Red Hat OpenStack Platform 10
OpenDaylight and Red Hat OpenStack Installation and Configuration Guide

Install and Configure OpenDaylight using Red Hat OpenStack Platform

OpenStack Team

Red Hat OpenStack Platform 10 OpenDaylight and Red Hat OpenStack Installation and Configuration Guide

---

Install and Configure OpenDaylight using Red Hat OpenStack Platform

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide provides information on Red Hat OpenDaylight installation and configuration.

Table of Contents

PREFACE

This document describes how to deploy Red Hat OpenStack Platform 10 to use the OpenDaylight software-defined network (SDN) controller. The OpenDaylight controller is used as a drop-in replacement for the neutron **ML2/OVS** plug-in and its **L2** and **L3** agents, and provides network virtualization within the Red Hat OpenStack environment.

Note

The OpenDaylight solution provided with Red Hat OpenStack Platform 10 is available as technology preview. For more information on the support scope for features marked as technology previews, see Technology Preview Features Support Scope.

CHAPTER 1. OVERVIEW

1.1. WHAT IS OPENDAYLIGHT?

The OpenDaylight platform is a Java-based programmable SDN controller that can be used for network virtualization for OpenStack environments. The controller architecture consists of separated northbound and southbound interfaces. For OpenStack integration purposes, the main northbound interface uses the NeutronNorthbound project, which communicates with neutron, the OpenStack Networking service. The southbound OpenDaylight projects, the **OVSDB** and the **OpenFlow** plug-ins, are used to communicate with the **Open vSwitch** (OVS) control and the data plane. The main OpenDaylight project that translates the neutron configuration into network virtualization is the NetVirt project.

1.2. HOW DOES OPENDAYLIGHT WORK WITH OPENSTACK?

1.2.1. The default neutron architecture

The neutron reference architecture uses a series of agents to manage networks within OpenStack. These agents are provided to neutron as different plug-ins. The core plug-ins are used to manage the *Layer 2* overlay technologies and data plane types. The service plug-ins are used to manage network operations for *Layer 3* or higher in the OSI model, such as firewall, DHCP, routing and NAT.

By default, Red Hat OpenStack Platform uses the Modular Layer 2 (**ML2**) core plug-in with the OVS mechanism driver, that provides an agent to configure OVS on each Compute and Controller node. The service plug-ins, the DHCP agent, the metadata agent, along with the **L3** agent, run on controllers.

1.2.2. Networking architecture based on OpenDaylight

OpenDaylight integrates with the **ML2** core plug-in by providing its own driver called **networking-odl**. This eliminates the necessity to use the OVS agent on every node. OpenDaylight is able to program each OVS instance across the environment directly, without any per-node agents. For *Layer 3* services, neutron is configured to use the OpenDaylight **L3** plug-in. This approach reduces the number of agents on multiple nodes that handle routing and network address translation (NAT), because OpenDaylight can handle the distributed virtual routing functionality by programming the data plane directly. The neutron DHCP and metadata agents are still used for managing DHCP and metadata (cloud-init) requests.

Note

OpenDaylight is able to provide DHCP services. However, when deploying the current Red Hat OpenStack Platform director architecture, using the neutron DHCP agent provides High Availability (HA) and support for the VM instance metadata (**cloud-init**), and therefore Red Hat recommends you deploy the neutron DHCP agent rather than rely on OpenDaylight for such functionality.

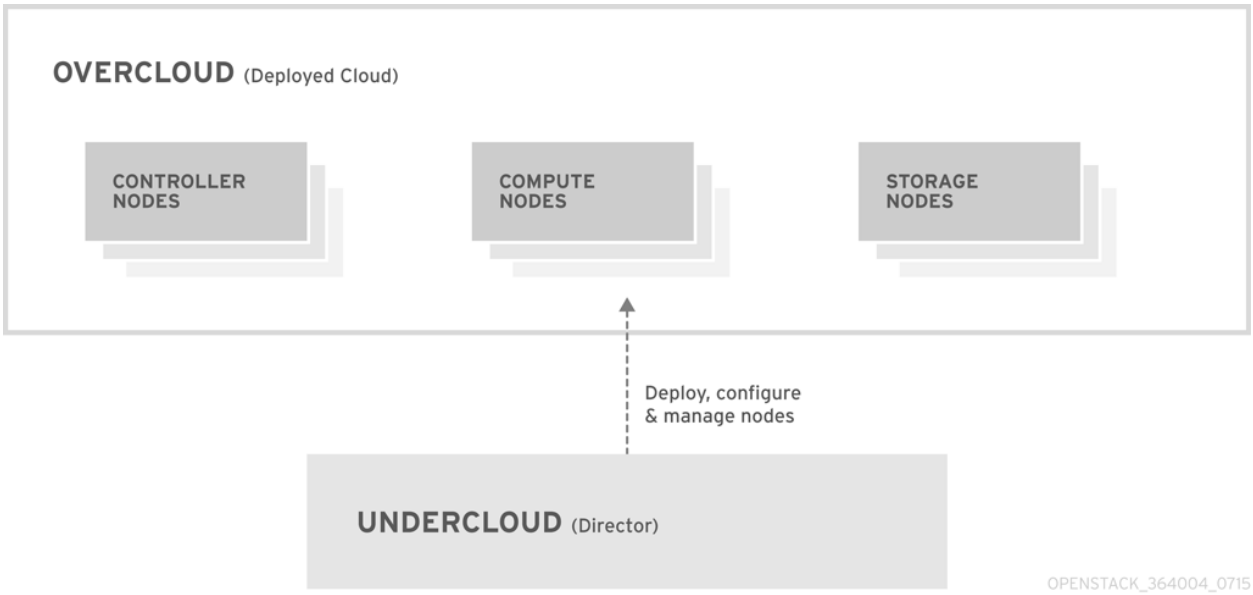
1.3. WHAT IS RED HAT OPENSTACK PLATFORM DIRECTOR AND HOW IS IT DESIGNED?

The Red Hat OpenStack Platform director is a toolset for installing and managing a complete OpenStack environment. It is primarily based on the OpenStack TripleO (OpenStack-On-OpenStack) program.

The project uses OpenStack components to install a fully operational OpenStack environment. It also includes new OpenStack components that provision and control bare metal systems to work as OpenStack nodes. With this approach, you can install a complete Red Hat OpenStack Platform environment, that is both lean and robust.

The Red Hat OpenStack Platform director uses two main concepts: an *undercloud* and an *overcloud*. The undercloud installs and configures the overcloud. For more information about the Red Hat OpenStack Platform director architecture, see the Director Installation and Usage guide.

Figure 1.1. OpenStack Platform Director — undercloud and overcloud



1.3.1. Red Hat OpenStack Platform director and OpenDaylight

Red Hat OpenStack Platform director 10 introduces the new feature of composable services and custom roles. They form isolated resources, that can be included and enabled per role, when they are needed. Custom roles enable users to create their own roles, independent from the default controller and Compute roles. In other words, users now have the option to choose which OpenStack services they will deploy, and which node will host them.

Two new services have been added for OpenDaylight integration:

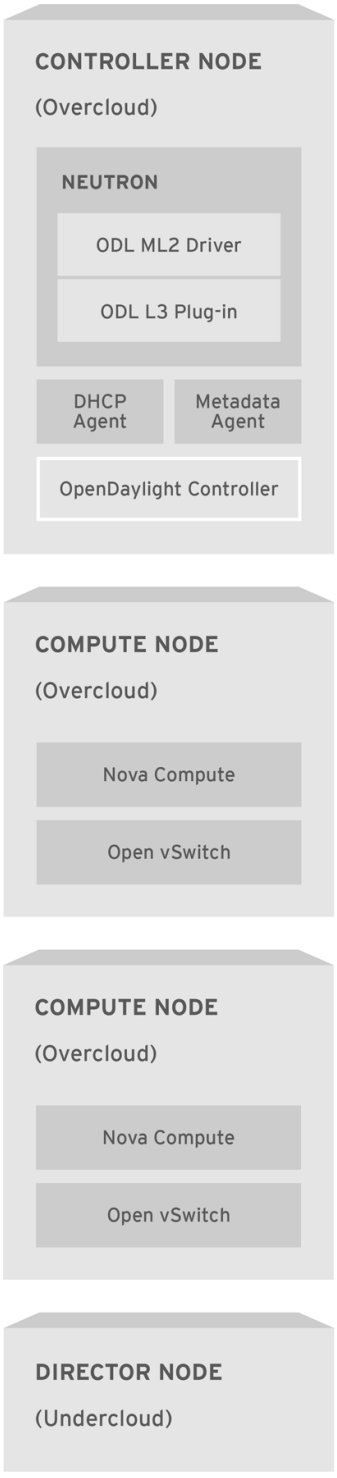
- The **OpenDaylightApi** service for running the OpenDaylight *SDN* controller, and
- The **OpenDaylightOvs** service for configuring OVS on each node to properly communicate with OpenDaylight.

By default, the **OpenDaylightApi** service is configured to run on the controller role, while the **OpenDaylightOvs** service is configured to run on controller and Compute roles.

Note

In Red Hat OpenStack Platform 10 only a single instance of OpenDaylight is supported. In a deployment with multiple overcloud controllers, the **OpenDaylightApi** service will be applied to each controller role, however only the first controller will actually enable OpenDaylight.

Figure 1.2. OpenDaylight and OpenStack — base architecture



1.3.2. Network isolation in Red Hat OpenStack Platform director

Red Hat OpenStack Platform director is capable of configuring individual services to specific, predefined network types. These network traffic types include:

IPMI	The network used for the power management of nodes. This network must be set up before the installation of the undercloud.
Provisioning (ctlplane)	The director uses this network traffic type to deploy new nodes over the *DHCP* and *PXE* boot and orchestrates the installation of OpenStack Platform on the overcloud bare metal servers. The network must be set up before the installation of the undercloud. Alternatively, operating system images can be deployed directly by ironic. In that case, the *PXE* boot is not necessary.
Internal API (internal_api)	The *Internal API* network is used for communication between the OpenStack services using API communication, RPC messages, and database communication, as well as for internal communication behind the load balancer.
Tenant (tenant)	neutron provides each tenant with their own networks using either *VLANs* (where each tenant network is a network *VLAN*), or overlay tunnels. Network traffic is isolated within each tenant network. If tunneling is used, multiple tenant networks can use the same IP address range without any conflicts.

Note

While both Generic Routing Encapsulation (GRE) and Virtual eXtensible Local Area Network (VXLAN) are available in the codebase, *VXLAN* is the recommended tunneling protocol to use with OpenDaylight. *VXLAN* is defined in RFC 7348. The rest of this document is focused on *VXLAN* whenever tunneling is used.

Storage (storage)	Block Storage, NFS, iSCSI, and others. Ideally, this would be isolated to an entirely separate switch fabric for performance reasons.
Storage Management (storage_mgmt)	OpenStack Object Storage (swift) uses this network to synchronize data objects between participating the replica nodes. The proxy service acts as an intermediary interface between user requests and the underlying storage layer. The proxy receives incoming requests and locates the necessary replica to retrieve the requested data. Services that use a *Ceph* backend connect over the Storage Management Network, since they do not interact with *Ceph* directly but rather use the frontend service. Note that the **RBD** driver is an exception, as this traffic connects directly to *Ceph*.

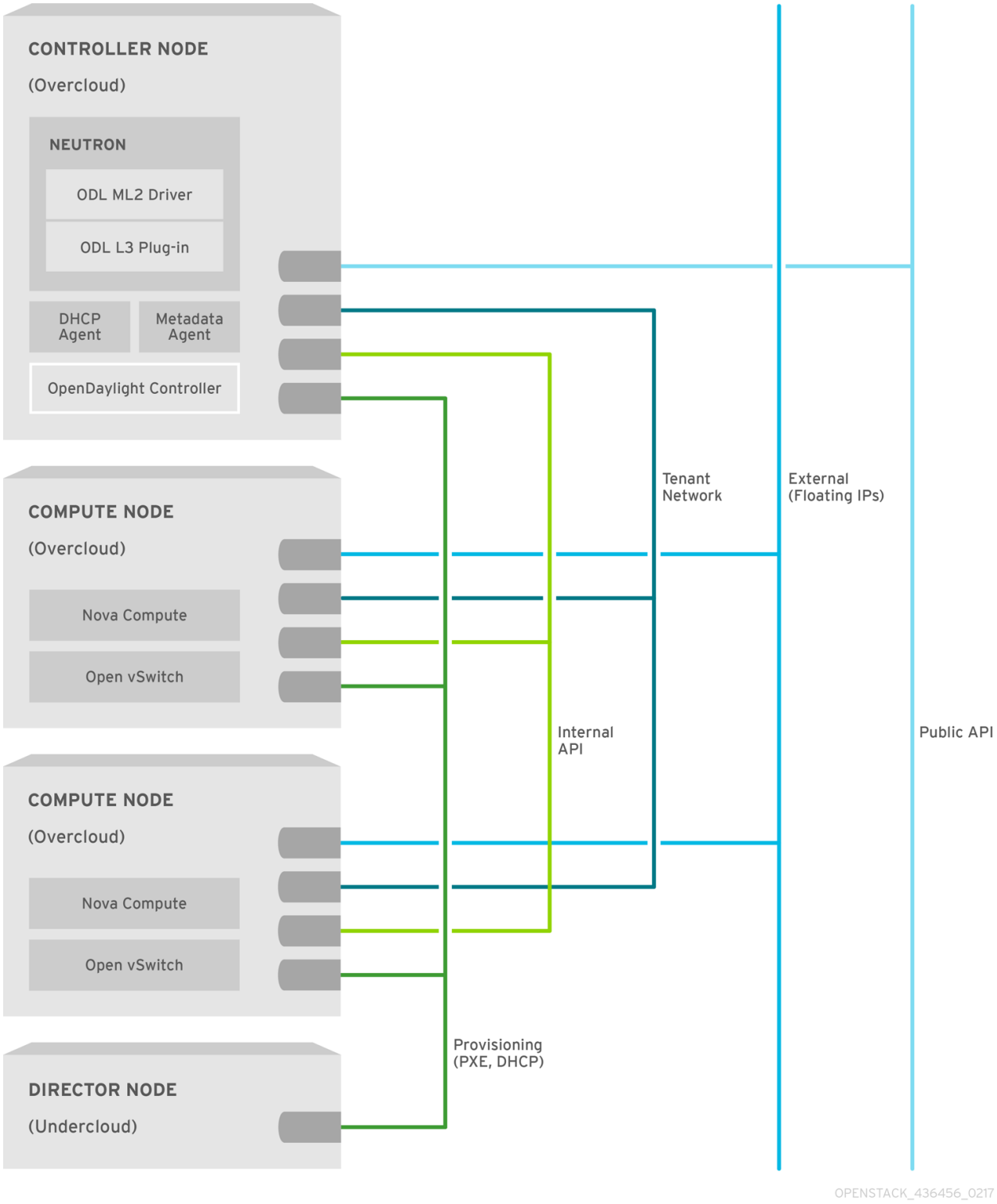
External/Public API	This API hosts the OpenStack Dashboard (horizon) for graphical system management, the public APIs for OpenStack services, and performs SNAT for incoming traffic going to the instances. If the external network uses private IP addresses (as per RFC-1918), then further NAT must be performed for any traffic coming in from the internet.
Floating IPs	Allows incoming traffic to reach instances using 1-to-1 *IPv4* address mapping between the floating IP address and the fixed IP address, assigned to the instance in the tenant network. A common configuration is to combine the external and the floating IPs network instead of maintaining a separate one.
Management	Provides access for system administration functions such as SSH access, DNS traffic, and NTP traffic. This network also acts as a gateway for non-controller nodes.

In a typical Red Hat OpenStack Platform installation, the number of network types often exceeds the number of physical network links. In order to connect all the networks to the proper hosts, the overcloud may use the 802.1q *VLAN* tagging to deliver more than one network per interface. Most of the networks are isolated subnets but some require a *Layer 3* gateway to provide routing for Internet access or infrastructure network connectivity.

For OpenDaylight, the relevant networks include *Internal API*, *Tenant*, and *External* services, that are mapped to each network inside of the **ServiceNetMap**. By default, the **ServiceNetMap** maps the **OpenDaylightApi** network to the *Internal API* network. This configuration means that northbound traffic to neutron as well as southbound traffic to **OVS** are isolated to the *Internal API* network.

As OpenDaylight uses a distributed routing architecture, each Compute node should be connected to the *Floating IP* network. By default, Red Hat OpenStack Platform director assumes that the *External* network will run on the physical neutron network *data centre*, which is mapped to the OVS bridge *br-ex*. Therefore, you must include the *br-ex* bridge in the default configuration of the Compute node NIC templates.

Figure 1.3. OpenDaylight and OpenStack — Network isolation example



1.3.3. Network and firewall configuration

On some deployments, such as those where restrictive firewalls are in place, you might need to configure the firewall manually in order to enable OpenStack and OpenDaylight service traffic.

By default, OpenDaylight Northbound uses the *8080* and *8181* ports. In order not to conflict with the swift service, that also uses the *8080* port, the OpenDaylight ports are set to *8081* and *8181* when installed with Red Hat OpenStack Platform director. The Southbound, in Red Hat OpenDaylight solution, is configured to listen on ports *6640* and *6653*, that the **OVS** instances usually connect to.

Warning

The default OpenDaylight southbound port *6633* is also open and available. If you do not want to use that port, you can consider closing it, in order to prevent security issues.

In OpenStack, each service typically has its own virtual IP address (VIP) and OpenDaylight behaves the same way. **HAProxy** is configured to open the *8081* port to the public and control the plane's VIPs that are already present in OpenStack. The VIP and the port are presented to the **ML2** plug-in and neutron sends all communication through it. The **OVS** instances connect directly to the physical IP of the node where OpenDaylight is running for Southbound.

Service	Protocol	Default Ports	Network
OpenStack Neutron API	TCP	9696	Internal API
OpenStack Neutron API (SSL)	TCP	13696	Internal API
OpenDaylight Northbound	TCP	8081, 8181	Internal API
OpenDaylight Southbound: OVSDB	TCP	6640	Internal API
OpenDaylight Southbound: OpenFlow	TCP	6653	Internal API
VXLAN	UDP	4789	Tenant

Table 1: Network and Firewall configuration

Note

The above section focuses on the services and protocols relevant to the OpenDaylight integration and is not exhaustive. For a complete list of network ports required for services running on Red Hat OpenStack, see the Configure Firewall Rules for Red Hat OpenStack Platform Director guide.

CHAPTER 2. PREPARE FOR OPENDAYLIGHT INSTALLATION

The following section lists the steps needed to deploy the overcloud with OpenDaylight.

This document only focuses on OpenDaylight installation. Before you can deploy OpenDaylight, you must make sure that you have a working undercloud environment and that the overcloud nodes are connected to the physical network.

See Installing the Undercloud and Configuring Basic Overcloud Requirements with the CLI Tools of the *Director Installation and Usage* guide, which describes the necessary procedures to deploy the undercloud and overcloud.

2.1. WHAT ARE THE MINIMUM HARDWARE REQUIREMENTS?

To correctly install and run Red Hat OpenDaylight, you should have enough computer resources. The following are the minimum requirements.

2.1.1. Compute Node Requirements

Compute nodes are responsible for running virtual machine instances after they are launched. All Compute nodes must support hardware virtualization. They also must have enough memory and disk space to support the requirements of the virtual machine instances they host.

Processor	64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions, and the AMD-V or Intel VT hardware virtualization extensions enabled. It is recommended this processor has a minimum of 4 cores.
Memory	A minimum of 6 GB of RAM. Add additional RAM to this requirement based on the amount of memory that you intend to make available to virtual machine instances.
Disk Space	A minimum of 40 GB of available disk space.
Network Interface Cards	A minimum of one 1 Gbps Network Interface Cards, although it is recommended to use at least two NICs in a production environment. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic.
Power Management	Each Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server's motherboard.

2.1.2. Controller Node Requirements

Controller nodes are responsible for hosting the core services in a Red Hat OpenStack Platform environment, such as the horizon dashboard, the back-end database server, keystone authentication, and High Availability services.

Processor	64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.
Memory	Minimum amount of memory is 20 GB. However, the amount of recommended memory depends on the number of CPU cores. Use the following calculations as guidance: **Controller RAM minimum calculation:** Use 1.5 GB of memory per core. For example, a machine with 48 cores should have 72 GB of RAM. **Controller RAM recommended calculation:** Use 3 GB of memory per core. For example, a machine with 48 cores should have 144 GB of RAM. For more information on measuring memory requirements, see Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers on the Red Hat Customer Portal.
Disk Space	A minimum of 40 GB of available disk space.
Network Interface Cards	A minimum of 2 x 1 Gbps Network Interface Cards. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic.
Power Management	Each Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (*IPMI*) functionality, on the server's motherboard.

CHAPTER 3. INSTALL OPENDAYLIGHT ON THE OVERCLOUD

Red Hat OpenStack Platform director 10 provides an overcloud image that contains all of the necessary software for OpenStack services that run on overcloud nodes. However, the OpenDaylight software and dependencies are not included in the image by default. To install OpenDaylight, use the **`virt-customize`** command on the undercloud to upload the OpenDaylight RPM and install it:

Prepare the installation of OpenDaylight

1. Log on to the undercloud node.
2. Run the script:

```
$ source ~/stackrc
```

3. Install the following packages:

```
$ sudo yum -y install libguestfs-tools libvirt rhosp-director-
images
```

4. Start the **`libvirtd`** service:

```
$ sudo systemctl start libvirtd
```

5. Create the directory where you will store the images:

```
$ mkdir ~/images
```

6. Enter the directory:

```
$ cd ~/images
```

7. Extract the **`overcloud-full.tar`** file:

```
$ tar xvf /usr/share/rhosp-director-images/overcloud-full.tar
```

8. Extract the **`ironic-python-agent.tar`** file:

```
$ tar xvf  /usr/share/rhosp-director-images/ironic-python-
agent.tar
```

Install the OpenDaylight packages

1. Register the image with your RHN user:

```
$ virt-customize -a overcloud-full.qcow2 --run-command
'subscription-manager register --username <your RHN user>  --
password <RHN password>’
```

2. Attach the Red Hat Enterprise Linux subscription:

```
$ virt-customize -a overcloud-full.qcow2 --run-command
'subscription-manager attach --pool <pool id>'
```

3. Enable the necessary repositories in your image:

```
$ virt-customize -a overcloud-full.qcow2 --run-command
'subscription-manager repos --enable=rhel-7-server-rpms --
enable=rhel-7-server-extras-rpms --enable=rhel-7-server-
openstack-10-rpms'
```

4. Install the OpenDaylight package and its dependencies:

```
$ virt-customize -a overcloud-full.qcow2 --install opendaylight -
-selinux-relabel --update
```

5. Detach the subscription and unregister the image:

```
$ virt-customize -a overcloud-full.qcow2 --run-command
'subscription-manager remove --all && subscription-manager
unregister && subscription-manager clean'
```

6. If you do not have an overcloud image installed, skip this and proceed to **next step**. Otherwise update the image:

```
$ openstack overcloud image upload --update-existing --image-path
~/images/
```

7. If you have skipped **Step 6**, upload the new overcloud image to glance:

```
$ openstack overcloud image upload
```

More information

- If you need to update the images on the undercloud before you start the procedure, use:

```
$ sudo openstack overcloud image upload --update
```

- To get the Pool ID (Step 10), you can run:

```
subscription-manager list --available
```

- See the Red Hat Subscription Management guide for more details.

CHAPTER 4. CONFIGURE AND DEPLOY OPENDAYLIGHT WITH RED HAT OPENSTACK PLATFORM 10

There are two methods of deploying OpenDaylight with Red Hat OpenStack Platform 10.

The first includes running OpenDaylight on the default Controller role, while the second isolates OpenDaylight on its own node using a custom role.

Note

In Red Hat OpenStack Platform 10 only a single instance of OpenDaylight is supported running in either type of deployment. OpenDaylight HA (clustering) will be supported in a future release.

4.1. CONFIGURE AND RUN THE DEPLOYMENT

The recommended approach to installing OpenDaylight is to use the default environment file and pass it as an argument to the install command on the undercloud. This will deploy OpenDaylight using the default enviroment file **`neutron-opendaylight-l3.yaml`**.

```
$ openstack overcloud deploy --templates  -e /usr/share/openstack-
tripleo-heat-templates/environments/neutron-opendaylight-l3.yaml
```

Useful information

- The default file contains these values:

```
# A Heat environment that can be used to deploy OpenDaylight with L3
DVR
# resource_registry:
OS::TripleO::Services::NeutronOvsAgent: OS::Heat::None
OS::TripleO::Services::ComputeNeutronOvsAgent: OS::Heat::None
OS::TripleO::Services::ComputeNeutronCorePlugin: OS::Heat::None
OS::TripleO::Services::OpenDaylightApi:
../puppet/services/opendaylight-api.yaml
OS::TripleO::Services::OpenDaylightOvs:
../puppet/services/opendaylight-ovs.yaml
OS::TripleO::Services::NeutronL3Agent: OS::Heat::None

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronMechanismDrivers: 'opendaylight_v2'
  NeutronServicePlugins: "odl-router_v2"
  OpenDaylightEnableL3: "'yes'"
```

- In Red Hat OpenStack Platform director, the **`resource_registry`** is used to map resources for a deployment to the corresponding *yaml* resource definition file. Services are one type of resource that can be mapped. The **`OS::Heat::None`** option disables services that will not be used. In this example, **OpenDaylightApi** and **OpenDaylightOvs** services are enabled, while default neutron agents are explicitly disabled.

- The heat parameters are normally set inside the director. You can override their default values by using the **`parameter_defaults`** section of the environment file. In the above example, certain parameters are overridden to enable OpenDaylight *Layer 3* functionality.

Important

There is another default OpenDaylight environment file (**`neutron-opendaylight.yaml`**) that only enables *Layer 2* functionality. However, support for that configuration is deprecated and it should not be used.

Note

The list of other services and their configuration options are provided further in the text.

4.1.1. Configure the OpenDaylight API Service

You can configure the OpenDaylight API Service by editing **`opendaylight-api.yaml`**, located in the **`/usr/share/openstack-tripleo-heat-templates/puppet/services/`** directory.

Configurable options

You can configure the following options:

OpenDaylightPort	Sets the port used for Northbound communication. Defaults to *8081*.
OpenDaylightUsername	Sets the login username for OpenDaylight. Defaults to *admin*. Overriding this parameter is currently unsupported.
OpenDaylightPassword	Sets the login password for OpenDaylight. Defaults to *admin*. Overriding this parameter is currently unsupported.
OpenDaylightEnableL3	L3 DVR functionality cannot be disabled with 'odl-netvirt-openstack' OpenDaylight feature. *Deprecated.*
OpenDaylightEnableDHCP	Enables OpenDaylight to act as the DHCP service. Defaults to *false*. Overriding this parameter is currently unsupported.
OpenDaylightFeatures	Comma-delimited list of features to boot in OpenDaylight. Defaults to [**odl-netvirt-openstack**, **odl-netvirt-ui**]. When using OpenDaylight in the deprecated Layer 2 only mode, this parameter must be overridden to use the **odl-ovsdb-openstack** feature.

4.1.2. Configure the OpenDaylight OVS Service

Configure the OpenDaylight OVS Service by changing the values in **`opendaylight-ovs.yaml`**, located in **`/usr/share/openstack-tripleo-heat-templates/puppet/services/`**.

Configurable options

You can configure the following:

OpenDaylightPort	Sets the port used for Northbound communication to OpenDaylight. Defaults to *8081*. The OVS Service uses the Northbound to query OpenDaylight to ensure that it is fully up before connecting.
OpenDaylightConnectionProtocol	*Layer 7* protocol used for *REST* access. Defaults to *http*. Currently, *http* is the only supported protocol in OpenDaylight.
OpenDaylightCheckURL	The URL to use to verify OpenDaylight is fully up before OVS connects. Defaults to *restconf/operational/network-topology:network-topology/topology/netvirt:1*
OpenDaylightProviderMappings	Comma-delimited list of mappings between logical networks and physical interfaces. This setting is required for VLAN deployments. Defaults to *datacentre:br-ex*.

4.1.3. Using Neutron Metadata Service with OpenDaylight

The OpenStack Compute service allows virtual machines to query metadata associated with them by making a web request to a special address, *169.254.169.254*. The OpenStack Networking proxies such requests to the **nova-api**, even when the requests come from isolated or multiple networks with overlapping IP addresses.

The Metadata service uses either the neutron **L3** agent router to serve the metadata requests or the **DHCP** agent instance. Deploying OpenDaylight with the *Layer 3* routing plug-in enabled disables the neutron **L3** agent. Therefore Metadata must be configured to flow through the **DHCP** instance, even when a router exists in a tenant network. To configure this, set the **`NeutronEnableForceMetadata`** to **`true`**.

Note

This functionality is already enabled in the default environment file **neutron-opendaylight-l3.yaml**.

VM instances will have a static host route installed, using the *DHCP* option 121, for *169.254.169.254/32*. With this static route in place, Metadata requests to *169.254.169.254:80* will go to the Metadata nameserver proxy in the *DHCP* network namespace. The namespace proxy then adds the *HTTP* headers with the instance's IP to the request, and connects it to the Metadata agent

through the Unix domain socket. The Metadata agent queries neutron for the instance ID that corresponds to the source IP and the network ID and proxies it to the nova Metadata service. The additional *HTTP* headers are required to maintain isolation between tenants and allow overlapping IP support.

4.1.4. Configure Network and NIC template

In Red Hat OpenStack Platform director, the physical neutron network data center is mapped to an OVS bridge called *br-ex* by default. It is consistently the same with the OpenDaylight integration. If you use the default OpenDaylightProviderMappings and plan to create a flat or *VLAN _External* network, you have to configure the OVS *br-ex* bridge in the *NIC* template for Compute nodes. Since the *Layer 3* plug-in uses distributed routing to these nodes, it is not necessary to configure *br-ex* on the controller role *NIC* template any more.

The *br-ex* bridge can be mapped to any network in network isolation, but it is typically mapped to the *External* network as you can see in the example.

```
type: ovs_bridge
  name: {get_input: bridge_name}
  use_dhcp: false
  members:
    -
      type: interface
      name: nic3
      # force the MAC address of the bridge to this interface
      primary: true
  dns_servers: {get_param: DnsServers}
  addresses:
    -
      ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    -
      default: true
      ip_netmask: 0.0.0.0/0
      next_hop: {get_param: ExternalInterfaceDefaultRoute}
```

Note

When using network isolation, you do not have to place an IP address, or a default route, on this bridge on Compute nodes.

Alternatively, it is possible to configure external network access without using the *br-ex* bridge at all. To use the method, you must know the interface name of the overcloud Compute node in advance. For example, if **eth3** is the deterministic name of the third interface on the Compute node, then will specify an interface in the *NIC* template for the Compute node:

```
-
  type: interface
  name: eth3
  use_dhcp: false
```

Having configured the *NIC* template, you must override the **OpenDaylightProviderMappings** parameter to map the interface to the physical neutron network: **`datacentre:eth3`**. This will cause OpenDaylight to move the **eth3** interface onto the *br-int* bridge.

The OVS *br-int* bridge is used to carry the tenant traffic. OpenDaylight supports *VXLAN* and *VLAN* type tenant networks. Red Hat OpenStack Platform director will automatically configure the OVS to use the correct IP interface for *VXLAN* tenant traffic when *VXLAN* tenant network types are used.

You do not need to create the *br-int* bridge in the *NIC* template files. OpenDaylight will create the bridge automatically. However, if you use *VLAN* tenant networks, you may wish to configure one more physical neutron network and include that interface mapping in the **OpenDaylightProviderMappings**.

OpenDaylight will then move that interface to the *br-int* bridge. Regarding such behavior, you must use different interfaces to use both the *VXLAN* and *VLAN* tenant networks. Another way is to use an extra bridge within the **OpenDaylightProviderMappings** for tenant networks.

Note

For example, you could use **`tenant:br-isolated`**, where *br-isolated* is an OVS bridge that contains the tenant network interface and is also configured with an IP. OpenDaylight will create a patch port from OVS bridge *br-int* to *br-isolated*. This way, you can use the *br-isolated* bridge for *VXLAN* traffic, as well as transporting the VLAN traffic.

4.2. INSTALL OPENDAYLIGHT IN CUSTOM ROLE

Installing OpenDaylight in a custom role results in an isolated **`OpenDaylightApi`** service, running on a node that is not the controller.

To use a custom role for OpenDaylight, follow this procedure:

Install OpenDaylight using a special role file.

1. Copy the existing **`roles_data.yaml`** file to a new file:

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
my_roles_data.yaml
```

2. Modify the default controller role and remove the **OpenDaylightApi** service from the controller section of the new file (line 5):

```
  - name: Controller
    CountDefault: 1
    ServicesDefault:
     - OS::TripleO::Services::TripleoFirewall
     - OS::TripleO::Services::OpenDaylightApi #<--Remove this
     - OS::TripleO::Services::OpenDaylightOvs
```

3. Create the OpenDaylight role in the new file in the OpenDaylight section:

```
- name: OpenDaylight
  CountDefault: 1
  ServicesDefault:
```

```
      - OS::TripleO::Services::Kernel
      - OS::TripleO::Services::Ntp
      - OS::TripleO::Services::OpenDaylightApi
      - OS::TripleO::Services::TripleoPackages
      - OS::TripleO::Services::TripleoFirewall
```

4. Run the installation command using the **-r** argument (see Related information of this procedure). In this example, there are three ironic nodes in total, from which one is reserved for the custom OpenDaylight role:

```
$ openstack overcloud deploy --templates  -e
/usr/share/openstack-tripleo-heat-templates/environments/neutron-
opendaylight-l3.yaml -e network-environment.yaml --compute-scale
1 --ntp-server 0.se.pool.ntp.org --control-flavor control --
compute-flavor compute -r my_roles_data.yaml
```

5. List the instances:

```
$ nova list
```

6. Verify that the new OpenDaylight role is dedicated as an instance. The following is an example output of the previous command:

```
+--------------------------------------+--------------------
-----+--------+------------+-------------+--------------------
+
| ID                                   | Name
| Status | Task State | Power State | Networks           |
+--------------------------------------+--------------------
-----+--------+------------+-------------+--------------------
+
| 360fb1a6-b5f0-4385-b68a-ff19bcf11bc9 | overcloud-controller-0
| BUILD  | spawning   | NOSTATE     | ctlplane=192.0.2.4 |
| e38dde02-82da-4ba2-b5ad-d329a6ceaef1 | overcloud-novacompute-0
| BUILD  | spawning   | NOSTATE     | ctlplane=192.0.2.5 |
| c85ca64a-77f7-4c2c-a22e-b71d849a72e8 | overcloud-opendaylight-0
| BUILD  | spawning   | NOSTATE     | ctlplane=192.0.2.8 |
+--------------------------------------+--------------------
-----+--------+------------+-------------+--------------------
+
```

More information

- This argument is used to override the role definitions within Red Hat OpenStack Platform director at installation time:

```
-r <roles_data>.yaml
```

- Using a custom role requires an extra ironic node that will be used for the custom role during the installation.

CHAPTER 5. TEST THE DEPLOYMENT

5.1. PERFORM A BASIC TEST

The basic test will verify that instances are able to ping each other. It will also check the *Floating IP SSH* access. This example describes how you can perform the test from the undercloud.

This procedure requires you to follow a large number of individual steps; for convenience, the procedure was divided into smaller parts. However, the steps must be followed in the given order.

Note

In this setup, a flat network is used to create the *External* network, and *VXLAN* is used for the *Tenant* networks. *VLAN External* networks and *VLAN Tenant* networks are also supported, depending on the desired deployment.

5.1.1. Create a new network for testing

1. Source the credentials to access the overcloud:

```
$ source /home/stack/overcloudrc
```

2. Create an external neutron network that will be used to access the instance from outside of the overcloud:

```
$ openstack network create --external --project $(openstack project
show service | grep id | awk '{ print $4 }')  --provider-network-type
flat --provider-physical-network external
```

3. Create the corresponding neutron subnet for the new external network (created in the previous step):

```
$ openstack subnet create  --project $(openstack project show service |
grep id | awk '{ print $4 }') --no-dhcp --network external --gateway
192.168.37.1 --allocation-pool start=192.168.37.200,end=192.168.37.220
--subnet-range 192.168.37.0/24 external-subnet
```

4. Download the cirros image to be used for creating overcloud instances:

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

5. Upload the cirros image into glance on the overcloud:

```
$ openstack image create cirros --public --file ./cirros-0.3.4-x86_64-
disk.img --disk-format qcow2 --container-format bare
```

6. Create a **`tiny`** flavor to use for overcloud instances:

```
$ openstack flavor create m1.tiny --ram 512 --disk 1 --public
```

7. Create a VXLAN-based tenant network to host the instances:

```
$ openstack network create net_test --provider-network-type=vxlan --
provider-segment 100
```

8. Create a subnet for the tenant network (created in the previous step):

```
$ openstack subnet create --network net_test --subnet-range
123.123.123.0/24 test
```

9. Find and store the ID of the tenant network:

```
$ net_mgmt_id=$(openstack network list | grep net_test | awk '{print
$2}')
```

10. Create an instance called **`cirros1`** and attach it to the **`net_test`** network:

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-
id=$net_mgmt_id cirros1
```

11. Create a second instance called **`cirros2`**, also attached to the **`net_test`** network:

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-
id=$net_mgmt_id cirros2
```

5.1.2. Set up networking in the test environment

1. Find and store the ID of the admin project:

```
$ admin_project_id=$(openstack project list | grep admin | awk '{print
$2}')
```

2. Find and store the admin project's default security group:

```
$ admin_sec_group_id=$(openstack security group list | grep
$admin_project_id | awk '{print $2}')
```

3. Add a rule to the admin default security group to allow ICMP traffic ingress:

```
$ openstack security group rule create $admin_sec_group_id --protocol
icmp --ingress
```

4. Add a rule to the admin default security group to allow ICMP traffic egress:

```
$ openstack security group rule create $admin_sec_group_id --protocol
icmp --egress
```

5. Add a rule to the admin default security group to allow SSH traffic ingress:

```
$ openstack security group rule create $admin_sec_group_id --protocol
tcp --dst-port 22 --ingress
```

6. Add a rule to the admin default security group to allow SSH traffic egress:

```
$ openstack security group rule create $admin_sec_group_id --protocol
tcp --dst-port 22 --egress
```

5.1.3. Test the connectivity

1. From horizon, you should be able to access the *novnc* console for an instance. Use the password from ***overcloudrc*** to login to horizon as *admin*. The default login for *cirros* images is the username **`cirros`**, and **`cubswin:)`** as the password.

2. From the *novnc* console, verify that the instance received a DHCP address:

```
$ ip addr show
```

Note

Another method of doing this is by using the **`nova console-log <instance id>`** from the undercloud, which will show if a DHCP lease was obtained.

3. Now repeat the steps 1 and 2 for all other instances.

4. From one instance, attempt to ping the other instances. This will validate the basic *Tenant* network connectivity in the overcloud.

5. Verify that you can reach other instances by using a *Floating IP*.

5.1.4. Create devices

1. Create a floating IP on the external network to be associated with cirros1 instance:

```
$ openstack floating ip create external
```

2. Create a router which will be used to handle NAT between the floating IP and cirros1 tenant IP:

```
$ openstack router create test
```

3. Set the gateway of the router to be the external network:

```
$ neutron router-gateway-set test external
```

4. Add and interface to the router attached to the tenant network:

```
$ neutron router-interface-add test test
```

5. Find and store the floating IP created in Step 23:

```
floating_ip=$(openstack floating ip list | head -n -1 | grep -Eo '[0-
9]+\.[0-9]+\.[0-9]+\.[0-9]+')
```

6. Associate the floating IP with the **`cirros1`** instance:

```
$ openstack server add floating ip cirros1 $floating_ip
```

7. From a node that has external network access, attempt to login to the instance:

```
$ ssh cirros@$floating_ip
```

5.2. PERFORM ADVANCED TESTS

Several components of the OpenDaylight configuration and deployment may be checked post deployment. To test specific parts of the installation, you need to follow several procedures. Each procedure is described separately.

The procedures are to be performed on the **overcloud** nodes.

5.2.1. Connect to overcloud nodes

1. Login onto the undercloud.
2. Run the following command to start the process:

```
$ source /home/stack/stackrc
```

3. List all instances:

```
$ nova list
```

4. Choose the required instance and note its IP address in the list:
5. Connect to the machine. You will use the IP address from the list above:

```
$ ssh heat-admin@<IP from step 3>
```

6. Switch to superuser:

```
$ sudo -i
```

5.2.2. Test OpenDaylight

To test that OpenDaylight is working, you have to verify that the service is up and that the particular features are correctly loaded.

1. As a superuser, login to the node.
2. Verify that the OpenDaylight service is active:

```
# systemctl status opendaylight
```

3. Verify that HAProxy is properly configured to listen on port 8081:

```
# grep -A7 opendaylight /etc/haproxy/haproxy.cfg
```

4. Connect to the karaf account:

```
$ ssh -p 8101 karaf@localhost
```

5. List the installed features.

```
$ feature:list | grep odl-netvirt-openstack
```

6. Verify that the API is up and running.

```
# web:list | grep neutron
```

7. Verify that inter-nodes' *VXLAN* tunnels are up.

```
# vxlan:show
```

8. To test that the REST API is responding correctly, you can list the modules that are using it.

```
# curl -u "admin:admin" http://localhost:8181/restconf/modules
```

The output will be similar (the example has been shortened).

```
{"modules":{"module":[{"name":"netty-event-
executor","revision":"2013-11-
12","namespace":"urn:opendaylight:params:xml:ns:yang:controller:n
etty:eventexecutor"},{"name" ...
```

9. You can list the REST streams.

```
# curl -u "admin:admin" http://localhost:8181/restconf/streams
```

You will get something like this:

```
{"streams":{}}
```

10. Enter the following command to verify that NetVirt is ready and running.

```
# curl -u "admin:admin"
http://localhost:8181/restconf/operational/network-
topology:network-topology/topology/netvirt:1
```

The following output will confirm it.

```
{"topology":[{"topology-id":"netvirt:1"}]}
```

More information

- Step 3: As mentioned before, OpenDaylight is not running in HA mode yet, therefore the service is only active on one node.
- Step 5: If there is an **x** in the third column of the list, as generated during the procedure, then the feature is correctly installed.
- Step 6: This API endpoint is set in **`/etc/neutron/plugins/ml2/ml2_conf.ini`** and used

by the neutron to communicate with OpenDaylight.

5.2.3. Test Open vSwitch

In order to validate **Open vSwitch**, connect to one of the Compute nodes and verify that it is properly configured and connected to OpenDaylight.

1. Connect to one of the Compute nodes in the overcloud as a superuser.
2. List the Open vSwitch settings.

```
# ovs-vsctl show
```

3. Notice multiple Managers in the output (lines 2 and 3 in the example).

```
4b624d8f-a7af-4f0f-b56a-b8cfabf7635d
    Manager "ptcp:6639:127.0.0.1"
    Manager "tcp:192.0.2.4:6640"
        is_connected: true
    Bridge br-extu
        Port br-ex
            Interface br-ex
                type: internal
        Port "eth2"
            Interface "eth2"
        Port br-ex-int-patch
            Interface br-ex-int-patch
                type: patch
                options: {peer=br-ex-patch}
    Bridge br-int
        Controller "tcp:192.0.2.4:6653"
            is_connected: true
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port br-ex-patch
            Interface br-ex-patch
                type: patch
                options: {peer=br-ex-int-patch}
    ovs_version: "2.5.0"
```

4. Verify that the **`tcp`** manager points to the IP of the node where OpenDaylight is running.
5. Verify that the Managers show **`is_connected: true`** to ensure that connectivity to OpenDaylight from OVS is established and uses the OVSDB protocol.
6. Verify that each bridge (other than *br-int*) exists and matches the NIC template used for deployment with the Compute role.
7. Verify that the *tcp* connection corresponds to the IP where the OpenDaylight service is running.
8. Verify that the bridge *br-int* shows **`is_connected: true`** and an OpenFlow protocol connection to OpenDaylight is established.

More information

- The *br-int* bridge is created automatically by OpenDaylight.

5.2.4. Verify the Open vSwitch configuration on Compute nodes.

1. Connect to a Compute node as a superuser.
2. List the *Open vSwitch* configuration settings.

```
# ovs-vsctl list open_vswitch
```

3. Read the output. It will be similar to this example.

```
_uuid               : 4b624d8f-a7af-4f0f-b56a-b8cfabf7635d
bridges             : [11127421-3bcc-4f9a-9040-ff8b88486508,
350135a4-4627-4e1b-8bef-56a1e4249bef]
cur_cfg             : 7
datapath_types      : [netdev, system]
db_version          : "7.12.1"
external_ids        : {system-id="b8d16d0b-a40a-47c8-a767-
e118fe22759e"}
iface_types         : [geneve, gre, internal, ipsec_gre, lisp,
patch, stt, system, tap, vxlan]
manager_options     : [c66f2e87-4724-448a-b9df-837d56b9f4a9,
defec179-720e-458e-8875-ea763a0d8909]
next_cfg            : 7
other_config        : {local_ip="11.0.0.30",
provider_mappings="datacentre:br-ex"}
ovs_version         : "2.5.0"
ssl                 : []
statistics          : {}
system_type         : RedHatEnterpriseServer
system_version      : "7.3-Maipo"
```

4. Verify that the value of the **`other_config`** option has the correct **`local_ip`** set for the local interface that connects to the Tenant network through *VXLAN* tunnels.
5. Verify that the **`provider_mappings`** value under the **`other_config`** option matches the value given in the **OpenDaylightProviderMappings** heat template parameter. This configuration maps the neutron logical networks to corresponding physical interfaces.

5.2.5. Verify neutron configuration

1. Connect to the superuser account on one of the controller role nodes.
2. Make sure that the file **`/etc/neutron/neutron.conf`** contains **`service_plugins=odl-router_v2`**.
3. Check that the file **`/etc/neutron/plugin.ini`** contains the following **ml2** configuration:

```
[ml2]
mechanism_drivers=opendaylight_v2
```

```
[ml2_odl]
password=admin
username=admin
url=http://192.0.2.9:8081/controller/nb/v2/neutron
```

4. On one of the **overcloud** controllers, verify that neutron agents are running properly.

```
# openstack network agent list
```

5. Verify that both the Metadata and DHCP agents are in the up state (the **`admin_state_up`** option is **`True`**):

```
+--------------------------------------+----------------+-----
--------------------+-------------------+-------+------------
----+------------------------+
| id                                   | agent_type     | host
| availability_zone | alive | admin_state_up | binary
|
+--------------------------------------+----------------+-----
--------------------+-------------------+-------+------------
----+------------------------+
| 3be198c5-b3aa-4d0e-abb4-51b29db3af47 | Metadata agent |
controller-0.localdomain |                   | :-)   | True
| neutron-metadata-agent |
| 79579d47-dd7d-4ef3-9614-cd2f736043f3 | DHCP agent     |
controller-0.localdomain | nova              | :-)   | True
| neutron-dhcp-agent     |
+--------------------------------------+----------------+-----
--------------------+-------------------+-------+------------
----+------------------------+
```

More information

- The IP in the **`plugin.ini`**, mentioned in step 3, should be the **InternalAPI** Virtual IP Address (VIP).
- Note, that there is no Open vSwitch agent, nor L3 agent, listed in output of step 5, which is a desired state, as both are now managed by OpenDaylight.

CHAPTER 6. DEBUGGING

6.1. LOCATE THE LOGS

6.1.1. Access OpenDaylight logs

The OpenDaylight logs are stored on OpenDaylight nodes, where you can find them in the **`/opt/opendaylight/data/log/`** directory.

OpenDaylight stores its logs in the **`karaf.log`** file.

- The latest log is named **`karaf.log`**, while any older logs are numbered, such as **`karaf.log.1`**, and so on.

6.1.2. Access OpenDaylight logs through karaf shell

Another way to access the logs is to login to the karaf shell on the OpenDaylight node and display the log files.

1. Connect to the karaf account:

```
$ ssh -p 8101 karaf@localhost
```

2. Enable trace level logging on NetVirt.

```
$ log set TRACE org.opendaylight.netvirt
```

3. If you need to tail the logs inside of the karaf shell, use

```
$ log:tail
```

More information

- The karaf shell helps users enable different logging levels for any OpenDaylight feature.
- If you enable TRACE, it is possible that you will receive an extremely big number of log files.

6.1.3. Access OpenStack Networking logs

When OpenStack network-related commands fail, the first step is to examine the neutron logs:

These logs are stored in **`server.log`**, located on each neutron node in the **`/var/log/neutron`** directory.

- The **`server.log`** file also includes errors about the communication with OpenDaylight.
- If the neutron error originates from interacting with OpenDaylight, it is necessary to examine the OpenDaylight logs as well, to locate the cause of the failure.

6.2. DEBUG NETWORKING ERRORS

If you experience network error (for example, there is no instance connectivity), but no errors are reported when issuing OpenStack commands or in the neutron logs, then it may be useful to inspect the OVS nodes for network traffic and OpenFlow flows:

1. Login (as the superuser) to the affected node where the network error has occurred.
2. Display the information about the *br-int* switch.

```
# ovs-ofctl -O openflow13 show br-int
```

3. Examine the output. It will be similar to this example:

```
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000e4c153bdb306
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS
QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(br-ex-patch): addr:ae:38:01:09:66:5b
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 2(tap1f0f610c-8e): addr:00:00:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 3(tun1147c81b59c): addr:66:e3:d2:b3:b8:e3
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:e4:c1:53:bd:b3:06
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x5): frags=normal
miss_send_len=0
```

4. List the statistics for the *br-int* switch.

```
# ovs-ofctl -O openflow13 dump-ports br-int
```

5. Examine the output. It will be similar to this example:

```
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port LOCAL: rx pkts=101215, bytes=6680190, drop=0, errs=0,
frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=90117.708s
  port  1: rx pkts=126887, bytes=8970074, drop=0, errs=0,
frame=0, over=0, crc=0
           tx pkts=18764, bytes=2067792, drop=0, errs=0, coll=0
           duration=90117.418s
  port  2: rx pkts=1171, bytes=70800, drop=0, errs=0, frame=0,
over=0, crc=0
           tx pkts=473, bytes=44448, drop=0, errs=0, coll=0
           duration=88644.819s
```

```
  port  3: rx pkts=120197, bytes=8776126, drop=0, errs=0,
frame=0, over=0, crc=0
           tx pkts=119408, bytes=8727254, drop=0, errs=0, coll=0
           duration=88632.426s
```

More information

- In step 3, you can see that there are three ports created on this OVS node. The first is a patch port going to the bridge *br-ex*, which in this scenario is used for *External* network connectivity. The second port is a tap port which connects to a DHCP agent instance (we know this because the host is a controller, otherwise on a Compute role it would be an instance), while the third port is a *VXLAN* tunnel port created for the tenant traffic. When you know what each port is, you can examine the port statistics to verify that the port is indeed receiving/sending traffic (see Step 4).
- From the output in Step 5, you can see that each port is receiving (**`rx pkts`**) and sending packets (**`tx pkts`**).

6.2.1. Advanced debugging using OpenFlow flows

For advanced users who are familiar with OpenFlow, the next level of debugging is to examine the flows on the switch in order to detect where traffic is being dropped.

You can list the flows, as well as how many packets have hit them, by entering the following command:

```
# ovs-ofctl -O openflow13 dump-flows br-int
```

The output of the command will provide you with the necessary information.

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x8000000, duration=90071.665s, table=0, n_packets=126816,
n_bytes=8964820, priority=1,in_port=1
actions=write_metadata:0x20000000001/0xffffff0000000001,goto_table:17
 cookie=0x8000000, duration=88967.292s, table=0, n_packets=473,
n_bytes=44448, priority=4,in_port=2
actions=write_metadata:0x40000000000/0xffffff0000000001,goto_table:17
 cookie=0x8000001, duration=88954.901s, table=0, n_packets=120636,
n_bytes=8807869, priority=5,in_port=3
actions=write_metadata:0x70000000001/0x1fffff0000000001,goto_table:36
 cookie=0x8000001, duration=90069.534s, table=17, n_packets=126814,
n_bytes=8964712, priority=5,metadata=0x20000000000/0xffffff0000000000
actions=write_metadata:0xc0000200000222e0/0xfffffffffffff
ffe,goto_table:19
 cookie=0x8040000, duration=90069.533s, table=17, n_packets=126813,
n_bytes=8964658,
priority=6,metadata=0xc000020000000000/0xffffff0000000000
actions=write_metadata:0xe00002138a000000/0xffffffff
fffffffe,goto_table:48
 cookie=0x8040000, duration=88932.689s, table=17, n_packets=396,
n_bytes=36425,
priority=6,metadata=0xc000040000000000/0xffffff0000000000
actions=write_metadata:0xe00004138b000000/0xfffffffffffff
ffe,goto_table:48
```

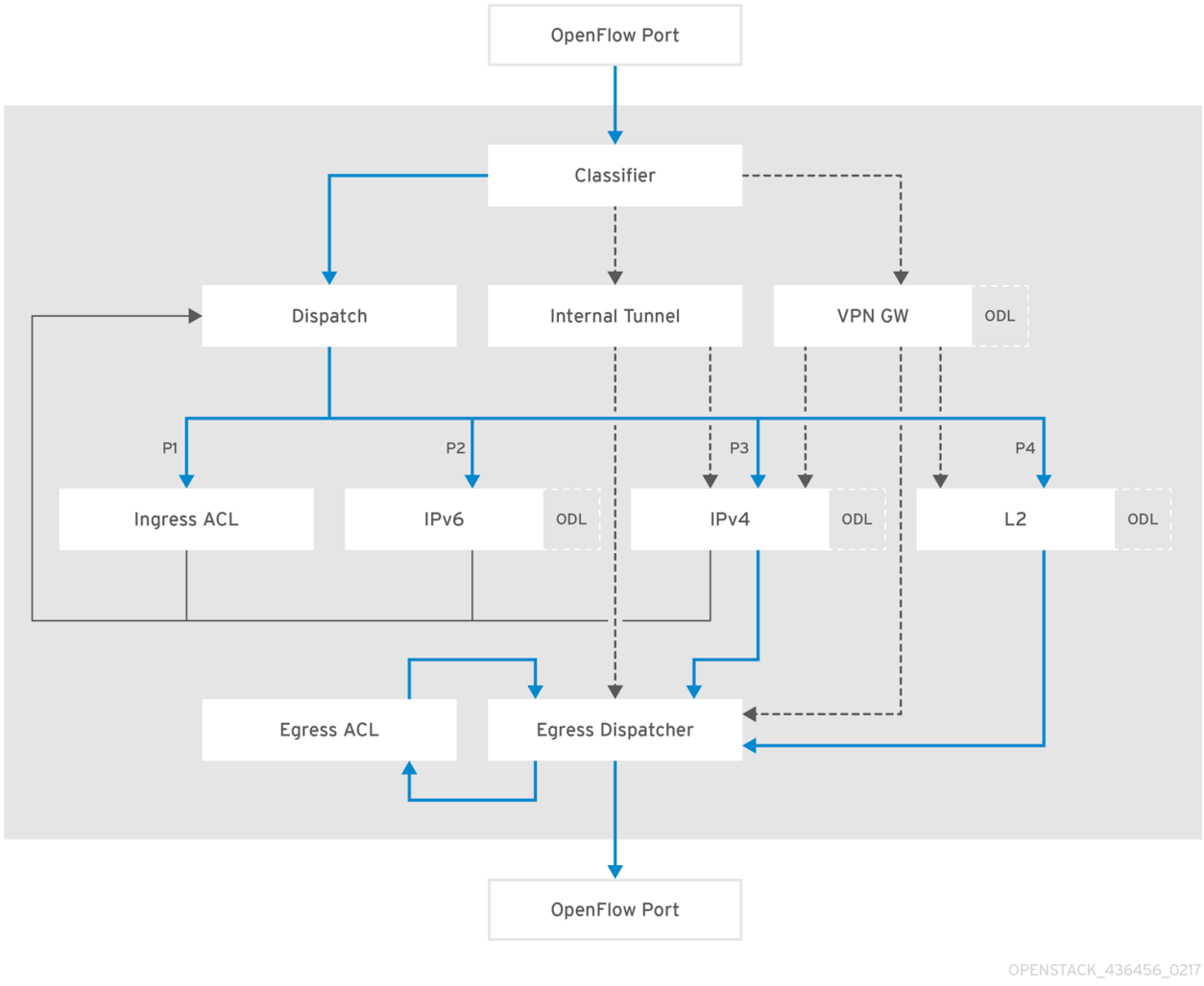
Note

The sample output has been edited for length.

6.2.2. Packet traverse in OpenFlow

The important things to understand are that the network functions performed on a packet are broken into different OpenFlow tables, and packets traverse those tables in order, starting from zero. An incoming packet lands in table 0, and then progresses via the *OpenFlow Pipeline* until it is sent out of a port, to the OpenDaylight Controller, or dropped. A packet may skip one or more tables depending on which network function it may need to go to. The full diagram of tables and how they correspond to network functions is shown below:

Figure 6.1. OpenDaylight NetVirt OpenFlow *Pipeline*



CHAPTER 7. MODEL INSTALLATION SCENARIO

In this part you will explore an example of OpenDaylight installation using OpenStack in a production environment. In this scenario, tunneling (*VXLAN*) is used for tenant traffic separation.

7.1. PHYSICAL TOPOLOGY

The topology of this scenario consists of six nodes:

- 1 x director undercloud node
- 3 x OpenStack overcloud controllers; the first one has the OpenDaylight SDN controller installed in addition to other OpenStack services
- 2 x OpenStack overcloud Compute nodes

7.2. PLANNING PHYSICAL NETWORK ENVIRONMENT

The overcloud controller nodes use three network interface cards (NICs) each:

Name	Purpose
nic1	Management network (e.g accessing the node via SSH)
nic2	Tenant (*VXLAN*) carrier, provisioning (PXE, DHCP), and *Internal API* networks
nic3	Public API network access

The overcloud Compute nodes are equipped with three NICs:

Name	Purpose
nic1	Management network
nic2	Tenant carrier, provisioning, and *Internal API* networks
nic3	*External* (Floating IPs) network

The undercloud node is equipped with two NICs:

Name	Purpose
nic1	Used for the Management network
nic2	Used for the Provisioning network

7.3. PLANNING NIC CONNECTIVITY

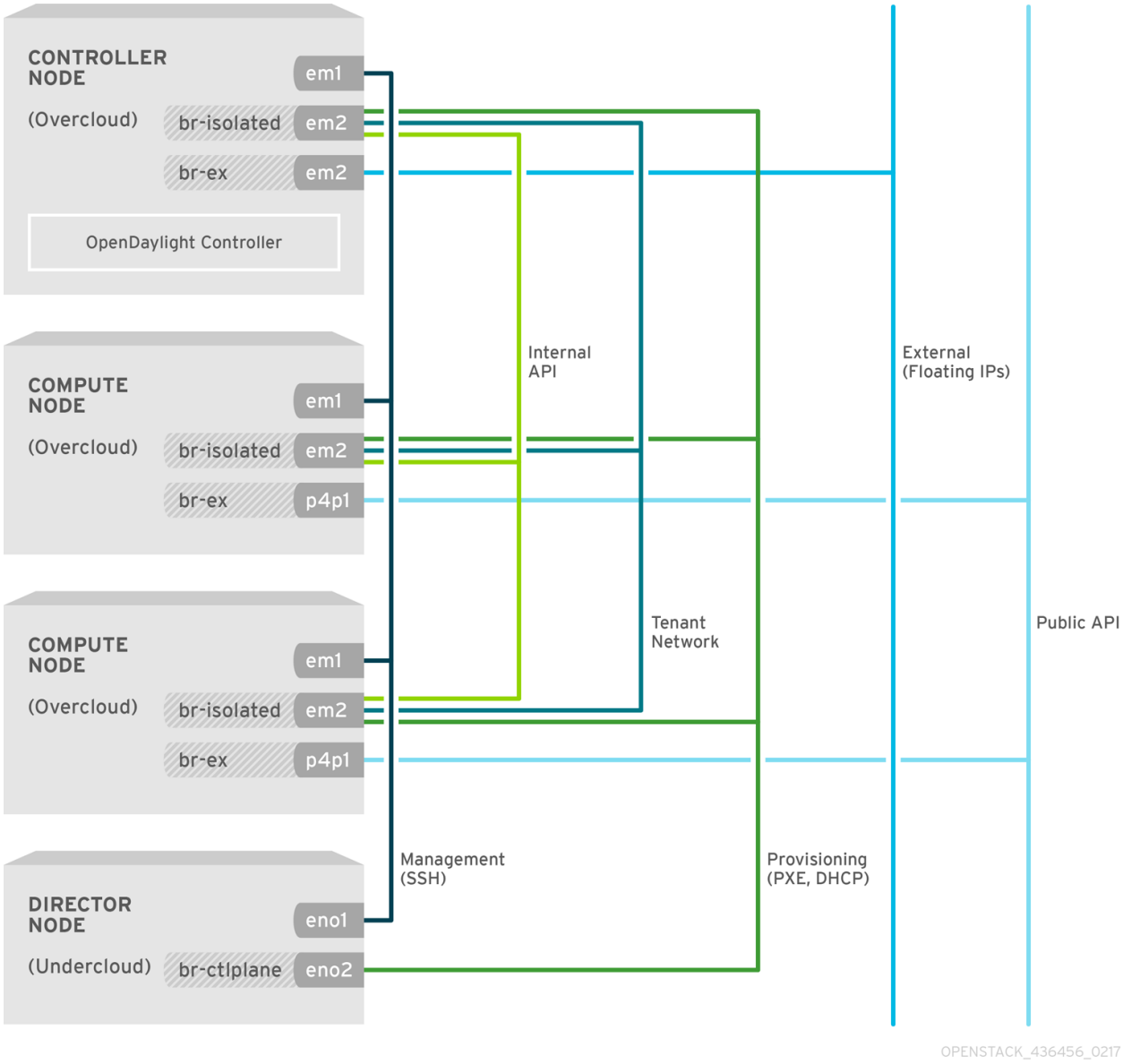
In this case, the environment files use abstracted numbered interfaces (**nic1**, **nic2**) and not the actual device names presented on the host operating system (like **eth0** or **eno2**). The hosts that belong to the same role do not require identical network interface device names. There is no problem if one host uses the **em1** and **em2** interfaces, while the other uses **eno1** and **eno2**. All NICs values will be accessed as **nic1** and **nic2**.

The abstracted NIC scheme only relies on interfaces that are alive and connected. In cases, when the hosts have a different number of interfaces, it is enough to use the minimal number of interfaces that you need to connect the hosts. For example, if there are four physical interfaces on one host and six on the other, you should only use **nic1**, **nic2**, **nic3**, and **nic4** and plug in four cables on both hosts.

7.4. PLANNING NETWORKS, VLANS AND IPS

In this scenario, network isolation is used to separate the Management, Provisioning, *Internal API*, Tenant, Public API, and Floating IPs network traffic.

Figure 7.1. Detailed network topology used in this scenario



The table shows the VLAN ID and IP subnet associated with each network:

Network	VLAN ID	IP Subnet
Provisioning	Native	192.0.5.0/24
Internal API	600	172.17.0.0/24
Tenant	603	172.16.0.0/24
Public API	411	10.35.184.144/28
Floating IP	412	10.35.186.146/28

The OpenStack Platform director creates the *br-isolated* OVS bridge and adds the *VLAN* interfaces for each network as defined in the network configurations files. The *br-ex* bridge, too, is created automatically by the director with the relevant network interface attached to it.

Make sure, that your physical network switches that provide connectivity between the hosts are properly configured to carry those *VLAN IDs*. You must configure all switch ports facing the hosts as "trunks" with the above mentioned *VLANs*. The term "trunk" is used here to describe a port that allows multiple *VLAN IDs* to traverse through the same port.

Note

Configuration guidance for the physical switches is outside the scope of this document.

Note

The **TenantNetworkVlanID** in **`network-environment.yaml`** is where a VLAN tag can be defined for Tenant network when using *VXLAN* tunneling (i.e *VXLAN* tenant traffic transported over a VLAN tagged underlay network). This value may also be empty if the Tenant network is desired to run over the native VLAN. Also note, that when using VLAN tenant type networks, VLAN tags other than the value provided for **TenantNetworkVlanID** may be used.

7.5. OPENDAYLIGHT CONFIGURATION FILES REFERENCE

To deploy the model installation of OpenStack, the following commands were entered on the undercloud node:

```
$ source ~/stackrc
```

```
$ openstack overcloud deploy --debug \
  --templates \
  --environment-file "$HOME/extra_env.yaml" \
  --libvirt-type kvm \
  -e /home/stack/baremetal-vlan/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-
opendaylight-l3.yaml \
  --log-file overcloud_install.log &> overcloud_install.log
```

7.5.1. The *extra_env.yaml* file

This file has only one parameter:

```
  parameter_defaults:
    OpenDaylightProviderMappings: 'datacentre:br-ex,tenant:br-isolated'
```

These are the mappings that each node, controlled by OpenDaylight, will use. The physical network data center will be mapped to the *br-ex OVS* bridge and tenant network traffic will be mapped to the *br-isolated OVS* bridge.

7.5.2. The *undercloud.conf* file

This file is located in the **/home/stack/baremetal-vlan/** directory.

```
  [DEFAULT]
  local_ip = 192.0.5.1/24
  network_gateway = 192.0.5.1
  undercloud_public_vip = 192.0.5.2
  undercloud_admin_vip = 192.0.5.3
  local_interface = eno2
  network_cidr = 192.0.5.0/24
  masquerade_network = 192.0.5.0/24
  dhcp_start = 192.0.5.5
  dhcp_end = 192.0.5.24
  inspection_iprange = 192.0.5.100,192.0.5.120
```

In this example, the 192.0.5.0/24 subnet for the Provisioning network is used. Note that the physical interface **eno2** is used on the undercloud node for provisioning.

7.5.3. The *network-environment.yaml* file

This is the main file for configuring the network. It is located in the **/home/stack/baremetal-vlan/** directory. In the following file the VLAN IDs and IP subnets are specified for the different networks, as well as the provider mappings.

Also note, the two files under nic-configs directory **controller.yaml** and **compute.yaml** are used for specifying the network configuration for the controller and Compute nodes.

The number of controller nodes (3) and Compute nodes (2) is specified in the example.

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to
use for
  # override the default.
  OS1::TripleO::Compute::Net::SoftwareConfig: nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-
configs/controller.yaml

  # Network isolation configuration
  # Service section
  # If some service should be disable, use the following example
  # OS::TripleO::Network::Management: OS::Heat::None
    OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
    OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-
heat-templates/network/internal_api.yaml
    OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml
    OS::TripleO::Network::Management: OS::Heat::None
    OS::TripleO::Network::StorageMgmt: OS::Heat::None
    OS::TripleO::Network::Storage: OS::Heat::None

  # Port assignments for the VIPs
    OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Network::Ports::InternalApiVipPort:
/usr/share/openstack-tripleo-heat-
```

```
templates/network/ports/internal_api.yaml
    OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/vip.yaml
    OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
    OS::TripleO::Network::Ports::StorageMgmtVipPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the controller role
    OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Controller::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
    OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
    OS::TripleO::Controller::Ports::ManagementPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml
    OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
    OS::TripleO::Controller::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the Compute role
    OS::TripleO::Compute::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
    OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
    OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
    OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
    OS::TripleO::Compute::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for service virtual IPs for the controller role
    OS::TripleO::Controller::Ports::RedisVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/vip.yaml

parameter_defaults:
  # Customize all these values to match the local environment
  InternalApiNetCidr: 172.17.0.0/24
  TenantNetCidr: 172.16.0.0/24
  ExternalNetCidr: 10.35.184.144/28
  # CIDR subnet mask length for provisioning network
  ControlPlaneSubnetCidr: '24'
  InternalApiAllocationPools: [{'start': '172.17.0.10', 'end':
'172.17.0.200'}]
  TenantAllocationPools: [{'start': '172.16.0.100', 'end':
'172.16.0.200'}]
  # Use an External allocation pool which will leave room for floating
IPs
  ExternalAllocationPools: [{'start': '10.35.184.146', 'end':
'10.35.184.157'}]
```

```
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.35.184.158
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.5.254
  # Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.5.1
  InternalApiNetworkVlanID: 600
  TenantNetworkVlanID: 603
  ExternalNetworkVlanID: 411
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["10.35.28.28","8.8.8.8"]
  # May set to br-ex if using floating IPs only on native VLAN on
bridge br-ex
  NeutronExternalNetworkBridge: "''"
  # The tunnel type for the tenant network (vxlan or gre). Set to '' to
disable tunneling.
  NeutronTunnelTypes: ''
  # The tenant network type for Neutron (vlan or vxlan).
  NeutronNetworkType: 'vxlan'
  # The OVS logical->physical bridge mappings to use.
  # NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-isolated'
  # The Neutron ML2 and OpenVSwitch vlan mapping range to support.
  NeutronNetworkVLANRanges: 'datacentre:412:412'
  # Nova flavor to use.
  OvercloudControlFlavor: baremetal
  OvercloudComputeFlavor: baremetal
  # Number of nodes to deploy.
  ControllerCount: 3
  ComputeCount: 2

  # Sets overcloud nodes custom names
  # http://docs.openstack.org/developer/tripleo-
docs/advanced_deployment/node_placement.html#custom-hostnames
  ControllerHostnameFormat: 'controller-%index%'
  ComputeHostnameFormat: 'compute-%index%'
  CephStorageHostnameFormat: 'ceph-%index%'
  ObjectStorageHostnameFormat: 'swift-%index%'
```

7.5.4. The *controller.yaml* file

The file is located in the **/home/stack/baremetal-vlan/nic-configs/** directory.

In this example, you are defining two switches: *br-isolated* and *br-ex*. **nic2** will be under *br-isolated* and **nic3** under *br-ex*:

```
heat_template_version: 2015-04-30

description: >
  Software Config to drive os-net-config to configure VLANs for the
  controller role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
```

```
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  ExternalNetworkVlanID:
    default: ''
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: ''
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ControlPlaneSubnetCidr: # Override this via parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations)
that will be added to resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this via parameter_defaults
```

```
    description: The IP address of the EC2 metadata server.
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                -
                  ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
              members:
                -
                  type: interface
                  name: nic2
                  # force the MAC address of the bridge to this
interface
                  primary: true
                -
                  type: vlan
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: InternalApiIpSubnet}
                -
                  type: vlan
                  vlan_id: {get_param: TenantNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: TenantIpSubnet}
            -
              type: ovs_bridge
              name: br-ex
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              members:
                -
                  type: interface
                  name: nic3
                  # force the MAC address of the bridge to this
```

```
interface
                -
                  type: vlan
                  vlan_id: {get_param: ExternalNetworkVlanID}
                  addresses:
                  -
                    ip_netmask: {get_param: ExternalIpSubnet}
                  routes:
                    -
                      default: true
                      next_hop: {get_param:
ExternalInterfaceDefaultRoute}

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

7.5.5. The *compute.yaml* file

The file is located in the **/home/stack/baremetal-vlan/nic-configs/** directory.

Most of the options in the Compute configuration is the same as the controller. In this example, **nic3** is under **br-ex** to be used for External connectivity (Floating IP network)

```
heat_template_version: 2015-04-30

description: >
  Software Config to drive os-net-config to configure VLANs for the
  Compute role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  InternalApiNetworkVlanID:
    default: ''
```

```
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  ControlPlaneSubnetCidr: # Override this via parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this via parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations)
that will be added to resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this via parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
               -
                 ip_netmask:
                   list_join:
                     - '/'
```

```
                    - - {get_param: ControlPlaneIp}
                      - {get_param: ControlPlaneSubnetCidr}
              routes:
               -
                 ip_netmask: 169.254.169.254/32
                 next_hop: {get_param: EC2MetadataIp}
               -
                 next_hop: {get_param: ControlPlaneDefaultRoute}
              members:
                -
                  type: interface
                  name: nic2
                  # force the MAC address of the bridge to this
interface
                  primary: true
                -
                  type: vlan
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: InternalApiIpSubnet}
                -
                  type: vlan
                  vlan_id: {get_param: TenantNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: TenantIpSubnet}
            -
              type: ovs_bridge
              name: br-ex
              use_dhcp: false
              members:
                -
                  type: interface
                  name: nic3

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

7.6. DIRECTOR CONFIGURATION FILES REFERENCES

7.6.1. The *neutron.conf* file

This file is located in the **`/etc/neutron/`** directory and should contain the following information.

```
[DEFAULT]
service_plugins=odl-router_v2
```

7.6.2. The *ml2_conf.ini* file

This file is located in the **/etc/neutron/plugins/ml2/** directory and should contain the following information:

```
[ml2]
type_drivers = vxlan,vlan,flat,gre
tenant_network_types = vxlan
mechanism_drivers = opendaylight_v2

[ml2_type_vlan]
network_vlan_ranges = datacentre:412:412

[ml2_odl]
password = admin
username = admin
url = http://172.17.1.18:8081/controller/nb/v2/neutron
```

- Under the **[ml2]** section note that *VXLAN* is used as the networks' type and so is the **opendaylight_v2** mechanism driver.
- Under **[ml2_type_vlan]**, the same mappings as configured in **network-environment.yaml** file, should be set.
- Under **[ml2_odl]**, you should see the configuration accessing the **OpenDaylightController**.

You can use the above details to access to check that the access to OpenDaylight controller works:

```
$ curl -H "Content-Type:application/json" -u admin:admin
http://172.17.1.18:8081/controller/nb/v2/neutron/networks
```

CHAPTER 8. WHERE CAN I FIND MORE INFORMATION ABOUT RED HAT OPENSTACK PLATFORM AND OPENDAYLIGHT?

Component	Reference
OpenDaylight	For further information that is not covered in this document, see the OpenDaylight Boron documentation.
Red Hat OpenDaylight Product Guide	For more information about the Red Hat OpenDaylight and its relation to the Red Hat OpenStack Platform, see the Red Hat OpenDaylight Product Guide.
Red Hat Enterprise Linux	Red Hat OpenStack Platform is supported on Red Hat Enterprise Linux 7.3. For information on installing Red Hat Enterprise Linux, see the corresponding installation guide at Red Hat Enterprise Linux Documentation Suite.
Red Hat OpenStack Platform	To install OpenStack components and their dependencies, use the Red Hat OpenStack Platform director. The director uses a basic OpenStack undercloud, which is then used to provision and manage the OpenStack nodes in the final overcloud. Be aware that you will need one extra host machine for the installation of the undercloud, in addition to the environment necessary for the deployed overcloud. For detailed instructions, see Director Installation and Usage. For information on configuring advanced features for a Red Hat OpenStack Platform enterprise environment using the Red Hat OpenStack Platform director such as network isolation, storage configuration, SSL communication, and general configuration method, see Advanced Overcloud Customization. You can also manually install the Red Hat OpenStack Platform components, see Manual Installation Procedures.
NFV Documentation	For more details on planning your Red Hat OpenStack Platform deployment with NFV, see Network Function Virtualization Planning Guide.