Following is the exemplary output.

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=69939.142s, table=0, n_packets=13086, n_bytes=1478718,
dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=65420.194s, table=0, n_packets=24, n_bytes=2624,
in_port=1,dl_src=fa:16:3e:cc:ea:fe actions=set_field:0x437->tun_id,load:0x1-
>NXM_NX_REG0[],goto_table:20
 cookie=0x0, duration=44107.027s, table=0, n_packets=24, n_bytes=1956,
in_port=4,dl_src=fa:16:3e:fa:83:58 actions=set_field:0x437->tun_id,load:0x1-
>NXM_NX_REG0[],goto_table:20
```

```
 cookie=0x0, duration=65420.038s, table=0, n_packets=34, n_bytes=3312,
tun_id=0x437,in_port=2 actions=load:0x2->NXM_NX_REG0[],goto_table:20
 cookie=0x0, duration=65420.214s, table=0, n_packets=0, n_bytes=0,
priority=8192,in_port=1 actions=drop
 cookie=0x0, duration=44107.023s, table=0, n_packets=0, n_bytes=0,
priority=8192,in_port=4 actions=drop
 cookie=0x0, duration=69936.354s, table=0, n_packets=11, n_bytes=954,
priority=0 actions=goto_table:20
 cookie=0x0, duration=69936.351s, table=20, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:30
 cookie=0x0, duration=69936.344s, table=30, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:40
 cookie=0x0, duration=65420.301s, table=40, n_packets=3, n_bytes=1026,
priority=61012,udp,tp_src=68,tp_dst=67 actions=goto_table:50
 cookie=0x0, duration=69936.338s, table=40, n_packets=90, n_bytes=7820,
priority=0 actions=goto_table:50
 cookie=0x0, duration=69936.332s, table=50, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:60
 cookie=0x0, duration=69936.326s, table=60, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:70
 cookie=0x0, duration=69936.321s, table=70, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:80
 cookie=0x0, duration=69936.315s, table=80, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:90
 cookie=0x0, duration=69936.310s, table=90, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:100
 cookie=0x0, duration=69936.304s, table=100, n_packets=93, n_bytes=8846,
priority=0 actions=goto_table:110
 cookie=0x0, duration=65420.200s, table=110, n_packets=14, n_bytes=1280,
tun_id=0x437,dl_dst=fa:16:3e:cc:ea:fe actions=output:1
 cookie=0x0, duration=44107.021s, table=110, n_packets=14, n_bytes=1092,
tun_id=0x437,dl_dst=fa:16:3e:fa:83:58 actions=output:4
 cookie=0x0, duration=64791.499s, table=110, n_packets=22, n_bytes=2556,
tun_id=0x437,dl_dst=fa:16:3e:a0:ba:94 actions=output:2
 cookie=0x0, duration=65420.048s, table=110, n_packets=13, n_bytes=1458,
priority=16384,reg0=0x2,tun_id=0x437,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
actions=output:1,output:4
 cookie=0x0, duration=65420.048s, table=110, n_packets=19, n_bytes=1506,
priority=16383,reg0=0x1,tun_id=0x437,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
actions=output:1,output:2,output:4
 cookie=0x0, duration=65420.043s, table=110, n_packets=0, n_bytes=0,
priority=8192,tun_id=0x437 actions=drop
 cookie=0x0, duration=69936.299s, table=110, n_packets=11, n_bytes=954,
priority=0 actions=drop
```

DLUX (OpenDaylight User eXperience) is a web-based graphical interface for ODL. After successful installation as described above, users still cannot access additional DLUX features. This is because some features are not installed by default.

1. To install these features, do the following:

```
$ cd /opt/stack/opendaylight/distribution-karaf-0.3.3-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
```

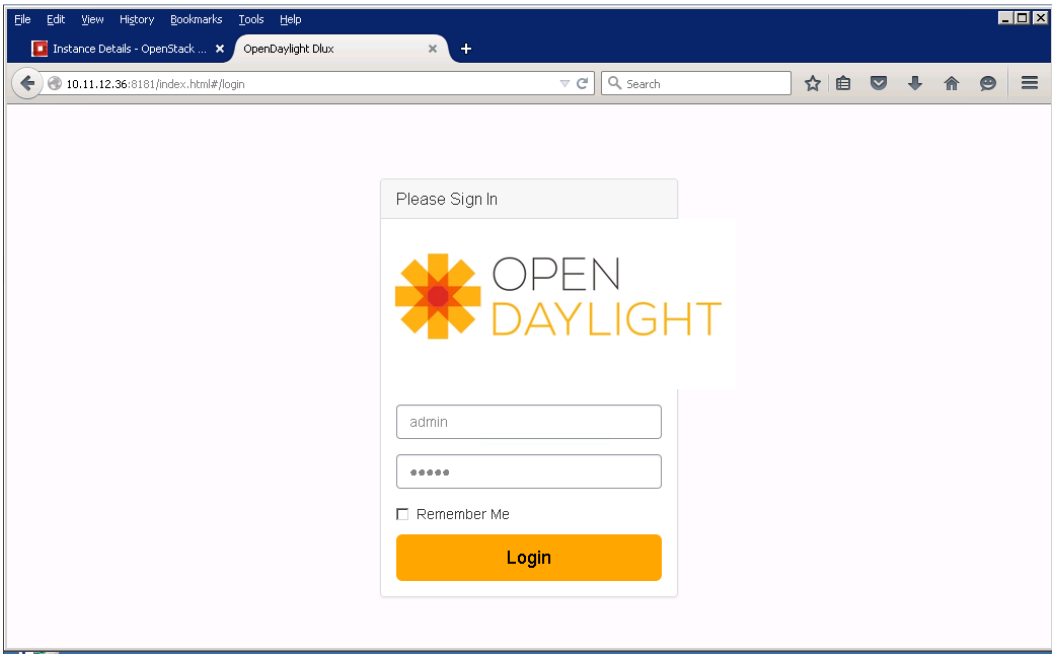

Figure 6-3 Open Daylight login page

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:245137361866694), one for br-int on compute node (openflow:196062188207951), and one for br-ex on controller (openflow:28619425664328).

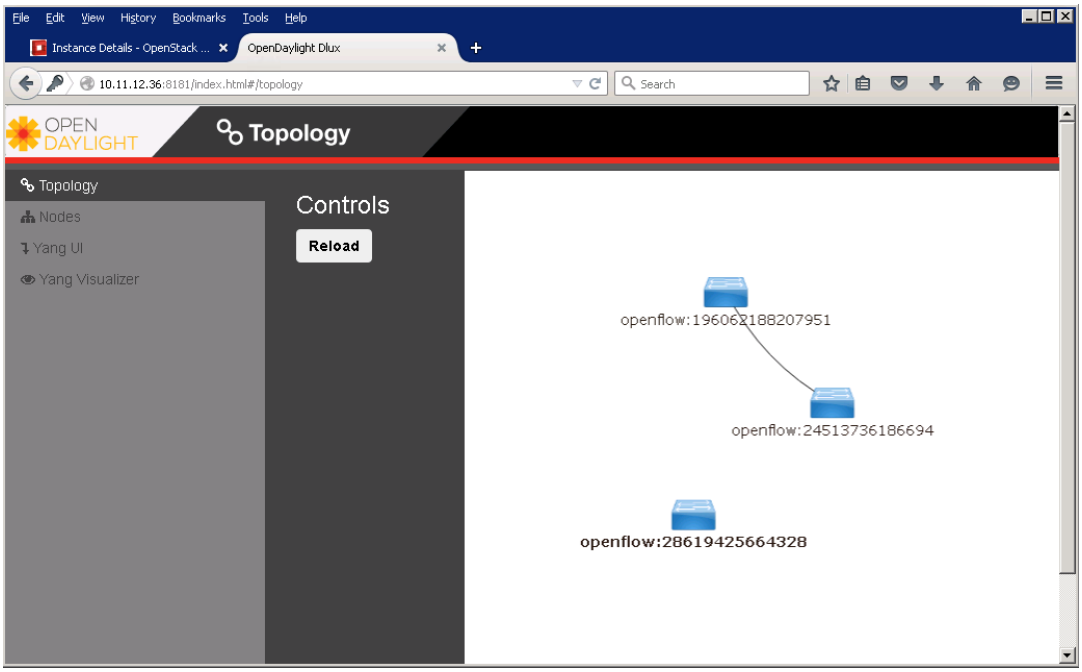

Figure 6-4 OpenDaylight Topology UI

5. Browse to “Nodes” with the following view.

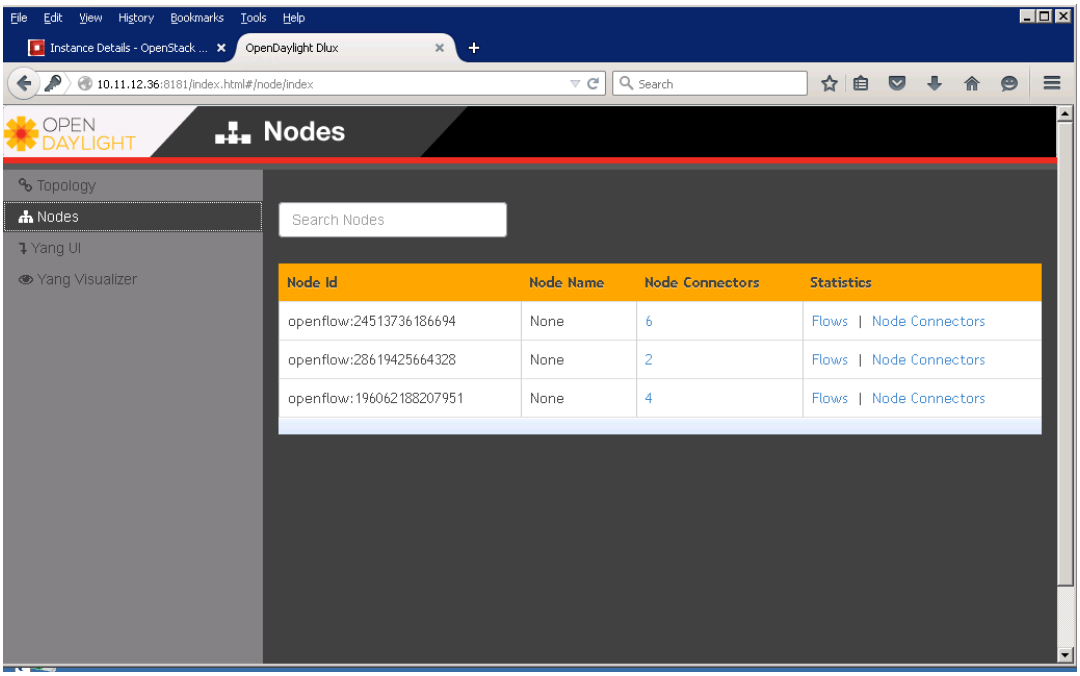

Figure 6-5 OpenDaylight Nodes UI

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.

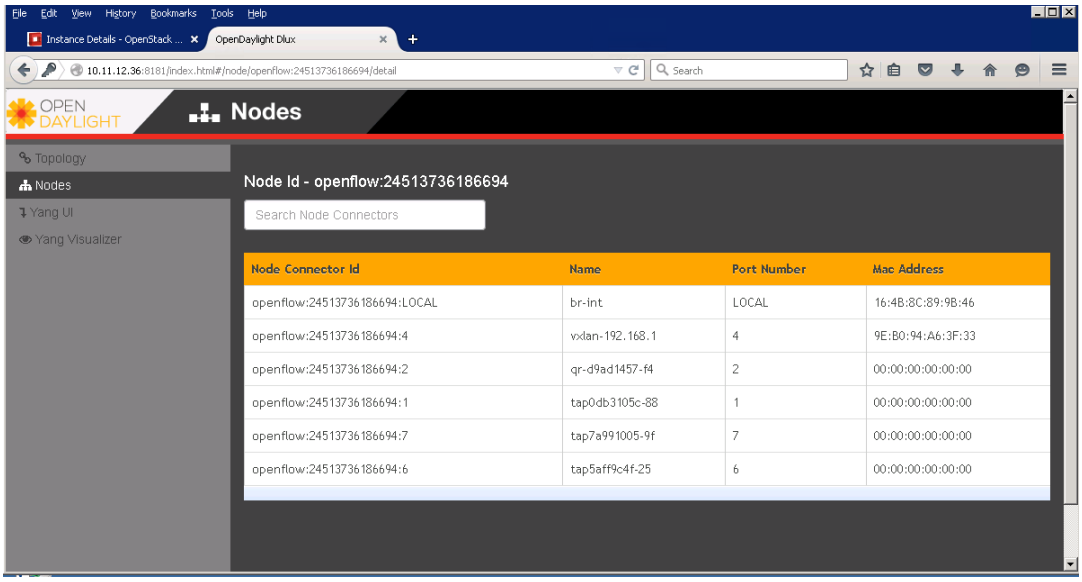

Figure 6-6 Open Daylight Nodes UI enhanced view on flows

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

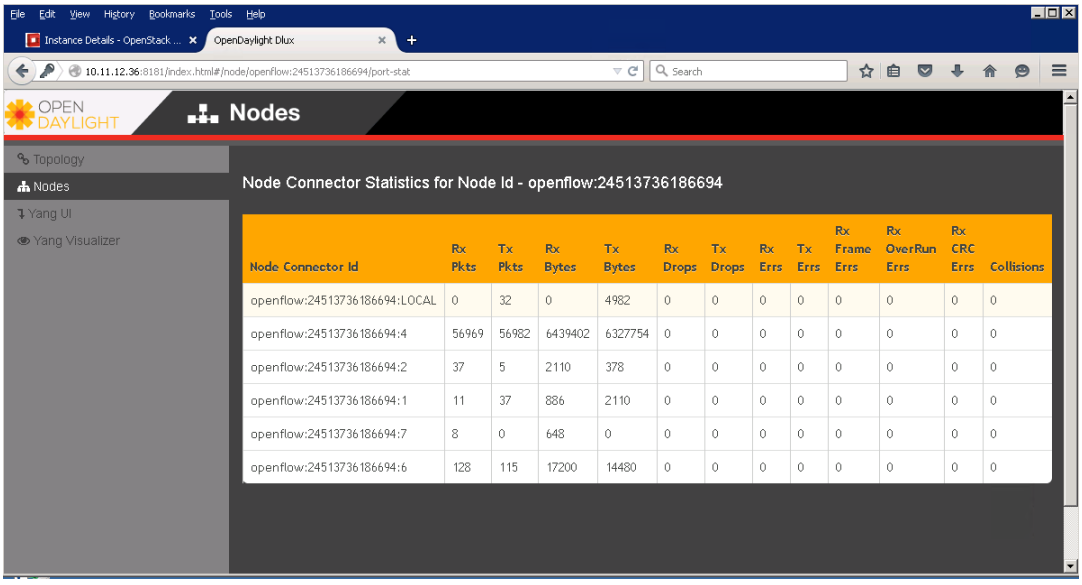

Figure 6-7 Open Daylight UI with Tx and Rx statistics of the flows

7.0 Use Cases with Virtual Network Functions

This section describes the VNFs that have been used in Intel® ONP. These functions assume VMs have been prepared in a way similar to Compute Nodes. Refer to the 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 a sink) with a VNF between them. All three are locally configured on the same compute node.

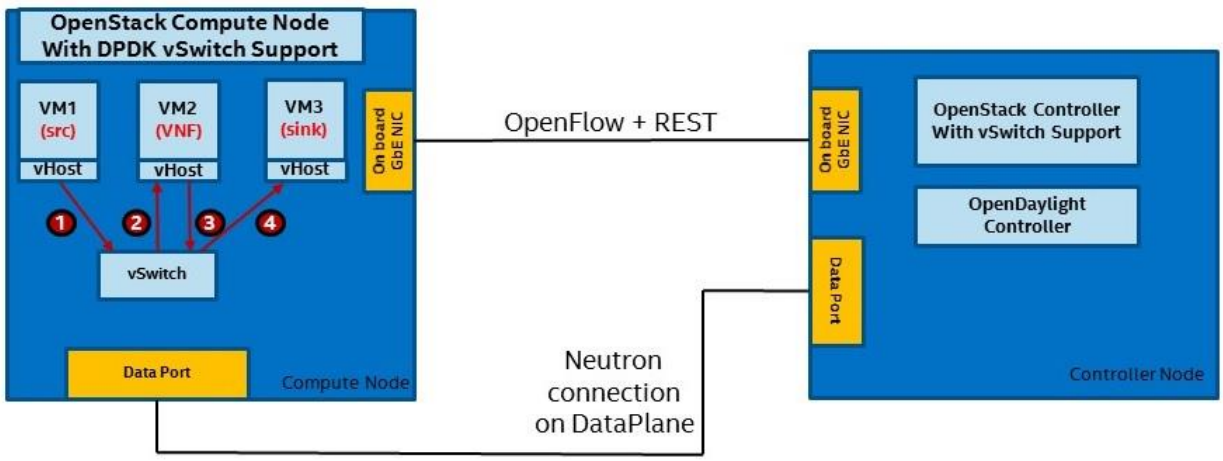

Figure 7-1 Local VNF Configuration

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 the section 6.4 Using OpenDaylight

4.).

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 (a source and a sink) with the VNF configured on a separate compute node.

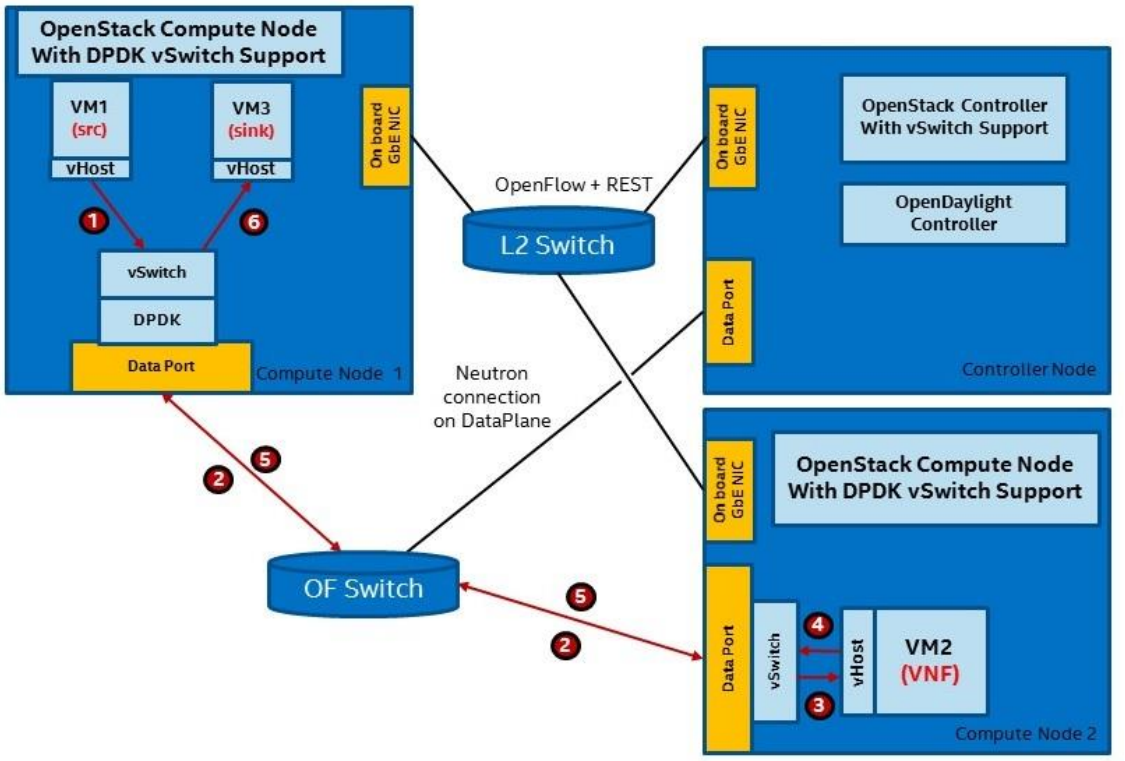

Figure 7-2 Remote VNF Configuration

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 port of Compute Node 2 into the second port of the 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 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

This section provides an example to configure a virtual Intrusion Prevention System (vIPS) VNF using Intel® ONP. With the help of OpenStack services user can choose whether the vIPS VNF should be deployed in the same compute host as the other VNFs or deploy as a standalone VNF on a remote node. The details below provide how to configure vIPS VNF either as a local VNF (7.2.2 Local vIPS Test) or as a remote VNF (7.2.2 Local vIPS Test).

7.2.1 Setup

The Controller and Compute Nodes have been set up with the following ingredients:

- DPDK (only for Compute Node)
- OvS
- OpenStack Liberty 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.

For vIPS and vBNG use cases we need to create instances (VMs) on security disabled ports to allow packets go through particular instance (VM) between two or more networks.

```
$ neutron port-create <1st network_name> --port-security-enabled=False --name
<port_name-1>
$ neutron port-create <2nd network_name> --port-security-enabled=False --name
<port_name-2>
```

Deploy instance (VM) on security disabled ports

```
$ nova boot --flavor=<flavor-id or flavor-name> --image=<image-id or image-
name> --nic port-id=$(neutron port-show -f value -F id <port_name-1>) --nic
port-id=$(neutron port-show -f value -F id <port_name-2>) <instance-name>
```

VXLAN adds an extra 50-byte header to the MTU of 1500 bytes for normal Ethernet packets. All VMs' interfaces should have MTU set to 1450 bytes, e.g.:

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

7.2.2 Local vIPS Test

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.

1. 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
```

2. From vm02:

Test ping the other VMs:

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

Turn on IP forwarding:

```
# 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
```

3. From vm01:

   Test ping vm02:

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

   Turn on IP forwarding:

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

   Test ping vm02 and vm03:

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

4. From vm03:

   Test ping vm02:

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

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

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

   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.

7. From vm03:

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

8. 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 the 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

After completing local vIPS test, delete vm02 (from the previous setup) from the Controller Node:

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

1. Return to step 8 of 6.1.2 Manual Deployment of VNFs with Custom Settings:
   - Create vm02 in a different availability zone (and, therefore, different aggregate and Compute Node) from the zone of vm01 and vm03.
   - Proceed with all steps as in the local vIPS test (refer to the section 7.2.2 Local vIPS Test).

Configuration

Note: Refer to the 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 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.

Note: vBNG is not supported with OpenDaylight.

Minimum requirements for a flavor used by vBNG are: 3 x vCPU, 4 GB memory. To proceed with installation, please follow the steps below.

1. Download kernel and components (only for Fedora VM)
   from https://kojipkgs.fedoraproject.org/packages/kernel/4.1.10/200.fc22/x86_64/

   - kernel-4.1.10-200.fc22.x86_64.rpm
   - kernel-devel-4.1.10-200.fc22.x86_64.rpm
   - kernel-modules-4.1.10-200.fc22.x86_64.rpm
   - kernel-modules-extra-4.1.10-200.fc22.x86_64.rpm
   - kernel-core-4.1.10-200.fc22.x86_64.rpm
   - kernel-headers-4.1.10-200.fc22.x86_64.rpm
2. Exclude kernel packages from update:

   ```
   # echo "exclude=kernel*" >> /etc/dnf/dnf.conf
   ```
3. Execute the following command in VM with two Virtio interfaces:

   ```
   # yum -y update
   ```
4. Disable SELINUX:

   ```
   # setenforce 0
   ```
5. Edit `/etc/selinux/config` to change the `SELinux` option.

   ```
   SELINUX=disabled
   ```
6. Disable the firewall:

   ```
   # systemctl disable firewalld.service
   # reboot
   ```
7. Edit the grub default configuration in `/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"
   ```
8. Rebuild grub config and then reboot the system:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   # reboot
   ```
9. Set the number of hugepages:

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

10. Verify that hugepages are available in the VM:

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

11. 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
# -----------------------------------------
```

12. Re-log in or source that file:

```
# . .bashrc
```

13. Install packages for build:

```
# yum -y install git @development-tools
```

14. Install DPDK:

```
# cd /root
# git clone http://dpdk.org/git/dpdk
# cd dpdk
# git checkout v2.1.0
# make install T=$RTE_TARGET
# modprobe uio
# insmod $RTE_SDK/$RTE_TARGET/kmod/igb_uio.ko
```

15. Download the BNG packages:

```
# wget https://01.org/sites/default/files/downloads/intel-data-plane-
performance-demonstrators/dppd-prox-v021.zip
```

16. Extract the DPPD BNG sources:

```
# unzip dppd-prox-v021.zip
```

17. Build the BNG DPPD application:

```
# yum -y install lua-devel libpcap-devel libedit-devel ncurses-devel
# cd dppd-PROX-v021
# make
```

18. Check the PCI addresses of the Virtio NICs:

```
# 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
```

19. 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
```

20. Create copy of config file, which will be changed:

```
# cp config/nop.cfg config/handle_none.cfg
```

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

```
# sed -i 's/s0]/]/g' config/handle_none.cfg
```

22. 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 3]
name=none
task=0
mode=none
rx port=if2
tx port=if3
drop=no

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

23. Start the application with the following command:

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

When run under OpenStack, it should look like this.

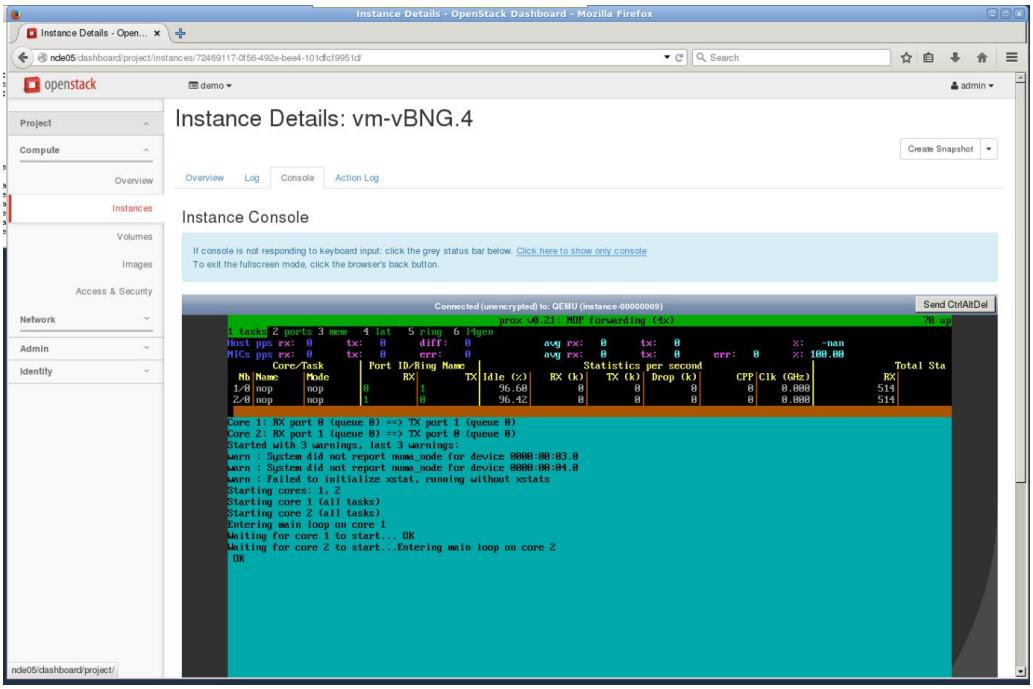

Figure 7-3 vBNG UI

To check functionality proceed with all steps as in the local vIPS test – ping, iPerf (refer to section 7.2.2 Local vIPS Test). vBNG counts packets when ping / iPerf is executed.

Appendix A: Configuring Proxy for OpenDaylight

For OpenDaylight 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 and 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 B: Configuring Horizon UI to Deploy VMs

B.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, e.g. 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."

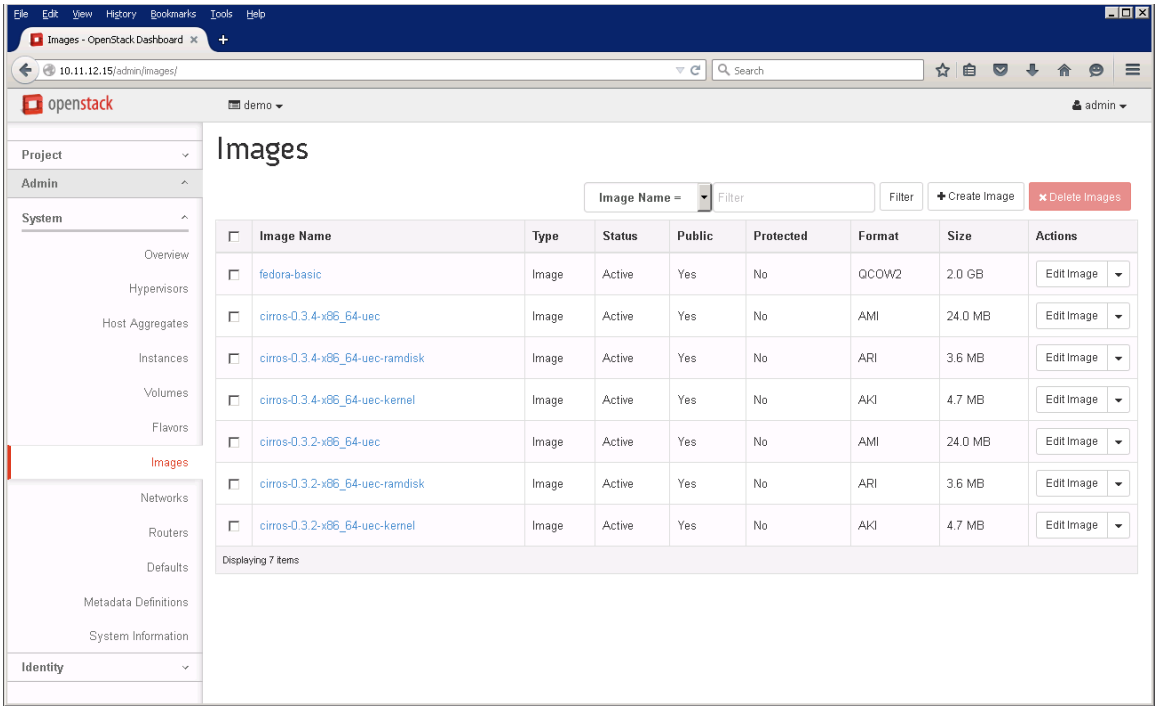

Figure B.1-1 OpenStack Images UI

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.

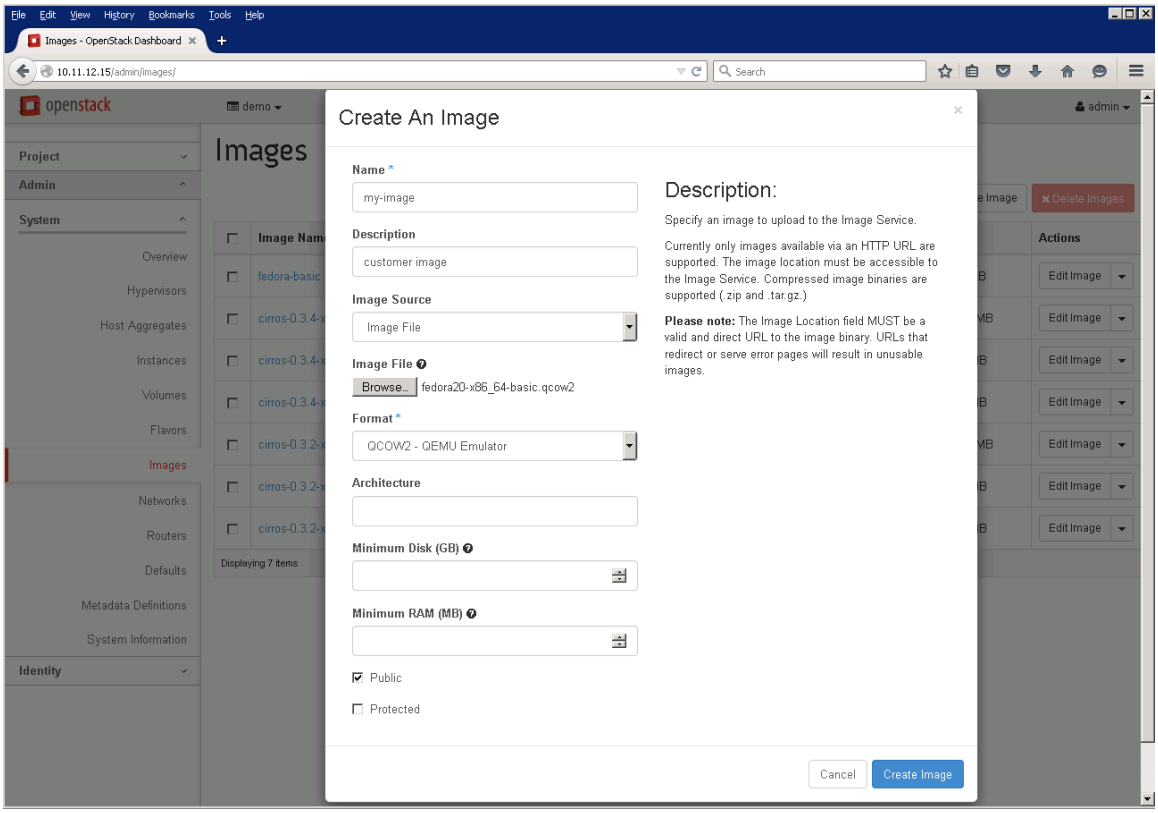

Figure B.1-2 OpenStack Images UI – Creating an Image

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.

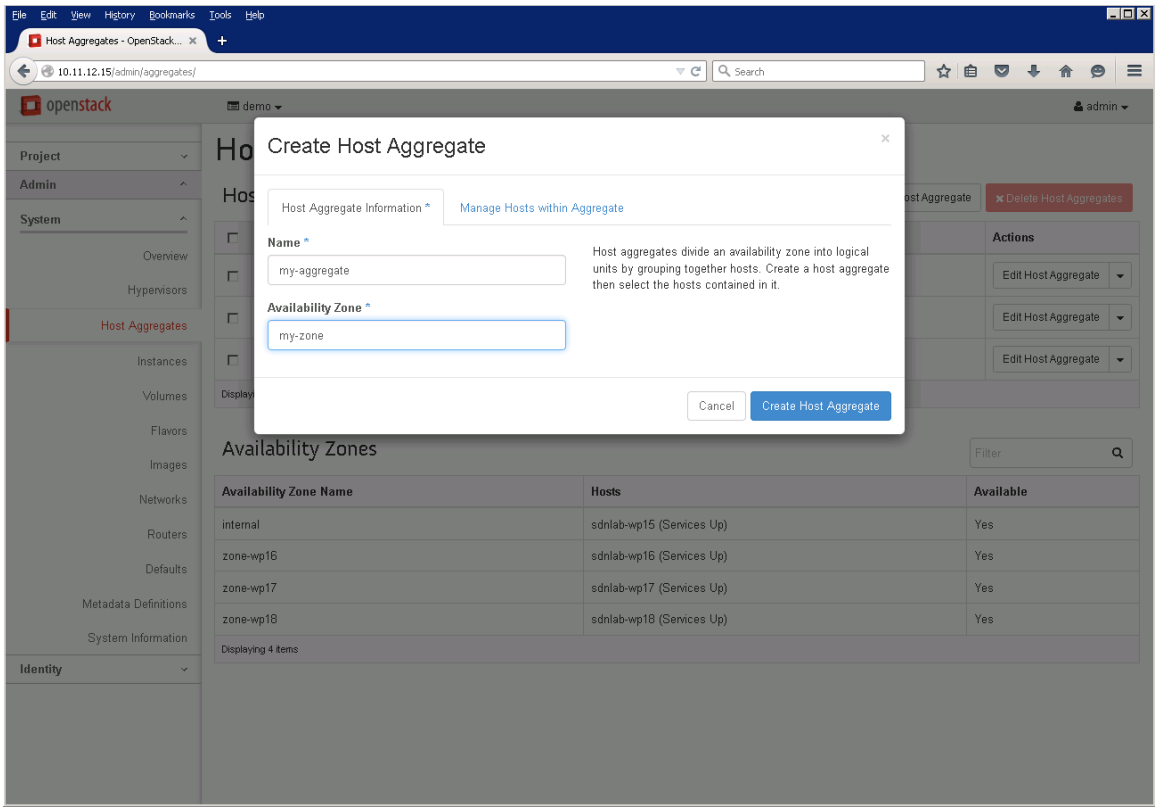

Figure B.1-3 OpenStack Host Aggregates UI – Assigning Host Aggregate Information

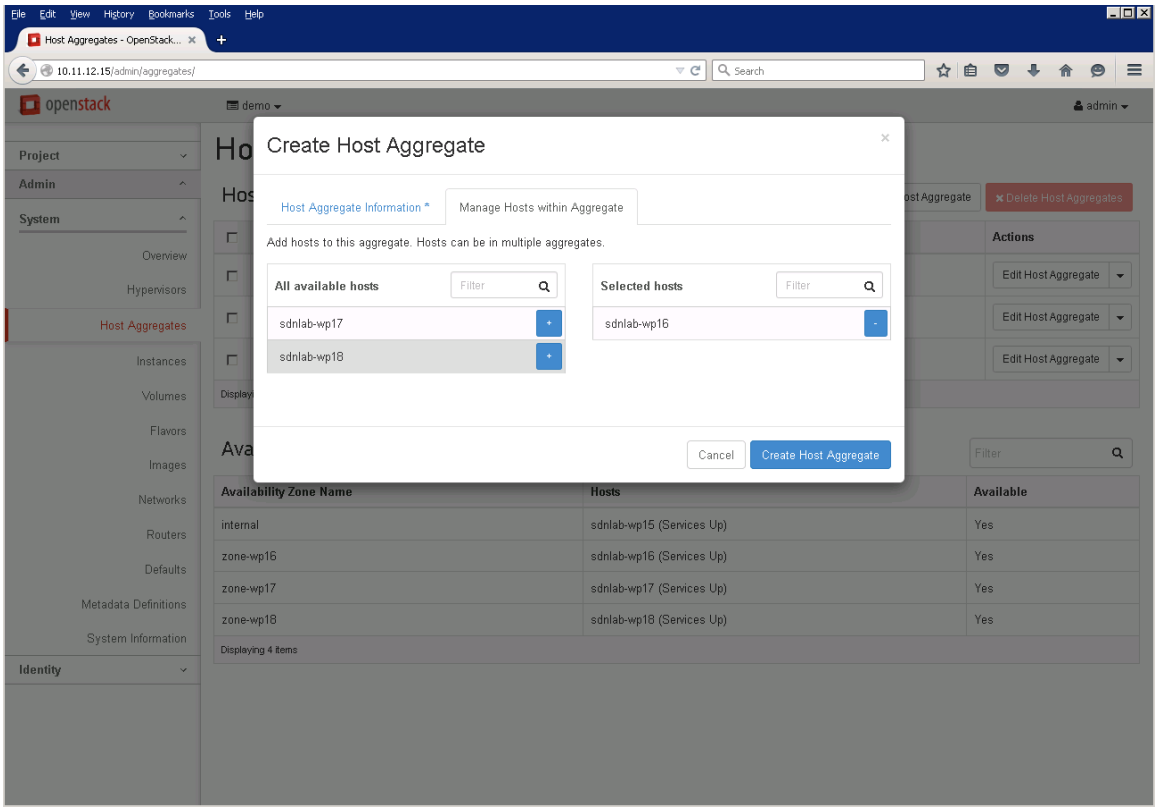

Figure B.1-4 OpenStack Host Aggregates UI – Managing Hosts within Aggregate

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.

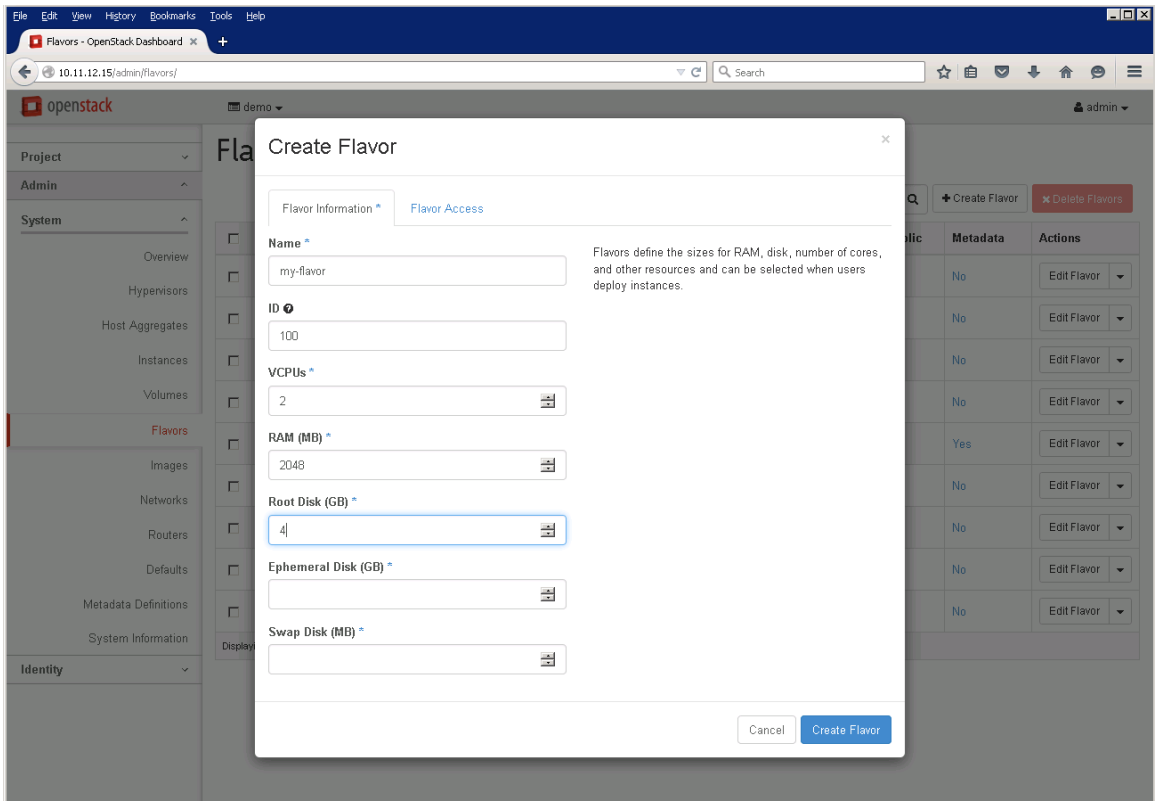

Figure B.1-5 OpenStack Flavors UI – Creating Flavor

B.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.

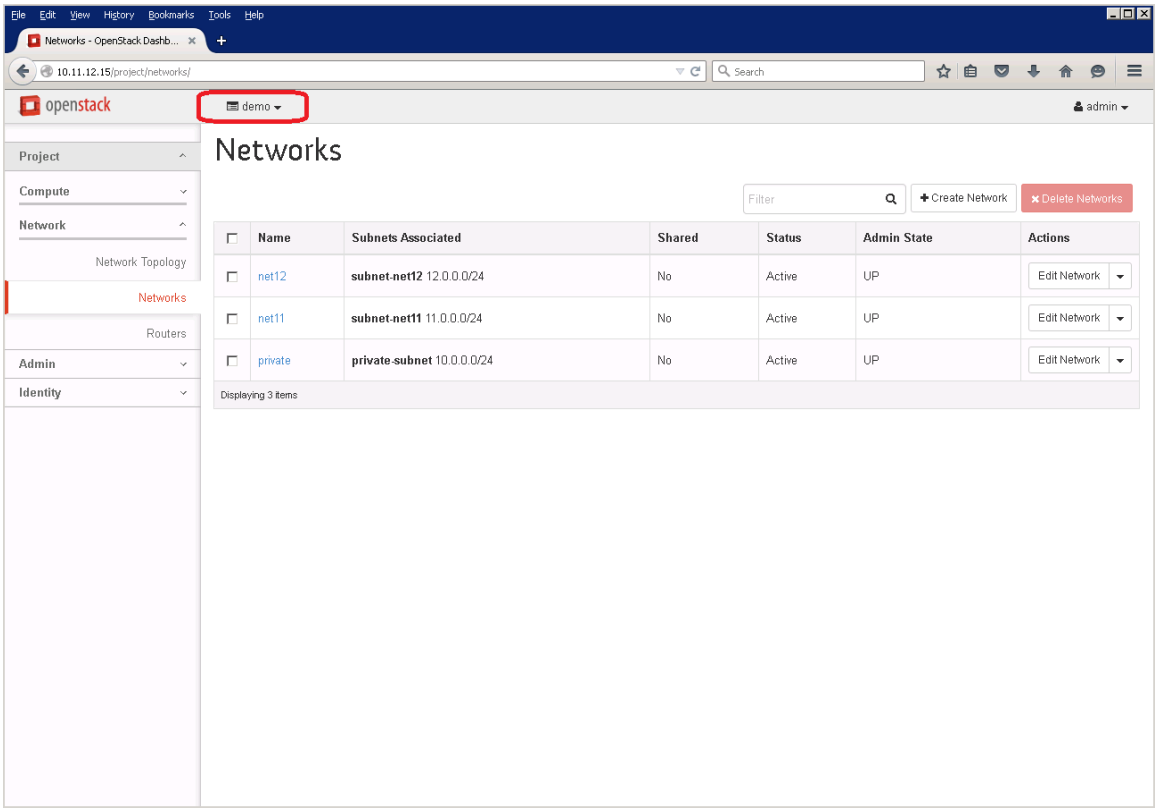

Figure B.2-1 OpenStack Networks UI

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.

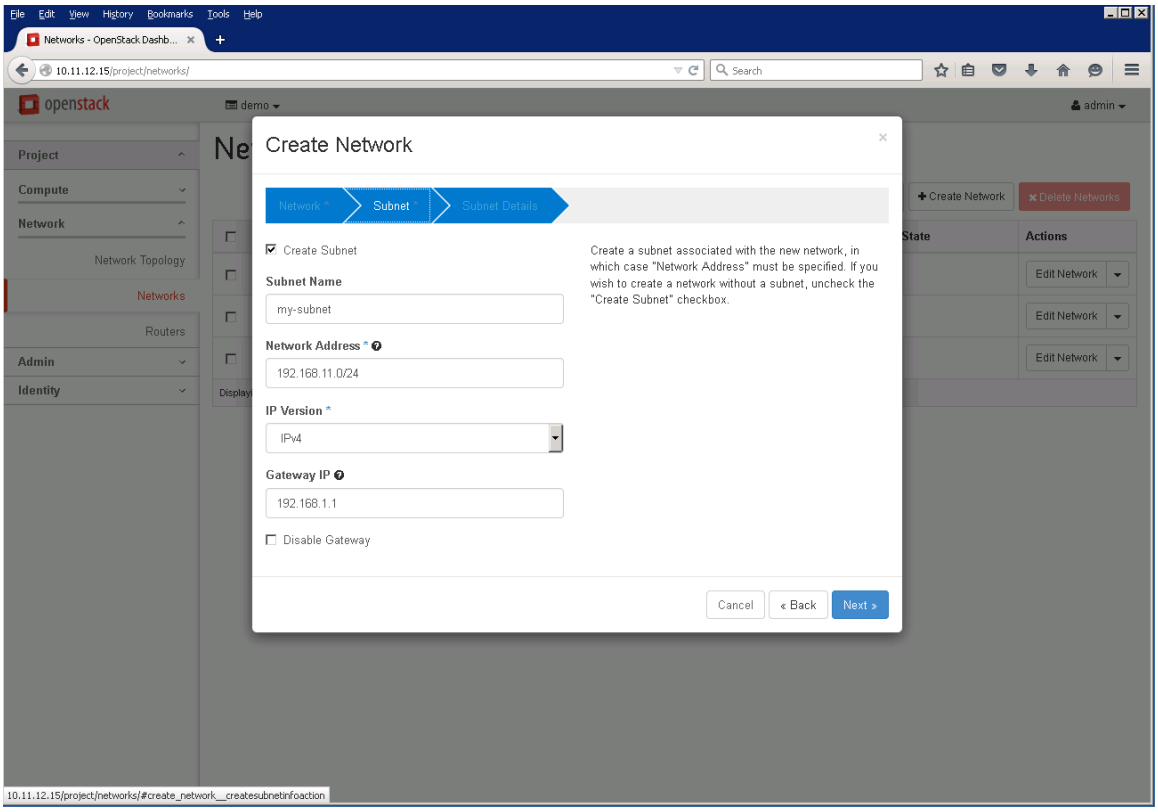

Figure B.2-2 OpenStack Networks UI – Creating Subnet

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

B.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 B.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.

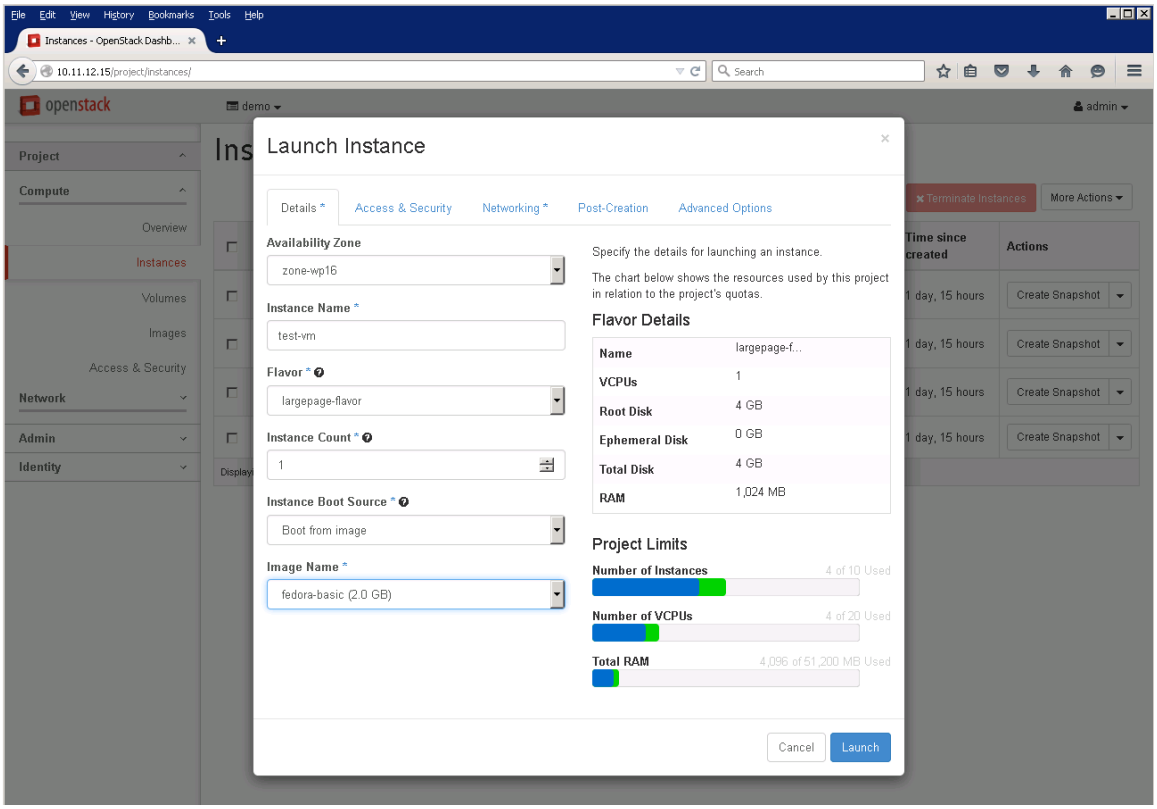

Figure B.3-1 OpenStack Instances UI – Launching Instance

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.

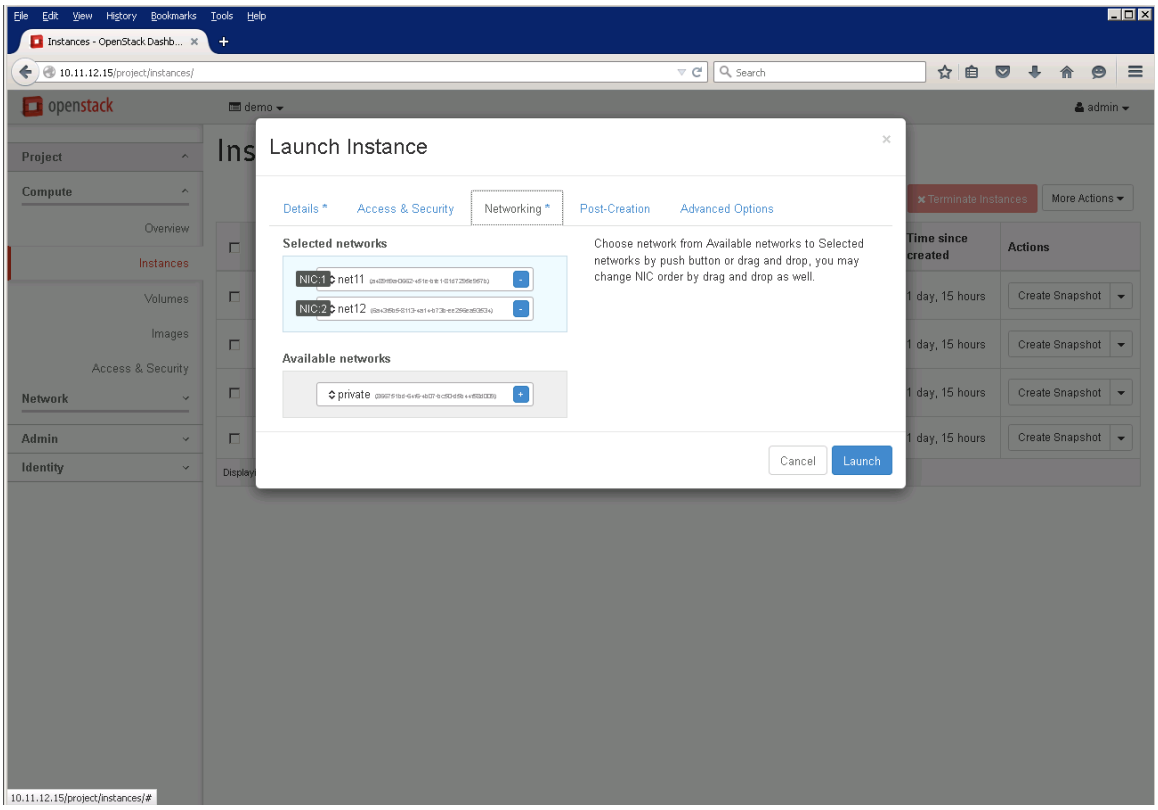

Figure B.3-2 OpenStack Instances UI – Assigning Networks to an Instance

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

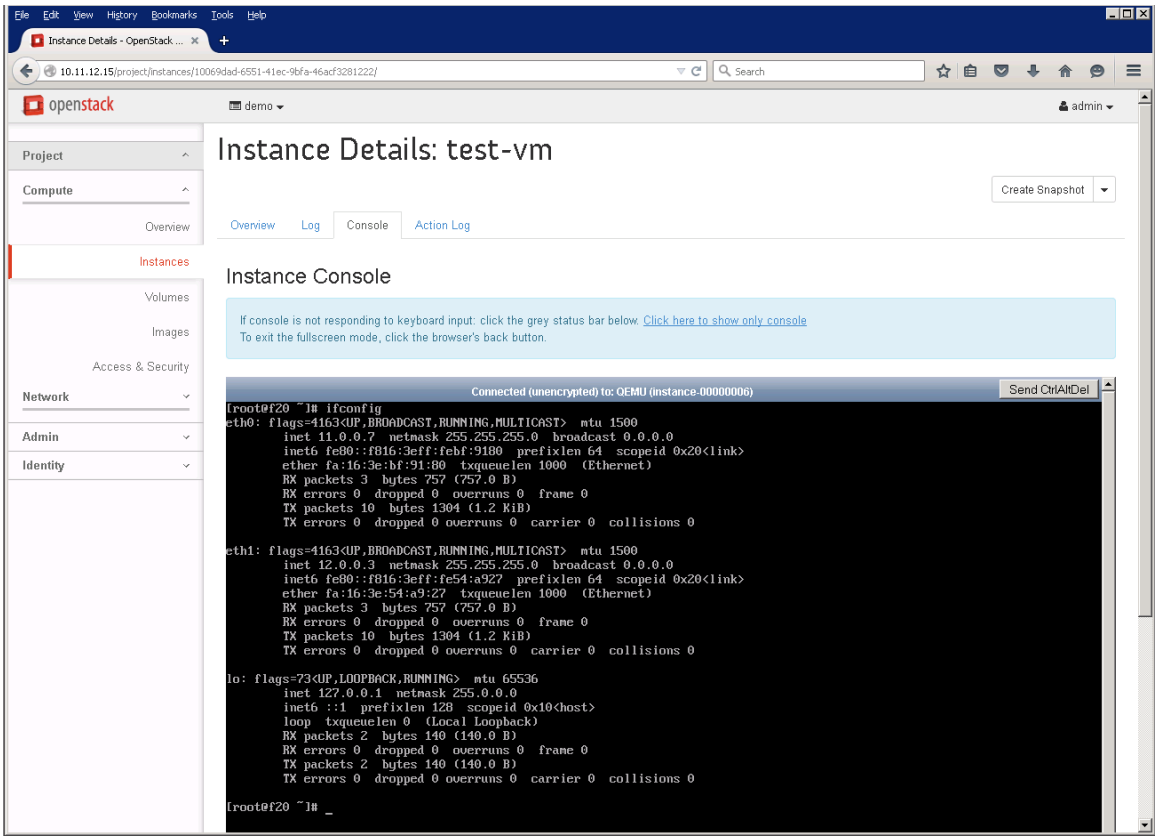

Figure B.3-3 OpenStack Instances UI – Console – Displaying Network Interfaces

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 C: Exposing External Physical Network as Virtual Public Network in OpenStack

In order to configure external network follow the instructions below. On Controller before running `prepare_system.sh`, follow as below.

1. In onps_config do not set public_net_if.

```
public_net_if=
```

2. Finish installing ONP as usually.

3. When successfully finished `prepare_stack.sh` on controller and compute(s), then add your internet interface to br-ex, e.g:

```
$ sudo ovs-vsctl add-port br-ex eno1
```

4. Remove IP from your internet interface and add the same IP to br-ex (you may lose the connection for a moment), e.g.:

```
$ sudo ip addr del 10.34.117.201/24 dev eno1; sudo ip addr add 10.34.117.201/24
dev br-ex; sudo ip route add default via 10.34.117.1
```

Note: The commands above must be sent as one-liner. Otherwise you will lose the SSH connection.

Note: If something goes wrong you can lose SSH connection. To restore the connection you may have to log in via physical terminal.

5. Now external physical network is bridged with OpenStack. You can create a public network via OpenStack dashboard. Log in to OpenStack dashboard as admin.

6. Go to Admin -> System -> Networks and create a new network:

   - Provider Network Type = flat
   - Check External network
   - Check Shared, if you want to expose it for other projects.

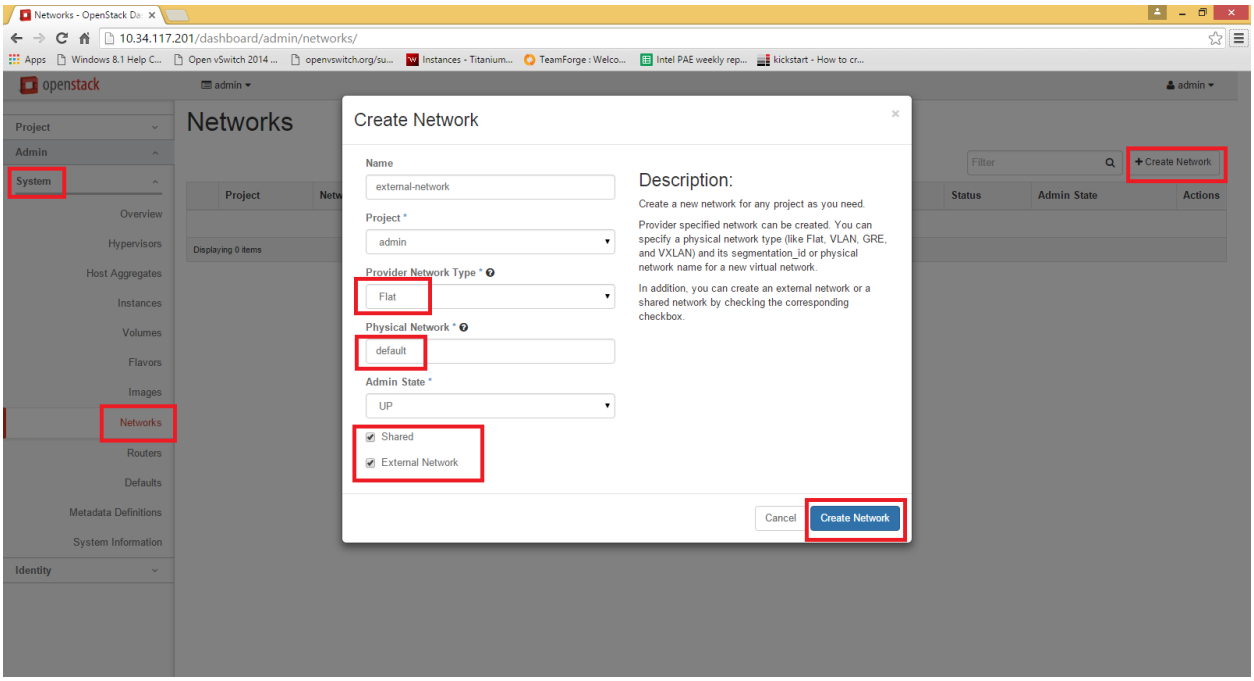

Figure C-1 OpenStack Networks UI - Creating Network

7. In the list with networks, click the name of the created network.

8. Click Create Subnet.

   - In Network Address put your physical network's CIDR.
   - Set Gateway IP to your physical gateway.

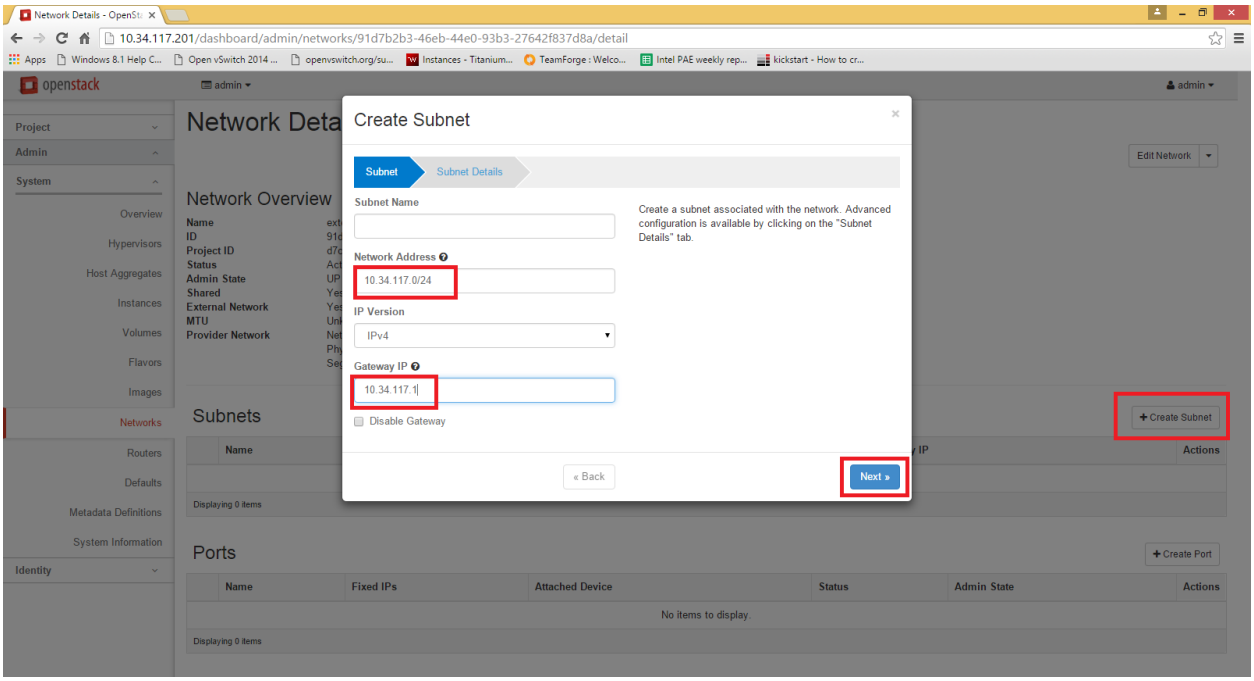

Figure C-2 OpenStack Network Details UI - Creating Subnet

9. In Subnet Details:

    - Uncheck DHCP.
    - Set Allocation Pools to a rage of IP addresses available for your cloud in your physical network.
    - Set DNS Servers

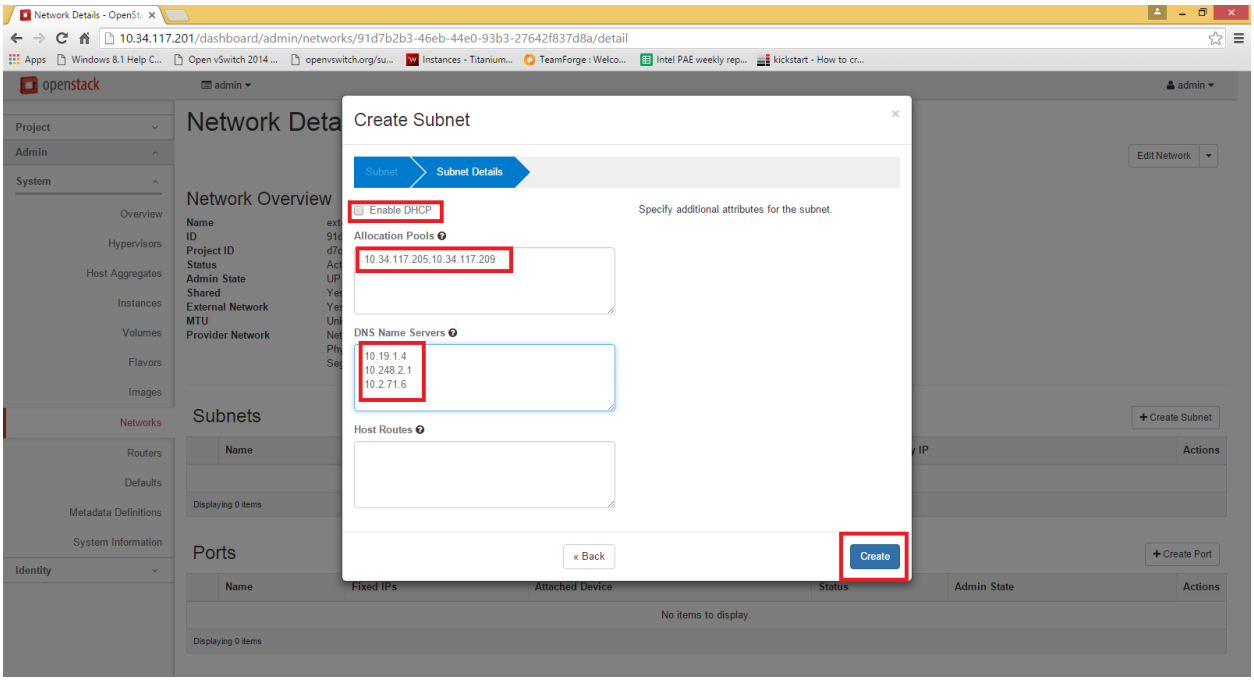

Figure C-3 OpenStack Network Details UI – Subnet Details

10. The virtual public network is now accessible from the external physical network and vice versa.

Appendix D: Exposing External and Management Physical Networks as Virtual Public Networks in OpenStack

Disclaimer:

Intel® recommends that when deploying a physical management network in your ONP topology which will be connected to a public network, that you use it behind a firewall or configure ACLs in the switch and do not expose it publically. Also it is good practice to isolate in a similar fashion your VMs from your management interfaces.

If you plan to expose only management network then the procedure is the same as described in Appendix E. Instead of Internet interface, use management interface.

If you plan to expose both Internet and Management networks, the procedure is more complicated because by default only one external network is possible.

1. In `onps_config` do not set public_net_if.

```
public_net_if=
```

2. Finish installing ONP as usually.

After successfully running `prepare_stack.sh`, perform the following steps on controller and compute(s).

In the controller node:

1. Login to the node via SSH using `stack` user.
2. Create two OvS bridges for the two networks:

```
$ sudo ovs-vsctl --no-wait -- --may-exist add-br br-ext -- set Bridge br-ext datapath_type=netdev
$ sudo ovs-vsctl --no-wait -- --may-exist add-br br-mgmt -- set Bridge br-mgmt datapath_type=netdev
```

3. Add both interfaces to the bridges:

```
$ sudo ovs-vsctl add-port br-ext eno1
$ sudo ovs-vsctl add-port br-mgmt eno2
```

4. Bring both bridges up:

```
$ sudo ip link set dev br-ext up
$ sudo ip link set dev br-mgmt up
```

5. Remove IP addresses from the interfaces and add them to the bridges:

```
$ sudo ip addr del 10.34.117.201/24 dev eno1; sudo ip addr add 10.34.117.201/24 dev br-ext; sudo ip route add default via 10.34.117.1
```

Note: The commands above must be sent as one-liner. Otherwise you will lose the SSH connection.

Note: If something goes wrong you can lose SSH connection. To restore the connection you may have to log in via physical terminal.

```
$ sudo ip addr del 192.168.0.1/24 dev eno2; sudo ip addr add 192.168.0.1/24 dev
br-mgmt
```

Note: The commands above must be sent as one-liner. Otherwise you will lose the SSH connection.

6. Edit /etc/neutron/l3_agent.ini

   Remove external network and external gateway by adding or modifying following lines accordingly:

   ```
   external_network_bridge =
   gateway_external_network_id =
   ```

7. Edit /etc/neutron/plugins/ml2/ml2_conf.ini

   Make sure flat networks are enabled by adding or modifying following lines accordingly:

   ```
   type_drivers = vxlan,vlan,flat,local
   ```

   Make sure you can name flat networks whatever you want by adding or modifying following lines accordingly:

   ```
   [ml2_type_flat]
   ...
   flat_networks = *
   ```

   Set external networks mappings to bridges by adding or modifying following lines accordingly:

   ```
   [ovs]
   ...
   bridge_mappings = physnet1:br-ens6f0,ext:br-ext,mgmt:br-mgmt
   ```

   Do not change tunnel bridge mapping.

8. Restart neutron services by running the script:

   ```
   $ /home/stack/devstack/rejoin-stack.sh
   ```

   This screen application with multiple windows open.

   You can traverse between different services by pressing Ctrl + a and then “n” for next or “p” for previous.

   Restart all services beginning with q-*

   Shut down service by pressing Ctrl + c

   Restart service by pressing up arrow (last command) end Enter.

   Close screen application by pressing Ctrl + a and then d

   The bridges have been mapped to the external networks.

9. Create external networks:

   Source credentials if neccessary:

   ```
   $ source /home/stack/devstack/openrc admin admin
   ```

   Create external network with flat type:

   ```
   $ neutron net-create external-network -- --router:external=true --
   provider:network_type=flat --provider:physical_network=ext
   ```

Create subnet for external network, disable dhcp, assign allocation pool according to your physical network policy, specify the gateway and dns nameservers:

```
$ neutron subnet-create --disable-dhcp external-network 10.34.117.0/24 --
allocation-pool start=10.34.117.205,end=10.34.117.209 --gateway 10.34.117.1 --
dns-nameserver 10.19.1.4 --dns-nameserver 10.248.2.1 --dns-nameserver 10.2.71.6
```

Create management network:

```
$ neutron net-create management-network -- --router:external=true --
provider:network_type=flat --provider:physical_network=mgmt
```

Create subnet for management network, disable dhcp, specify allocation pool and indicate that there is no gateway:

```
$ neutron subnet-create --disable-dhcp management-network 192.168.0.0/24 --
allocation-pool start=192.168.0.10,end=192.168.0.254 --no-gateway
```

Both networks are now configured and ready to be used.

Appendix E: 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
ML2	Mechanism Layer 2
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
ONP	Open Network Platform

Abbreviation	Description
OvS	Open vSwitch
PROX	Packet pROcessing eXecution engine
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 F: References

Title	Source
01.org: Intel® Open Network Platform	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 Application Note	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 Performance Test Report	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 Reference Architecture Guide	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 Release Notes	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 vE-CPE Performance Test Report	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: OpenStack Enhanced Platform Awareness	https://networkbuilders.intel.com/docs/OpenStack_EPA.pdf
DevStack	http://docs.openstack.org/developer/DevStack/
DPDK	http://www.intel.com/go/dpdk
Enhanced Platform Awareness whitepaper	http://docs.openstack.org/developer/nova/filter_scheduler.html
Intel® Atom™ processor C2000 product family for communications	http://ark.intel.com/products/series/77989/Intel-Atom-Processor-C2000-Product-Family-for-Communications
Intel® Atom™ processor C2000 product family for server	http://ark.intel.com/products/series/77975/Intel-Atom-Processor-C2000-Product-Family-for-Server
Intel® Ethernet Converged Network Adapter X540-T2	http://ark.intel.com/products/58954/Intel-Ethernet-Converged-Network-Adapter-X540-T2
Intel® Ethernet Server Adapter X520 Series	http://ark.intel.com/products/series/42489/Intel-Ethernet-Server-Adapter-X520-Series
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
Intel® Server Board S2600WTT	http://ark.intel.com/products/82156/Intel-Server-Board-S2600WTT
Intel® Xeon® processor D-1500 family	http://ark.intel.com/products/series/87040/Intel-Xeon-Processor-D-1500-Family?q=Intel%20Xeon%20D-1500#@All
Intel® Xeon® processor D-1520	http://ark.intel.com/products/87038/Intel-Xeon-Processor-D-1520-6M-Cache-2_20-GHz
Intel® Xeon® processor D-1540	http://ark.intel.com/products/87039/Intel-Xeon-Processor-D-1540-12M-Cache-2_00-GHz
Intel® Xeon® processor E5-2600 v3 product family	http://ark.intel.com/products/series/81065/Intel-Xeon-Processor-E5-2600-v3-Product-Family#@Server

Title	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
networking-ovs-dpdk ML2 Plugin	https://git.openstack.org/cgit/openstack/networking-ovs-dpdk https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/usage.rst
OpenDaylight Lithium	https://www.opendaylight.org/lithium
OpenStack	https://www.openstack.org/
SuperMicro A1SRi-2758F Motherboard	http://www.supermicro.com/products/motherboard/Atom/X10/A1SRi-2758F.cfm
SuperMicro SuperServer 5018A-FTN4	http://www.supermicro.com/products/system/1U/5018/SYS-5018A-FTN4.cfm
SuperMicro SuperServer 5018D-FN4T	http://www.supermicro.com/products/system/1u/5018/SYS-5018D-FN4T.cfm
SuperMicro X10SDV-8C-TLN4F Motherboard	http://www.supermicro.com/products/motherboard/Xeon/D/X10SDV-8C-TLN4F.cfm
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.