Red Hat

Red Hat OpenStack Platform 13

Red Hat OpenDaylight Installation and Configuration Guide

Install and Configure OpenDaylight using Red Hat OpenStack Platform

Last Updated: 2023-01-31

Red Hat OpenStack Platform 13 Red Hat OpenDaylight 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 13 with the OpenDaylight software-defined network (SDN) controller. The OpenDaylight controller is a drop-in replacement for the neutron **ML2/OVS** plug-in and for the **L2** and **L3** agents, and provides network virtualization within the Red Hat OpenStack environment.

CHAPTER 1. OVERVIEW

1.1. WHAT IS OPENDAYLIGHT?

The OpenDaylight platform is a programmable SDN controller written in Java that you can use 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 need to use the OVS agent on every node. OpenDaylight can program each OVS instance across the environment directly, without needing any agents on individual nodes. 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 provides 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 virtual machine (VM) instance metadata (**cloud-init**), and therefore Red Hat recommends you deploy the neutron DHCP agent rather than rely on OpenDaylight for such functionality.

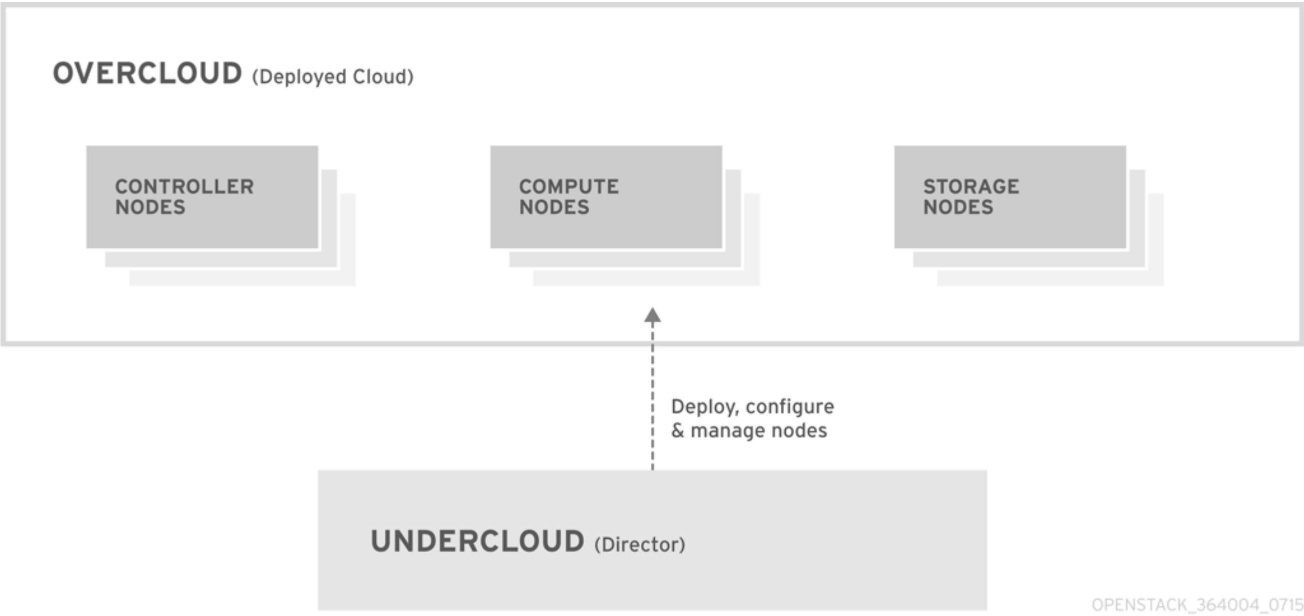
1.3. WHAT IS RED HAT OPENSTACK PLATFORM DIRECTOR AND HOW IS IT DESIGNED?

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) project.

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 operate 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 Director Installation and Usage.

Figure 1.1. Red Hat OpenStack Platform director – undercloud and overcloud



1.3.1. Red Hat OpenStack Platform director and OpenDaylight

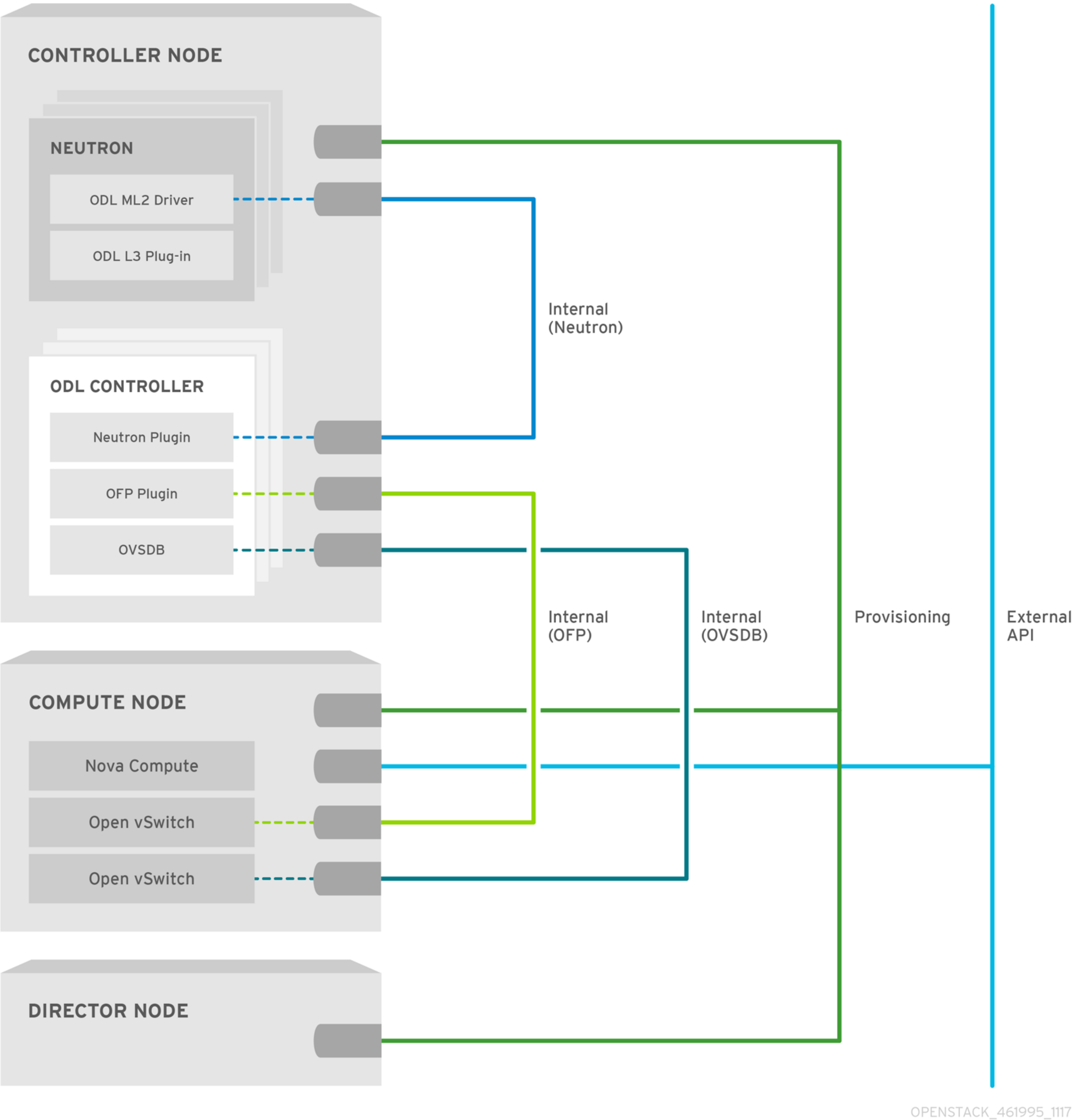
Red Hat OpenStack Platform director introduces the concept of composable services and custom roles. Composable services form isolated resources, that can be included and enabled per role, when they are needed. Custom roles enable users to create their own roles, independent from the default Controller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

Two services have been added to integrate OpenDaylight with director:

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

By default, the **OpenDaylightApi** service runs on the Controller role, while the **OpenDaylightOvs** service runs on Controller and Compute roles. OpenDaylight offers *High Availability* (HA) by scaling the number of **OpenDaylightApi** service instances. By default, scaling the number of Controllers to three or more automatically enables *HA*. For more information on the OpenDaylight *HA* architecture, see High Availability and Clustering with 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 can configure individual services to specific, predefined network types. These network traffic types include:

IPMI	The power management network for nodes. You must configure this network before you install 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. You must configure the network before you install 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 tunnelling 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.

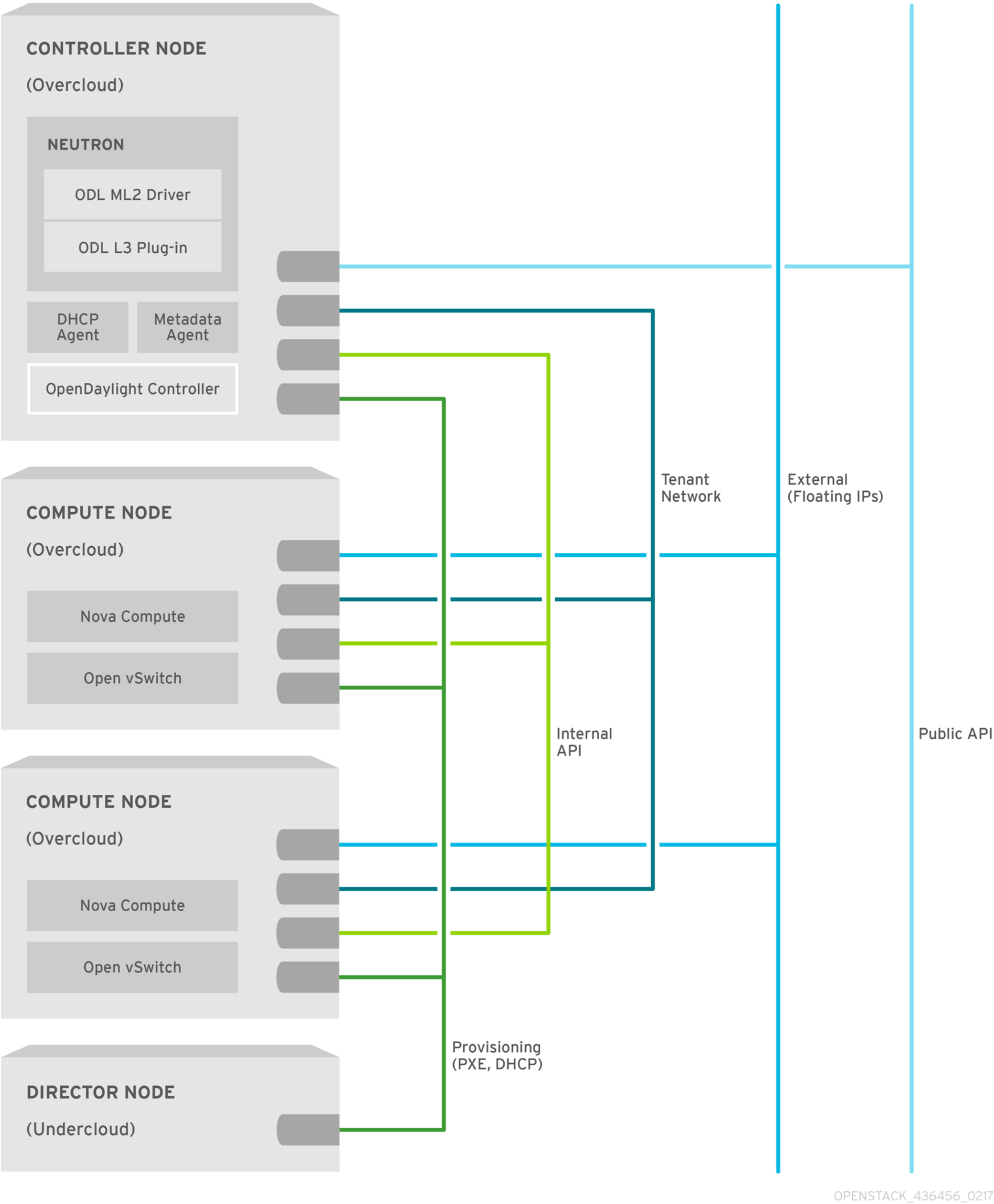
Storage (storage)	Block Storage, NFS, iSCSI, and others. Ideally, this would be isolated to an entirely separate switch fabric for performance optimization.
Storage Management (storage_mgmt)	OpenStack Object Storage (swift) uses this network to synchronize data objects between participating the replica 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* back-end connect over the Storage Management Network, because they do not interact with *Ceph* directly but rather use the front-end 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 one-to-one *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 nodes that are not controllers.

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* and *Tenant* 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 *datacentre*, 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 to enable OpenStack and OpenDaylight service traffic.

By default, OpenDaylight Northbound uses the *8080* port. In order not to conflict with the swift service, that also uses the *8080* port, the OpenDaylight ports are set to *8081* 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.

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	Internal API
OpenDaylight Southbound: OVSDB	TCP	6640	Internal API
OpenDaylight Southbound: OpenFlow	TCP	6653	Internal API
OpenDaylight High Availability	TCP	2550	Internal API
VXLAN	UDP	4789	Tenant

Table 1: Network and Firewall configuration

NOTE

This 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 Firewall Rules for Red Hat OpenStack Platform guide.

CHAPTER 2. WHAT DO YOU NEED TO RUN OPENDAYLIGHT?

The following section contains information about the deployment requirements for the overcloud with OpenDaylight. You must have sufficient computer resources available to correctly install and run Red Hat OpenDaylight. Use the following information to understand the minimum requirements.

2.1. COMPUTE NODE REQUIREMENTS

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

Processor	64-bit 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 that this processor has a minimum of 4 cores.
Memory	A minimum of 6 GB of RAM. Add additional RAM to this requirement according to 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 Network Interface Cards (NICs) in a production environment. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic. For more information about NICs, see Tested NICs.
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.

2.2. 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	A 64-bit 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 must have 72 GB of RAM. **Controller RAM recommended calculation:** Use 3 GB of memory per core. For example, a machine with 48 cores must have 144 GB of RAM. For more information about 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

This document focuses only on OpenDaylight installation. Before you can deploy OpenDaylight, you must ensure 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 procedures necessary to deploy the undercloud and overcloud.

There are several methods to install OpenDaylight in Red Hat OpenStack platform. The following chapter introduces the most useful scenarios of OpenDaylight and how to install them.

3.1. UNDERSTAND DEFAULT CONFIGURATION AND CUSTOMIZING SETTINGS

The recommended approach to installing OpenDaylight is to use the default environment file **neutron-opendaylight.yaml** and pass it as an argument to the deployment command on the undercloud. This deploys the default installation of OpenDaylight.

Other OpenDaylight installation and configuration scenarios are based on this installation method. You can deploy OpenDaylight with various different scenarios by providing specific environment files to the deployment command.

3.1.1. Understanding the default environment file

The default environment file is **neutron-opendaylight.yaml** in the **/usr/share/openstack-tripleo-heat-templates/environments/services** directory. This environment file enables or disables services that the OpenDaylight supports. The environment file also defines necessary parameters that the director sets during deployment.

The following file is an example **neutron-opendaylight.yaml** file that you can use for a Docker based deployment:

```
# A Heat environment that can be used to deploy OpenDaylight with L3 DVR using Docker containers
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: ../../docker/services/opendaylight-api.yaml
  OS::TripleO::Services::OpenDaylightOvs: ../../puppet/services/opendaylight-ovs.yaml
  OS::TripleO::Services::NeutronL3Agent: OS::Heat::None
  OS::TripleO::Docker::NeutronMl2PluginBase: ../../puppet/services/neutron-plugin-ml2-odl.yaml

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronPluginExtensions: 'port_security'
  NeutronMechanismDrivers: 'opendaylight_v2'
  NeutronServicePlugins: 'odl-router_v2,trunk'
  OpenDaylightLogMechanism: 'console'
```

Red Hat OpenStack Platform director uses the **resource_registry** to map resources for a deployment to the corresponding resource definition yaml file. Services are one type of resource that you can map. If you want to disable a particular service, set the value **OS::Heat::None**. In the default file, the

OpenDaylightApi and **OpenDaylightOvs** services are enabled, while default neutron agents are explicitly disabled as OpenDaylight inherits their functionality.

You can use heat parameters to configure settings for a deployment with director. You can override their default values with the **parameter_defaults** section of the environment file.

In this example, the **NeutronEnableForceMetadata**, **NeutronMechanismDrivers**, and **NeutronServicePlugins** parameters are set to enable OpenDaylight.

NOTE

The list of other services and their configuration options are provided later in this guide.

3.1.2. Configuring the OpenDaylight API Service

You can change the default values in the **/usr/share/openstack-tripleo-heat-templates/puppet/services/opendaylight-api.yaml** file to suit your needs. Do not overwrite the settings in this file directly. Duplicate this file and retain the original as a backup solution. Only modify the duplicate and pass the duplicate to the deployment command.

NOTE

The parameters in the latter environment files override those set in previous environment files. Ensure that you pay attention to the order of the environment files to avoid overwriting parameters accidentally.

3.1.2.1. Configurable Options

When you configure the OpenDaylight **API Service**, you can set several parameters:

OpenDaylightPort	The port used for Northbound communication. The default value is **0**. This parameter is deprecated in OSP 13.
OpenDaylightUsername	The login user name for OpenDaylight. The default value is **admin**.
OpenDaylightPassword	The login password for OpenDaylight. The default value is **admin**.
OpenDaylightEnableDHCP	Enables OpenDaylight to act as the DHCP service. The default value is **false**.
OpenDaylightFeatures	A comma-delimited list of features to boot in OpenDaylight. The default value is **[odl-netvirt-openstack, odl-jolokia]**.
OpenDaylightConnectionProtocol	The L7 protocol used for REST access. The default value is **http**.
OpenDaylightManageRepositories	Defines whether to manage the OpenDaylight repository. The default value is **false**.
OpenDaylightSNATMechanism	The SNAT mechanism to be used by OpenDaylight. Select **conntrack** or **controller**. The default value is **conntrack**.

OpenDaylightLogMechanism	The logging mechanism for OpenDaylight. Select **file** or **console**. The default value is **file**.
OpenDaylightTLSKeystorePassword	The password for the OpenDaylight TLS keystore. The default value is **opendaylight**. Passwords must be at least 6 characters.
EnableInternalTLS	Enables or disables TLS in the internal network. You can use values **true** or **false**. The default value is **false**.
InternalTLSCAFile	If you enable TLS for services in the internal network, you must use the **InternalTLSCAFile** parameter to specify the default CA cert. The default value is **/etc/ipa/ca.crt**.

For more information on how to deploy with TLS, see the Advanced Overcloud Customization Guide.

3.1.3. Configuring the OpenDaylight OVS Service

You can change the default values in the **/usr/share/openstack-tripleo-heat-templates/puppet/services/opendaylight-ovs.yaml** file to suit your needs. Do not overwrite the settings in this file directly. Duplicate this file and retain the original as a backup solution. Modify only the duplicate and pass the duplicate to the deployment command.

NOTE

The parameters in the latter environment files override those set in previous environment files. Ensure that you pay attention to the order of the environment files to avoid overwriting parameters accidentally.

3.1.3.1. Configurable options

When you configure the OpenDaylight **OVS Service**, you can set several parameters:

OpenDaylightPort	The port used for Northbound communication to OpenDaylight. The default value is **0**. The OVS Service uses the Northbound to query OpenDaylight to ensure that it is fully up before connecting. This parameter is deprecated in OSP 13.
OpenDaylightConnectionProtocol	The Layer 7 protocol used for REST access. The default value is **http**. **http** is the only supported protocol in OpenDaylight. This parameter is deprecated in OSP 13.
OpenDaylightCheckURL	The URL to verify OpenDaylight is fully up before OVS connects. The default value is **restconf/operational/network-topology:network-topology/topology/netvirt:1**
OpenDaylightProviderMappings	A comma-delimited list of mappings between logical networks and physical interfaces. This setting is required for VLAN deployments. The default value is **datacentre:br-ex**.

OpenDaylightUsername	The custom username for the OpenDaylight OVS service. The default value is **admin**.
OpenDaylightPassword	The custom password for the OpenDaylight OVS service. The default value is **admin**.
HostAllowedNetworkTypes	Defines the allowed tenant network types for this OVS host. They can vary per host or role to constrain the hosts that nova instances and networks are scheduled to. The default value is **['local', 'vlan', 'vxlan', 'gre', 'flat']**.
OvsEnableDpdk	Enable or disable DPDK in OVS. The default values is **false**.
OvsVhostuserMode	The mode for OVS with vhostuser port creation. In client mode, the hypervisor is responsible for creating vhostuser sockets. In server mode, OVS creates them. The default value is **client**.
VhostuserSocketDir	The directory to use for vhostuser sockets. The default value is **/var/run/openvswitch**.
OvsHwOffload	Enables or disables OVS Hardware Offload. You can use **true** or **false**. The default value is **false**. This parameter is in technical preview for this release.
EnableInternalTLS	Enables or disables TLS in the internal network. You can use values **true** or **false**. The default value is **false**.
InternalTLSCAFile	If you enable TLS for services in the internal network, you must use the **InternalTLSCAFile** parameter to specify the default CA cert. The default value is **/etc/ipa/ca.crt**.
ODLUpdateLevel	The OpenDaylight update level. You can use values **1** or **2**. The default value is **1**.
VhostuserSocketGroup	The vhost-user socket directory group. When vhostuser is in the default **dpdkvhostuserclient** mode, qemu creates the vhost socket. The default value for **VhostuserSocketGroup** is **qemu**.
VhostuserSocketUser	The vhost-user socket directory user name. When vhostuser is in the default **dpdkvhostuserclient** mode, qemu creates the vhost socket. The default value for **VhostuserSocketUser** is **qemu**.

3.1.4. 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. This functionality is enabled in the default environment file **neutron-opendaylight.yaml**. To disable it, set the **NeutronEnableForceMetadata** to **false**.

VM instances 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** go to the Metadata name server 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.

3.1.5. Understanding the network configuration and NIC template

In Red Hat OpenStack Platform director, the physical neutron network datacenter 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 shown 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}
```

With the DPDK, you must create another OVS bridge, typically called **br-phy**, and provide it with the ovs-dpdk-port. The IP address of the bridge is configured for VXLAN overlay network tunnels.

```
type: ovs_user_bridge
      name: br-phy
      use_dhcp: false
      addresses:
          -
              ip_netmask: {get_param: TenantIpSubnet}
              members:
                  -
                      type: ovs_dpdk_port
```

```
            name: dpdk0
            driver: uio_pci_generic
            members:
              -
                type: interface
                name: nic1
                # force the MAC address of the bridge to this interface
                primary: true
```

> **NOTE**
>
> When using network isolation, you do not need to place an IP address, or a default route, in this bridge on Compute nodes.

Alternatively, you can configure external network access without using the **br-ex** bridge. To use this 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 you can use it to specify an interface in the NIC template for the Compute node.

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

3.2. BASIC INSTALLATION OF OPENDAYLIGHT

This section shows how to deploy OpenDaylight with the standard environment files.

3.2.1. Prepare the OpenDaylight environment files for overcloud

Before you start

- Install the undercloud. For more information, see Installing the undercloud.
- Optionally, create a local registry with the container images that you want to use during the overcloud and OpenDaylight installation. For more information, see Configuring a container image source in the *Director installation and usage* guide.

Procedure

1. Log in to the undercloud and load the admin credentials.

   ```
   $ source ~/stackrc
   ```

2. Create a Docker registry file **odl-images.yaml** that contains references to the Docker container images that you need for the OpenStack and OpenDaylight installation.

   ```
   $ openstack overcloud container image prepare -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml --output-env-file /home/stack/templates/odl-images.yaml
   ```

You now successfully prepared the environment to deploy overcloud and you are ready to start the installation described in Section 3.2.2, "Install overcloud with OpenDaylight" .

More information

The **openstack overcloud image prepare** command prepares the container images environment files for the installation of overcloud and OpenDaylight. This command uses the following options:

-e
: specifies the service environment file to add specific container images required by that environment, such as OpenDaylight and OVS

--env-file
: creates a new container image environment file with a list of container images to use for the installation

--pull-source
: sets the location of the Docker containers registry

--namespace
: sets the version of the Docker containers

--prefix
: adds a prefix to the image name

--suffix
: adds a suffix to the image name

--tag
: defines the release of the images

3.2.2. Install overcloud with OpenDaylight

Before you start

- Follow the Prepare the OpenDaylight environment files for overcloud procedure to create the necessary environment files for the deployment.

Procedure

1. Log in to the undercloud and load the admin credentials.

```
$ source ~/stackrc
```

2. Deploy the overcloud using previously created environment files.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates \
 -e <other environment files>
 -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-
opendaylight.yaml \
 -e /home/stack/templates/odl-images.yaml
```

NOTE

Environment files present in the deployment command overwrite environment files that you include earlier in the command. You must pay attention to the order of the environment files that you include to avoid overwriting parameters accidentally.

TIP

You can override some of the parameters by creating a minimal environment file that sets only the parameters that you want to change and combining it with the default environment files.

More information

The **openstack overcloud deploy** command in this procedure uses the following options:

--templates
: defines the path to the heat templates directory

-e
: specifies an environment file

3.3. INSTALL OPENDAYLIGHT IN CUSTOM ROLE

Installing OpenDaylight in a Custom role results in an isolated **OpenDaylightApi** service that runs on a designated OpenDaylight node, different from the Controller node.

If you want to use a Custom role for OpenDaylight, you must create a role file that contains node layout and function configuration.

3.3.1. Customize the role file based on default roles

You can deploy OpenStack with a user-defined list of roles, each role running a user-defined list of services. A role is a group of nodes that contains individual services or configurations. For example, you can create a *Controller* role that contains the **nova API** service. You can view example roles in **openstack-tripleo-heat-templates**.

Use these roles to generate a **roles_data.yaml** file that contains the roles that you want for the overcloud nodes. You can also create custom roles by creating individual files in a directory and use them to generate a new **roles_data.yaml**.

To create customized environment files that install only specific OpenStack roles, complete the following steps:

Procedure

- Load the admin credentials.

```
$ source ~/stackrc
```

- List the default roles that you can use to generate the custom **roles_data.yaml** file.

```
$ openstack overcloud role list
```

- If you want to use all of these roles, run the following command to generate a **roles_data.yaml** file:

```
$ openstack overcloud roles generate -o roles_data.yaml
```

- If you want to customize the role file to include only some of the roles, you can pass the names of the roles as arguments to the command in the previous step. For example, to create a **roles_data.yaml** file with the **Controller**, **Compute** and **Telemetry** roles, run the following command:

```
$ openstack overcloud roles generate - roles_data.yaml Controller Compute Telemetry
```

3.3.2. Create a custom role for OpenDaylight

To create a custom role, create a new role file in the role files directory and generate a new **roles_data.yaml** file. For each custom role that you create, you must create a new role file. Each custom role file must include the data only for a specific role, and the custom role file name must match the role name.

Minimally, the file must define these parameters:

- **Name:** defines the name of the role. The name must always be a non-empty unique string.

```
- Name: Custom_role
```

- **ServicesDefault:** lists the services used in this role. The variable can remain empty, if there are no services used. The example format looks like this:

```
ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
```

In addition to the required parameters, you can also define further settings:

- **CountDefault:** defines the default number of nodes. If **CountDefault:** is empty, it defaults to zero.

```
CountDefault: 1
```

- **HostnameFormatDefault:** defines the format string for a host name. The value is optional.

```
HostnameFormatDefault: '%stackname%-computeovsdpdk-%index%'
```

- **Description:** describes and adds information about the role.

```
Description:
    Compute OvS DPDK Role
```

Procedure

1. Copy the default role files into a new directory and keep the original files as a backup.

```
$ mkdir ~/roles
$ cp /usr/share/openstack-tripleo-heat-templates/roles/* ~/roles
```

2. Modify the default Controller role in the **Controller.yaml** file in **~/roles** and remove the **OpenDaylightApi** line from the file to disable the **OpenDaylightAPI** service on the Controller node:

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

3. Create a new **OpenDaylight.yaml** file in the **~/roles** directory and add the OpenDaylight role description:

```
- name: OpenDaylight
   CountDefault: 1
   ServicesDefault:
     - OS::TripleO::Services::Aide
     - OS::TripleO::Services::AuditD
     - OS::TripleO::Services::CACerts
     - OS::TripleO::Services::CertmongerUser
     - OS::TripleO::Services::Collectd
     - OS::TripleO::Services::Docker
     - OS::TripleO::Services::Fluentd
     - OS::TripleO::Services::Ipsec
     - OS::TripleO::Services::Kernel
     - OS::TripleO::Services::LoginDefs
     - OS::TripleO::Services::MySQLClient
     - OS::TripleO::Services::Ntp
     - OS::TripleO::Services::ContainersLogrotateCrond
     - OS::TripleO::Services::Rhsm
     - OS::TripleO::Services::RsyslogSidecar
     - OS::TripleO::Services::Securetty
     - OS::TripleO::Services::SensuClient
     - OS::TripleO::Services::Snmp
     - OS::TripleO::Services::Sshd
     - OS::TripleO::Services::Timezone
     - OS::TripleO::Services::TripleoFirewall
     - OS::TripleO::Services::TripleoPackages
     - OS::TripleO::Services::Tuned
     - OS::TripleO::Services::Ptp
     - OS::TripleO::Services::OpenDaylightApi
```

4. Save the file.

5. Generate the new role file to use when you deploy the OpenStack overcloud with OpenDaylight in the custom role.

```
$ openstack overcloud roles generate --roles-path ~/roles -o ~/roles_data.yaml Controller
Compute OpenDaylight
```

–

3.3.3. Install OverCloud with OpenDaylight in the custom role

Before you start

- Install the undercloud. For more information, see Installing the undercloud.
- Create environment files with links to overcloud container images. For more information, see Preparing the installation of overcloud with OpenDaylight.
- Prepare the role file to configure OpenDaylight in a custom role. For more information, see Create a custom role for OpenDaylight.

Procedure

1. Create a custom role. Set the following parameter values in the environment file:

```
- OvercloudOpenDaylightFlavor: opendaylight
- OvercloudOpenDaylightCount: 3
```

   For more information, see Creating a roles_data file.

2. Run the deployment command with the **-r** argument to override the default role definitions. This option tells the deployment command to use the **roles_data.yaml** file that contains your custom role. Pass the **odl-composable.yaml** environment file that you created in the previous step to this deployment command. In this example, there are three ironic nodes in total. One ironic node is reserved for the custom OpenDaylight role:

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e /usr/share/openstack-tripleo-heat-templates/environments/docker.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-
opendaylight.yaml
-e network-environment.yaml --compute-scale 1 --ntp-server 0.se.pool.ntp.org --control-
flavor control --compute-flavor compute -r ~/roles_data.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
-e /home/stack/templates/odl-composable.yaml
```

NOTE

Environment files present in the deployment command overwrite environment files that you include earlier in the command. You must pay attention to the order of the environment files that you include to avoid overwriting parameters accidentally.

TIP

You can override some of the parameters by creating a minimal environment file that sets only the parameters that you want to change and combining it with the default environment files.

More information

- The **-r** option overrides the role definitions at installation time.

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

- A custom role requires an extra ironic node during the installation.
- To override the node counter in the rhosp13 composable role for any custom role, use the syntax in this example: <role-name>Count: <value> The role name updates with accurate name details from role_data.yaml file.

3.3.4. Verify the installation of OpenDaylight in custom role

Before you start

- Install the Overcloud with OpenDaylight in the custom role. For more information, see Install Overcloud with OpenDaylight in custom role.

Procedure

1. List the existing instances:

```
$ openstack server list
```

2. Verify that the new OpenDaylight role is dedicated as an instance:

```
+--------------------------------------+--------------------------+--------+------------+-------------+--------------------+
| 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 |
+--------------------------------------+--------------------------+--------+------------+-------------+--------------------+
```

3.4. INSTALL OPENDAYLIGHT WITH SR-IOV SUPPORT

OpenDaylight might be deployed with Compute nodes that support *Single Root Input/Output Virtualization* (SR-IOV). In this deployment, Compute nodes must operate as dedicated SR-IOV nodes and must not be mixed with nova instances based on OVS. It is possible to deploy both OVS and SR-IOV Compute nodes in a single OpenDaylight deployment.

This scenario utilizes a custom SR-IOV Compute role to accomplish this kind of deployment.

The SR-IOV deployment requires that you use the neutron SR-IOV agent to configure the virtual functions (VFs). These functions are then passed to the Compute instance directly when it is deployed, and they serve as a network port. The VFs derive from a host NIC on the Compute node, and therefore some information about the host interface is required before you start the deployment.

3.4.1. Prepare the SR-IOV Compute role

Following the same methodology as shown in Install of OpenDaylight In Custom Role , you must create a custom role for the SR-IOV Compute nodes to allow creation of the SR-IOV based instances, while the default Compute role serves the OVS based nova instances.

Before you start

- Study the chapter Install of OpenDaylight In Custom Role

Procedure

1. Copy the default role files into a new directory and keep the original files as a backup.

```
$ mkdir ~/roles
$ cp /usr/share/openstack-tripleo-heat-templates/roles/* ~/roles
```

2. Create a new **ComputeSriov.yaml** file in the **~/roles** directory and add the following role description:

```
- name: ComputeSRIOV
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::Securetty
```

3. Save the file.

4. Remove the **NeutronSriovAgent** and **NeutronSriovHostConfig** services from the default Compute role and save the information in roles_data.yaml.

```
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NeutronSriovAgent
```

5. Generate the new role file to use to deploy the OpenStack overcloud with OpenDaylight Compute SR-IOV support.

```
$ openstack overcloud roles generate --roles-path ~/roles -o ~/roles_data.yaml Controller
Compute ComputeSriov
```

6. Create the local registry:

```
openstack overcloud container image prepare   --namespace=192.168.24.1:8787/rhosp13   -
-prefix=openstack-   --tag=2018-05-07.2
-e /home/stack/templates/environments/services-docker/neutron-opendaylight.yaml -e
```

```
/home/stack/templates/environments/services-docker/neutron-opendaylight-sriov.yaml --
output-env-file=/home/stack/templates/overcloud_images.yaml --roles-file
/home/stack/templates/roles_data.yaml
```

3.4.2. Configuring the SR-IOV agent service

To deploy OpenDaylight with the SR-IOV support, you must override the default parameters in the **neutron-opendaylight.yaml** file. You can use a standard SR-IOV environment file that resides in **/usr/share/openstack-tripleo-heat-templates** and the **neutron-opendaylight.yaml** environment file. However, it is a good practice not to edit the original files. Instead, duplicate the original environment file and modify the parameters in the duplicate file.

Alternatively, you can create a new environment file in which you provide only the parameters that you want to change, and use both files for deployment. To deploy the customized OpenDaylight, pass both files to the deployment command. Because newer environment files override any previous settings, you must include them in the deployment command in the correct order. The correct order is **neutron-opendaylight.yaml** first, and then **neutron-opendaylight-sriov.yaml**.

If you want to deploy OpenDaylight and SR-IOV with the default settings, you can use the **neutron-opendaylight-sriov.yaml** that is provided by Red Hat. If you need to change or add parameters, make a copy of the default SR-IOV environment file and edit the newly created file.

The following is an illustrative example of a customized **neutron-opendaylight-sriov.yaml** file:

```
# A Heat environment that can be used to deploy OpenDaylight with SRIOV
resource_registry:
  OS::TripleO::Services::NeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronCorePlugin: ../puppet/services/neutron-plugin-ml2.yaml
  OS::TripleO::Services::NeutronCorePlugin: ../puppet/services/neutron-plugin-ml2-odl.yaml
  OS::TripleO::Services::OpenDaylightApi: ../docker/services/opendaylight-api.yaml
  OS::TripleO::Services::OpenDaylightOvs: ../puppet/services/opendaylight-ovs.yaml
  OS::TripleO::Services::NeutronSriovAgent: ../puppet/services/neutron-sriov-agent.yaml
  OS::TripleO::Services::NeutronL3Agent: OS::Heat::None

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronPluginExtensions: 'port_security'
  NeutronMechanismDrivers: ['sriovnicswitch','opendaylight_v2']
  NeutronServicePlugins: 'odl-router_v2,trunk'

  # Add PciPassthroughFilter to the scheduler default filters
  #NovaSchedulerDefaultFilters:
['RetryFilter','AvailabilityZoneFilter','RamFilter','ComputeFilter','ComputeCapabilitiesFilter',
'ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter']

  #NovaSchedulerAvailableFilters:
["nova.scheduler.filters.all_filters","nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter"]

  #NeutronPhysicalDevMappings: "datacentre:ens20f2"

  # Number of VFs that needs to be configured for a physical interface
  #NeutronSriovNumVFs: "ens20f2:5"
```

```
#NovaPCIPassthrough:
#  - devname: "ens20f2"
#    physical_network: "datacentre"
```

More information

You can configure the following options in the **neutron-opendaylight-sriov.yaml** file. The table describes individual options and mentions the required settings to enable the SR-IOV functionality:

NovaSchedulerDefaultFilters	Allows the use of PCI Passthrough for SR-IOV. This must be uncommented in the environment file and include **PciPassthroughFilter**
NovaSchedulerAvailableFilters	Enables specifying PCI Passthrough Filter for Nova Default filters. Must be set and include **nova.scheduler.filters.all_filters**
NeutronPhysicalDevMappings	Maps the logical neutron network to a host network interface. This must be specified so that neutron is able to bind the virtual network to a physical port.
NeutronSriovNumVFs	Number of VFs to create for a host network interface. Syntax: **<Interface name>:<number of VFs>**

NovaPCIPassthrough	Configures the whitelist of allowed PCI devices in nova to be used for PCI Passthrough in a list format, for example:
	`NovaPCIPassthrough:`
	`  - vendor_id: "8086"`
	`    product_id: "154c"`
	`    address: "0000:05:00.0"`
	`    physical_network: "datacentre"`
	It can also simply use logical device name rather than specific hardware attributes:
	`NovaPCIPassthrough:`
	`  - devname: "ens20f2"`
	`    physical_network: "datacentre"`

3.4.3. Install OpenDaylight with SR-IOV

Before you start

- Install the undercloud. For more information, see Installing the undercloud.
- Create environment files with links to overcloud container images. For more information, see Preparing the installation of overcloud with OpenDaylight).

- Prepare the role file to configure OpenDaylight in a custom role with the SR-IOV support. For more information, see Prepare the SR-IOV compute role.

Procedure

1. Run the deployment command with the **-r** argument to include your custom role file and the necessary environment files to enable the SR-IOV functionality with OpenDaylight.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-sriov.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
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

NOTE

Environment files present in the deployment command overwrite environment files that you include earlier in the command. You must pay attention to the order of the environment files that you include to avoid overwriting parameters accidentally.

TIP

You can override some of the parameters by creating a minimal environment file that sets only the parameters that you want to change and combining it with the default environment files.

More information

- The **-r** option overrides the role definitions at installation time.

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

- A custom role requires an extra ironic node during the installation.

3.5. INSTALL OPENDAYLIGHT WITH OVS-DPDK SUPPORT

OpenDaylight might be deployed with *Open vSwitch Data Plane Development Kit* (DPDK) acceleration with director. This deployment offers higher dataplane performance as packets are processed in user space rather than in the kernel. Deploying with OVS-DPDK requires knowledge of the hardware physical layout for each Compute node to take advantage of potential performance gains.

You should consider especially:

- that the network interface on the host supports DPDK
- the NUMA node topology of the Compute node (number of sockets, CPU cores, and memory per socket)
- that the DPDK NIC PCI bus proximity to each NUMA node

- the amount of RAM available on the Compute node
- consulting the Network Functions Virtualization Planning and Configuration Guide .

3.5.1. Prepare the OVS-DPDK deployment files

To deploy OVS-DPDK, use a different environment file. The file will override some of the parameters set by the **neutron-opendaylight.yaml** environment file in the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker** directory. Do not modify the original environment file. Instead, create a new environment file that contains the necessary parameters, for example **neutron-opendaylight-dpdk.yaml**.

If you want to deploy OpenDaylight with OVS-DPDK with the default settings, use the default **neutron-opendaylight-dpdk.yaml** environment file in the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker** directory.

The default file contains the following values:

```
# A Heat environment that can be used to deploy OpenDaylight with L3 DVR and DPDK.
# This file is to be used with neutron-opendaylight.yaml

parameter_defaults:
  NovaSchedulerDefaultFilters:
"RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,NUMA
TopologyFilter"
  OpenDaylightSNATMechanism: 'controller'

  ComputeOvsDpdkParameters:
    OvsEnableDpdk: True

    ## Host configuration Parameters
    #TunedProfileName: "cpu-partitioning"
    #IsolCpusList: ""               # Logical CPUs list to be isolated from the host process (applied via cpu-
partitioning tuned).
                                    # It is mandatory to provide isolated cpus for tuned to achive optimal
performance.
                                    # Example: "3-8,12-15,18"
    #KernelArgs: ""                 # Space separated kernel args to configure hugepage and IOMMU.
                                    # Deploying DPDK requires enabling hugepages for the overcloud compute
nodes.
                                    # It also requires enabling IOMMU when using the VFIO (vfio-pci)
OvsDpdkDriverType.
                                    # This should be done by configuring parameters via host-config-and-
reboot.yaml environment file.

    ## Attempting to deploy DPDK without appropriate values for the below parameters may lead to
unstable deployments
    ## due to CPU contention of DPDK PMD threads.
    ## It is highly recommended to to enable isolcpus (via KernelArgs) on compute overcloud nodes
and set the following parameters:
    #OvsDpdkSocketMemory: ""       # Sets the amount of hugepage memory to assign per NUMA
node.
                                   # It is recommended to use the socket closest to the PCIe slot used for the
                                   # desired DPDK NIC.  Format should be comma separated per socket string
such as:
                                   # "<socket 0 mem MB>,<socket 1 mem MB>", for example: "1024,0".
```

```
    #OvsDpdkDriverType: "vfio-pci" # Ensure the Overcloud NIC to be used for DPDK supports this
UIO/PMD driver.
    #OvsPmdCoreList: ""             # List or range of CPU cores for PMD threads to be pinned to.  Note,
NIC
                                    # location to cores on socket, number of hyper-threaded logical cores, and
                                    # desired number of PMD threads can all play a role in configuring this setting.
                                    # These cores should be on the same socket where OvsDpdkSocketMemory is
assigned.
                                    # If using hyperthreading then specify both logical cores that would equal the
                                    # physical core.  Also, specifying more than one core will trigger multiple PMD
                                    # threads to be spawned, which may improve dataplane performance.
    #NovaVcpuPinSet: ""             # Cores to pin Nova instances to.  For maximum performance, select
cores
                                    # on the same NUMA node(s) selected for previous settings.
```

3.5.2. Configuring the OVS-DPDK deployment

You can configure the OVS-DPDK service by changing the values in **neutron-opendaylight-dpdk.yaml**.

TunedProfileName	Enables pinning of IRQs in order to isolate them from the CPU cores to be used with OVS-DPDK. Default profile: **cpu-partitioning**
IsolCpusList	Specifies a list of CPU cores to prevent the kernel scheduler from using these cores that can instead be assigned and dedicated to OVS-DPDK. The format takes a comma separated list of individual or a range of cores, for example **1,2,3,4-8,10-12**
KernelArgs	Lists arguments to be passed to the kernel at boot time. For OVS-DPDK, it is required to enable **IOMMU** and **Hugepages**, for example: ---- **intel_iommu=on iommu=pt default_hugepagesz=1GB hugepagesz=1G hugepages=60** ---- Note the amount of RAM for specified is 60 GB for hugepages. It is important to consider the available amount of RAM on Compute nodes when setting this value.
OvsDpdkSocketMemory	Specifies the amount of hugepage memory (in MB) to assign to each NUMA node. For maximum performance, assign memory to the socket closest to the DPDK NIC. List format of memory per socket: ---- "<socket 0 mem MB>,<socket 1 mem MB>" ---- For example: "1024,0"
OvsDpdkDriverType	Specifies the UIO driver type to use with PMD threads. The DPDK NIC must support the driver specified. Red Hat OpenStack Platform deployments support the driver type **vfio-pci**. Red Hat OpenStack Platform deployments do not support UIO drivers, including **uio_pci_generic** and **igb_uio**.

OvsPmdCoreList	Lists single cores or ranges of cores for PMD threads to be pinned to. The cores specified here should be on the same NUMA node where memory was assigned with the **OvsDpdkSocketMemory** setting. If hyper-threading is being used, then specify the logical cores that would make up the physical core on the host.
OvsDpdkMemoryChannels	Specifies the number of memory channels per socket.
NovaVcpuPinSet	Cores to pin nova instances to with **libvirtd**. For best performance use cores on the same socket where OVS PMD Cores have been pinned to.

3.5.3. Install OpenDaylight with OVS-DPDK

Before you start

- Install the undercloud. For more information, see Installing the undercloud.
- Create environment files with links to overcloud container images. For more information, see Preparing the installation of overcloud with OpenDaylight.
- Prepare the role file to configure OpenDaylight in a custom role with the SR-IOV support. For more information, see Prepare the OVS-DPDK deployment files.

Procedure

1. Run the deployment command with the necessary environment files to enable the DPDK functionality with OpenDaylight.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-dpdk.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
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

> **NOTE**
>
> Environment files present in the deployment command overwrite environment files that you include earlier in the command. You must pay attention to the order of the environment files that you include to avoid overwriting parameters accidentally.

TIP

You can override some of the parameters by creating a minimal environment file that sets only the parameters that you want to change and combining it with the default environment files.

3.5.4. Example: Configuring OVS-DPDK with ODL and VXLAN tunnelling

This section describes an example configuration of OVS-DPDK with ODL and VXLAN tunnelling.

> **IMPORTANT**
>
> You must determine the best values for the OVS-DPDK parameters that you set in the **network-environment.yaml** file to optimize your OpenStack network for OVS-DPDK. See Deriving DPDK parameters with workflows for details.

3.5.4.1. Generating the ComputeOvsDpdk composable role

Generate **roles_data.yaml** for the **ComputeOvsDpdk** role.

```
# openstack overcloud roles generate --roles-path templates/openstack-tripleo-heat-templates/roles -
o roles_data.yaml Controller ComputeOvsDpdk
```

3.5.4.2. Configuring OVS-DPDK parameters

> **IMPORTANT**
>
> You must determine the best values for the OVS-DPDK parameters that you set in the **network-environment.yaml** file to optimize your OpenStack network for OVS-DPDK. See https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_planning_and_co dpdk-configure#proc_derive-dpdk for details.

1. Add the custom resources for OVS-DPDK under **resource_registry**:

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

2. Under **parameter_defaults**, set the tunnel type and the tenant type to **vxlan**:

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan'
```

3. Under **parameters_defaults**, set the bridge mappings:

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings: 'tenant:br-link0'
OpenDaylightProviderMappings: 'tenant:br-link0'
```

4. Under **parameter_defaults**, set the role-specific parameters for the **ComputeOvsDpdk** role:

```
  ##########################
  # OVS DPDK configuration #
  ##########################
  ComputeOvsDpdkParameters:
```

```
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
intel_iommu=on isolcpus=2-19,22-39"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList: "2-19,22-39"
    NovaVcpuPinSet: ['4-19,24-39']
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "4096,4096"
    OvsDpdkMemoryChannels: "4"
    OvsDpdkCoreList: "0,20,1,21"
    OvsPmdCoreList: "2,22,3,23"
    OvsEnableDpdk: true
```

NOTE

You must assign at least one CPU (with sibling thread) on each NUMA node with or without DPDK NICs present for DPDK PMD to avoid failures in creating guest instances.

NOTE

These huge pages are consumed by the virtual machines, and also by OVS-DPDK using the **OvsDpdkSocketMemory** parameter as shown in this procedure. The number of huge pages available for the virtual machines is the **boot** parameter minus the **OvsDpdkSocketMemory**.

You must also add **hw:mem_page_size=1GB** to the flavor you associate with the DPDK instance.

NOTE

OvsDPDKCoreList and **OvsDpdkMemoryChannels** are the **required** settings for this procedure. Attempting to deploy DPDK without appropriate values causes the deployment to fail or lead to unstable deployments.

3.5.4.3. Configuring the Controller node

1. Create the control plane Linux bond for an isolated network.

```
  - type: linux_bond
    name: bond_api
    bonding_options: "mode=active-backup"
    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: eth1
      primary: true
```

2. Assign VLANs to this Linux bond.

```
  - type: vlan
    vlan_id:
      get_param: InternalApiNetworkVlanID
    device: bond_api
    addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet

  - type: vlan
    vlan_id:
      get_param: StorageNetworkVlanID
    device: bond_api
    addresses:
    - ip_netmask:
        get_param: StorageIpSubnet

  - type: vlan
    vlan_id:
      get_param: StorageMgmtNetworkVlanID
    device: bond_api
    addresses:
    - ip_netmask:
        get_param: StorageMgmtIpSubnet

  - type: vlan
    vlan_id:
      get_param: ExternalNetworkVlanID
    device: bond_api
    addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
    routes:
    - default: true
      next_hop:
        get_param: ExternalInterfaceDefaultRoute
```

3. Create the OVS bridge for access to the floating IPs into cloud networks.

```
  - type: ovs_bridge
    name: br-link0
    use_dhcp: false
    mtu: 9000
    members:
    - type: interface
      name: eth2
      mtu: 9000
    - type: vlan
      vlan_id:
```

```
      get_param: TenantNetworkVlanID
    mtu: 9000
    addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
```

3.5.4.4. Configuring the Compute node for DPDK interfaces

Create the **compute-ovs-dpdk.yaml** file from the default **compute.yaml** file and make the following changes:

1. Create the control plane Linux bond for an isolated network.

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
  - type: interface
    name: nic7
    primary: true
  - type: interface
    name: nic8
```

2. Assign VLANs to this Linux bond.

```
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
```

3. Set a bridge with a DPDK port to link to the controller.

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_:
             get_param: TenantNetworkVlanID
```

```
addresses:
  - ip_netmask:
      get_param: TenantIpSubnet
members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    rx_queue: 2
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
          - type: interface
            name: nic3
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: interface
            name: nic4
```

NOTE

To include multiple DPDK devices, repeat the **type** code section for each DPDK device you want to add.

NOTE

When using OVS-DPDK, **all** bridges on the same Compute node should be of type **ovs_user_bridge**. The director may accept the configuration, but Red Hat OpenStack Platform does not support mixing **ovs_bridge** and **ovs_user_bridge** on the same node.

3.5.4.5. Deploying the overcloud

Run the **overcloud_deploy.sh** script to deploy the overcloud.

```
#!/bin/bash

 openstack overcloud deploy \
--templates \
-r /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-
opendaylight.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-
dpdk.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ovs-dpdk-permissions.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/docker-images.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/network-environment.yaml
```

3.6. INSTALL OPENDAYLIGHT WITH L2GW SUPPORT

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Layer 2 gateway services allow a tenant's virtual network to be bridged to a physical network. This integration enables users to access resources on a physical server through a layer 2 network connection rather than through a routed layer 3 connection. This means extending the layer 2 broadcast domain instead of going through L3 or Floating IPs.

3.6.1. Prepare L2GW deployment files

To deploy OpenDaylight with L2GW support, use the **neutron-l2gw-opendaylight.yaml** file in the **/usr/share/openstack-tripleo-heat-templates/environments** directory. If you need to change the settings in that file, do not modify the existing file. Instead, create a new copy of the environment file that contains the necessary parameters.

If you want to deploy OpenDaylight and L2GW with the default settings, you can use **neutron-l2gw-opendaylight.yaml** in the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker** directory.

The default file contains these values:

```
# A Heat environment file that can be used to deploy Neutron L2 Gateway service
#
# Currently there are only two service provider for Neutron L2 Gateway
# This file enables L2GW service with OpenDaylight as driver.
#
# - OpenDaylight:
L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default
resource_registry:
  OS::TripleO::Services::NeutronL2gwApi: ../../docker/services/neutron-l2gw-api.yaml

parameter_defaults:
  NeutronServicePlugins: "odl-router_v2,trunk,l2gw"
  L2gwServiceProvider:
['L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default']

  # Optional
  # L2gwServiceDefaultInterfaceName: "FortyGigE1/0/1"
  # L2gwServiceDefaultDeviceName: "Switch1"
  # L2gwServiceQuotaL2Gateway: 10
  # L2gwServicePeriodicMonitoringInterval: 5
```

3.6.2. Configuring OpenDaylight L2GW deployment

You can configure the service by changing the values in the **neutron-l2gw-opendaylight.yaml** file:

NeutronServicePlugins	Comma-separated list of service plugin entrypoints to be loaded from the **neutron.service_plugins** namespace. Defaults to **router**.

L2gwServiceProvider	Defines the provider that should be used to provide this service. Defaults to **L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default**
L2gwServiceDefaultInterfaceName	Sets the name of the default interface.
L2gwServiceDefaultDeviceName	Sets the name of the default device.
L2gwServiceQuotaL2Gateway	Specifies the service quota for the L2 gateway. Defaults to **10**.
L2gwServicePeriodicMonitoringInterval	Specifies the monitoring interval for the L2GW service.

3.6.3. Install OpenDaylight with L2GW

Before you start

- Install the undercloud. For more information, see Installing the undercloud.
- Create environment files with links to overcloud container images. For more information, see Preparing the installation of overcloud with OpenDaylight.
- Prepare the role file to configure OpenDaylight in a custom role with the SR-IOV support. For more information, see Prepare the L2GW deployment files.

Procedure

1. Run the deployment command with the necessary environment files to enable the L2GW functionality with OpenDaylight.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-
opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-l2gw-
opendaylight.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

> **NOTE**
>
> Environment files present in the deployment command overwrite environment files that you include earlier in the command. You must pay attention to the order of the environment files that you include to avoid overwriting parameters accidentally.

TIP

You can override some of the parameters by creating a minimal environment file that sets only the parameters that you want to change and combining it with the default environment files.

CHAPTER 4. TEST THE DEPLOYMENT

4.1. PERFORM A BASIC TEST

The basic test verifies that instances can ping each other. The test also checks the *Floating IP SSH* access. This example describes how you can perform this test from the undercloud.

This procedure requires that you follow a large number of individual steps. For convenience, the procedure is divided into smaller parts. However, you must follow all steps in the following 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.

4.1.1. Create a new network for testing

1. Source the credentials to access the overcloud:

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

2. Create an external neutron network to access the instance from outside the overcloud:

```
$ openstack network create --external --project service --external  --provider-network-type
flat --provider-physical-network datacentre external
```

3. Create the corresponding neutron subnet for the new external network that you create in the previous step:

```
$ openstack subnet create  --project service --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 that you want to use to create overcloud instances:

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

5. Upload the cirros image to 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 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 that you 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 **cirros1** and attach it to the **net_test** network and **SSH** security group:

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-id=$vlan1 --security-group
SSH --key-name RDO_KEY --availability-zone nova:overcloud-novacompute-0.localdomain
cirros1
```

11. Create a second instance called **cirros2**, also attached to the **net_test** network and **SSH** security group:

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-id=$vlan1 --security-group
SSH --key-name RDO_KEY --availability-zone nova:overcloud-novacompute-0.localdomain
cirros2
```

4.1.2. 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 default security group of the admin project:

```
$ 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
```

4.1.3. Test the connectivity

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

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

```
$ ip addr show
```

NOTE

You can also run the command **nova console-log <instance id>** from the undercloud to verify that the instance receives a DHCP address.

3. 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*.

4.1.4. 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 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:

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

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

```
$ openstack router add subnet test test
```

5. Find and store the floating IP that you create in Step 1:

```
$ 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 log in to the instance:

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

4.2. PERFORM ADVANCED TESTS

You can test several components of the OpenDaylight configuration and deployment after you deploy OpenDaylight. To test specific parts of the installation, you must follow several procedures. Each procedure is described separately.

You must perform the procedures on the **overcloud** nodes.

4.2.1. Connect to overcloud nodes

To connect to the overcloud nodes and ensure that they are operating correctly, complete the following steps:

Procedure

1. Log in to the undercloud.
2. Enter the following command to start the process:

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

3. List all instances:

```
$ openstack server list
```

4. Choose the required instance and note the instance IP address in the list.
5. Connect to the machine with the IP address from the list that you obtain in the previous step:

```
$ ssh heat-admin@<IP from step 4>
```

6. Switch to superuser:

```
$ sudo -i
```

4.2.2. Test OpenDaylight

To test that OpenDaylight is operating correctly, you must verify that the service is operational and that the specified features are loaded correctly.

Procedure

1. Log in to the overcloud node running OpenDaylight as a superuser, or to an OpenDaylight node running in a custom role.
2. Verify that the OpenDaylight Controller is running on all Controller nodes:

```
# docker ps | grep opendaylight
2363a99d514a        192.168.24.1:8787/rhosp13/openstack-opendaylight:latest
"kolla_start"           4 hours ago         Up 4 hours (healthy)                       opendaylight_api
```

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

```
# docker exec -it haproxy-bundle-docker-0 grep -A7 opendaylight /etc/haproxy/haproxy.cfg
listen opendaylight
```

```
  bind 172.17.0.10:8081 transparent
  bind 192.168.24.10:8081 transparent
  mode http
  balance source
  server overcloud-controller-0.internalapi.localdomain 172.17.0.22:8081 check fall 5 inter
2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.17.0.12:8081 check fall 5 inter
2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.17.0.13:8081 check fall 5 inter
2000 rise 2
```

4. Use HAproxy IP to connect the karaf account. The karaf password is **karaf**:

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

5. List the installed features.

```
# feature:list -i | grep odl-netvirt-openstack
```

If there is an **x** in the third column of the list, as generated during the procedure, then the feature is correctly installed.

6. Verify that the API is operational.

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

This API endpoint is set in **/etc/neutron/plugins/ml2/ml2_conf.ini** and used by the neutron to communicate with OpenDaylight.

7. Verify that *VXLAN* tunnels between the nodes are up.

```
# vxlan:show
```

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

```
# curl -u "admin:admin" http://localhost:8081/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:netty:eventexecutor"},
{"name" ...
```

9. List the REST streams that use the host internal_API IP.

```
# curl -u "admin:admin" http://localhost:8081/restconf/streams
```

You get a similar output:

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

10. Run the following command with host internal_API IP to verify that NetVirt is operational:

```
# curl -u "admin:admin" http://localhost:8081/restconf/operational/network-topology:network-topology/topology/netvirt:1
```

Note the following output to confirm that NetVirt is operational.

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

4.2.3. Test Open vSwitch

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

Procedure

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

```
# ovs-vsctl show
```

3. Note multiple Managers in the output. In this example, lines 2 and 3 display multiple Managers.

```
6b003705-48fc-4534-855f-344327d36f2a
    Manager "ptcp:6639:127.0.0.1"
    Manager "tcp:172.17.1.16:6640"
        is_connected: true
    Bridge br-ex
        fail_mode: standalone
        Port br-ex-int-patch
            Interface br-ex-int-patch
                type: patch
                options: {peer=br-ex-patch}
        Port br-ex
            Interface br-ex
                type: internal
        Port "eth2"
            Interface "eth2"
    Bridge br-isolated
        fail_mode: standalone
        Port "eth1"
            Interface "eth1"
        Port "vlan50"
            tag: 50
            Interface "vlan50"
                type: internal
        Port "vlan30"
            tag: 30
            Interface "vlan30"
                type: internal
        Port br-isolated
            Interface br-isolated
                type: internal
        Port "vlan20"
            tag: 20
```

```
            Interface "vlan20"
                type: internal
    Bridge br-int
        Controller "tcp:172.17.1.16:6653"
            is_connected: true
        fail_mode: secure
        Port br-ex-patch
            Interface br-ex-patch
                type: patch
                options: {peer=br-ex-int-patch}
        Port "tun02d236d8248"
            Interface "tun02d236d8248"
                type: vxlan
                options: {key=flow, local_ip="172.17.2.18", remote_ip="172.17.2.20"}
        Port br-int
            Interface br-int
                type: internal
        Port "tap1712898f-15"
            Interface "tap1712898f-15"
    ovs_version: "2.7.0"
```

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 corresponds to 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 that an OpenFlow protocol connection to OpenDaylight is established.

More information

- OpenDaylight creates the *br-int* bridge automatically.

4.2.4. 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         : "2.7.0"
 ssl                 : []
 statistics          : {}
 system_type         : RedHatEnterpriseServer
 system_version      : "7.4-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 in the **OpenDaylightProviderMappings** heat template parameter. This configuration maps the neutron logical networks to corresponding physical interfaces.

4.2.5. Verify neutron configuration

Procedure

1. Connect to the superuser account on one of the Controller role nodes.

2. Ensure that the **/etc/neutron/neutron.conf** file contains **service_plugins=odl-router_v2,trunk**.

3. Ensure that the **/etc/neutron/plugin.ini** file 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 the **admin_state_up** value for both the Metadata and DHCP agents are **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).
- 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 5. DEBUGGING

5.1. LOCATE THE LOGS

5.1.1. Access OpenDaylight logs

OpenDaylight stores logs in containers in the **/var/log/containers/opendaylight** directory. The most recent log is named **karaf.log**. OpenDaylight appends incremental numbering to previous logs, for example, **karaf.log.1**, **karaf.log.2**.

5.1.2. Access OpenStack Networking logs

When OpenStack networking commands fail, first examine the neutron logs. You can find the neutron logs in the **server.log** file on each neutron node in the **/var/log/containers/neutron** directory.

The **server.log** file also includes errors about the communication with OpenDaylight. If the neutron error originates from interacting with OpenDaylight, you must also examine the OpenDaylight logs to locate the cause of the failure.

5.2. DEBUG NETWORKING ERRORS

If you experience a network error such as loss of instance connectivity, but no errors are reported when issuing OpenStack commands or in the neutron logs, then it might be useful to inspect the OVS nodes for network traffic and OpenFlow flows:

1. Log in as **superuser** to the node where the network error occurs.
2. Display the information about the br-int switch.

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

3. Examine the output. It must 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 must 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**, note that there are three ports on this OVS node. The first port is a patch port going to the bridge br-ex, which is an External network connectivity port in this scenario. 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. The third port is a VXLAN tunnel port created for the tenant traffic.

- When you understand what each port is, you can examine the port statistics to verify that the port is receiving/sending traffic (see **Step 4**).

- From the output in **Step 5**, note that each port is receiving (rx pkts) and sending packets (tx pkts).

5.2.1. Advanced debugging using OpenFlow flows

For advanced users who are familiar with OpenFlow, you can examine the flows on the switch to detect where the traffic drops.

1. To list the flows, and to see how many packets have hit them, enter the following command:

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

2. Examine the output of the command to get 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
```

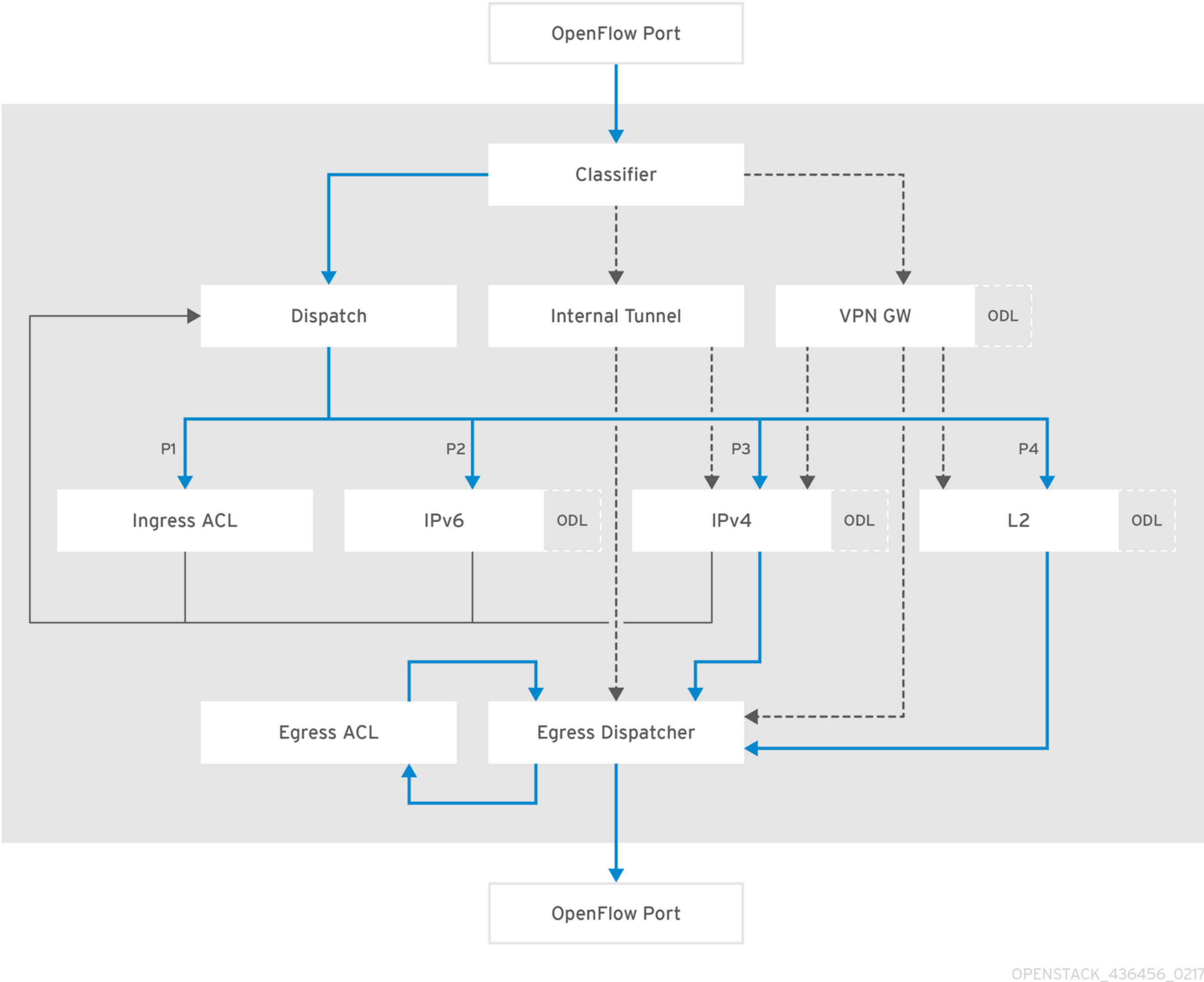
NOTE

This output has been edited for length.

5.2.2. 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 through 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 5.1. OpenDaylight NetVirt OpenFlow *Pipeline*



CHAPTER 6. DEPLOYMENT EXAMPLES

6.1. MODEL INSTALLATION SCENARIO USING TENANT NETWORK

In this section you explore an example of OpenDaylight installation using OpenStack in a production environment. This scenario uses tunneling (*VXLAN*) for tenant traffic separation.

6.1.1. Physical Topology

The topology of this scenario consists of six nodes:

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

6.1.2. 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 through 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

6.1.3. Planning NIC Connectivity

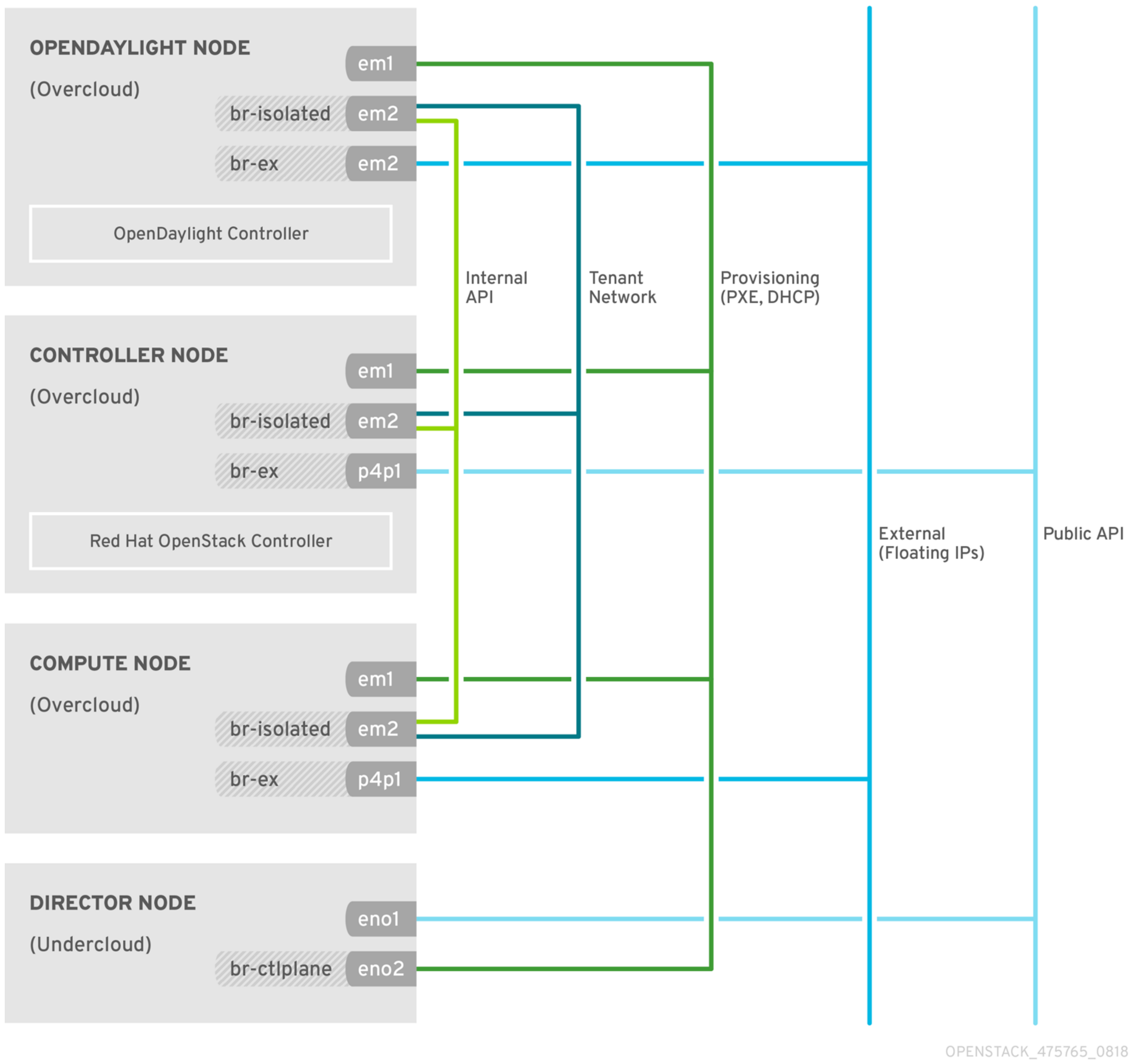
In this case, the environment files use abstracted numbered interfaces (**nic1**, **nic2**) and not the actual device names presented on the host operating system (like **eth0** or **eno2**). The hosts that belong to the same role do not require identical network interface device names. There is no problem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

The abstracted NIC scheme relies only on interfaces that are live and connected. In cases where the hosts have a different number of interfaces, it is sufficient 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.

6.1.4. Planning Networks, VLANs and IPs

This scenario uses network isolation to separate the Management, Provisioning, *Internal API*, Tenant, Public API, and Floating IPs network traffic. This graphic is an example network configuration. It shows custom role deployment. If required, you can also include OpenDaylight in the Red Hat OpenStack Platform conroller. This is the default setup. In this scheme IPMI network, NICs and routing are not shown. You might need additional networks depending on the OpenStack configuration.

Figure 6.1. 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

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 director also creates the *br-ex* bridge with the relevant network interface attached to it.

Ensure 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 *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

You can use the **TenantNetworkVlanID** attribute in the **network-environment.yaml** file to define a VLAN tag for the Tenant network when using *VXLAN* tunneling. For example, *VXLAN* tenant traffic transported over a VLAN tagged underlay network. This value can 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** can be used.

6.1.5. OpenDaylight configuration files used in this scenario

To deploy this scenario of OpenStack and OpenDaylight, the following deployment command was entered on the undercloud node:

```
$ 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.yaml \
  --log-file overcloud_install.log &> overcloud_install.log
```

Further, this guide will show the configuration files used in this scenario, their content, and it will also provide explanation on the setting used.

6.1.5.1. The extra_env.yaml file.

The 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 **datacenter** will be mapped to the **br-ex** OVS bridge and the tenant network traffic will be mapped to the **br-isolated** OVS bridge.

6.1.5.2. The undercloud.conf file

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

NOTE

The file path points to customized versions of the configuration files.

```
[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.

6.1.5.3. 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. The two files in the 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 VIP addresses
    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 IP addresses 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 IP addresses
  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 IP addresses 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%'
```

6.1.5.4. 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: pike

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 with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  DnsServers: # Override this with 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 with 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}
```

6.1.5.5. 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 are the same as in the Controller configuration. In this example, **nic3** is under **br-ex** to be used for External connectivity (Floating IP network)

```
heat_template_version: pike

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 with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
```

```
    type: string
  ControlPlaneDefaultRoute: # Override this with parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this with 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 with 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}
```

6.1.6. Red Hat OpenStack Platform director configuration files used in this scenario

6.1.6.1. 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,trunk
```

6.1.6.2. 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
```

1. Under the [ml2] section note that VXLAN is used as the networks' type and so is the opendaylight_v2 mechanism driver.

2. Under [ml2_type_vlan], the same mappings as configured in network-environment.yaml file, should be set.

3. Under [ml2_odl], you should see the configuration accessing the OpenDaylightController.

You can use these details to confirm access to the OpenDaylight Controller:

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

6.2. MODEL INSTALLATION SCENARIO USING PROVIDER NETWORKS

This installation scenario shows an example of OpenStack and OpenDaylight using provider networks instead of tenant networks. An external neutron provider network bridges VM instances to a physical network infrastructure that provides Layer-3 (L3) and other network services. In most cases, provider networks implement Layer-2 (L2) segmentation using the VLAN IDs. A provider network maps to a provider bridge on each Compute node that supports launching VM instances on the provider network.

6.2.1. Physical Topology

The topology of this scenario consists of six nodes:

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

6.2.2. Planning Physical Network Environment

The overcloud Controller nodes use four network interface cards (NICs) each:

Name	Purpose
nic1	Management network (e.g accessing the node through SSH)
nic2	Provisioning (PXE, DHCP), and *Internal API* networks
nic3	Tenant network
nic4	Public API network, Floating IP network

The overcloud Compute nodes are equipped with four NICs:

Name	Purpose
nic1	Management network
nic2	Provisioning, and *Internal API* networks
nic3	Tenant network
nic4	Floating IP network

The undercloud node is equipped with two NICs:

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

6.2.3. Planning NIC Connectivity

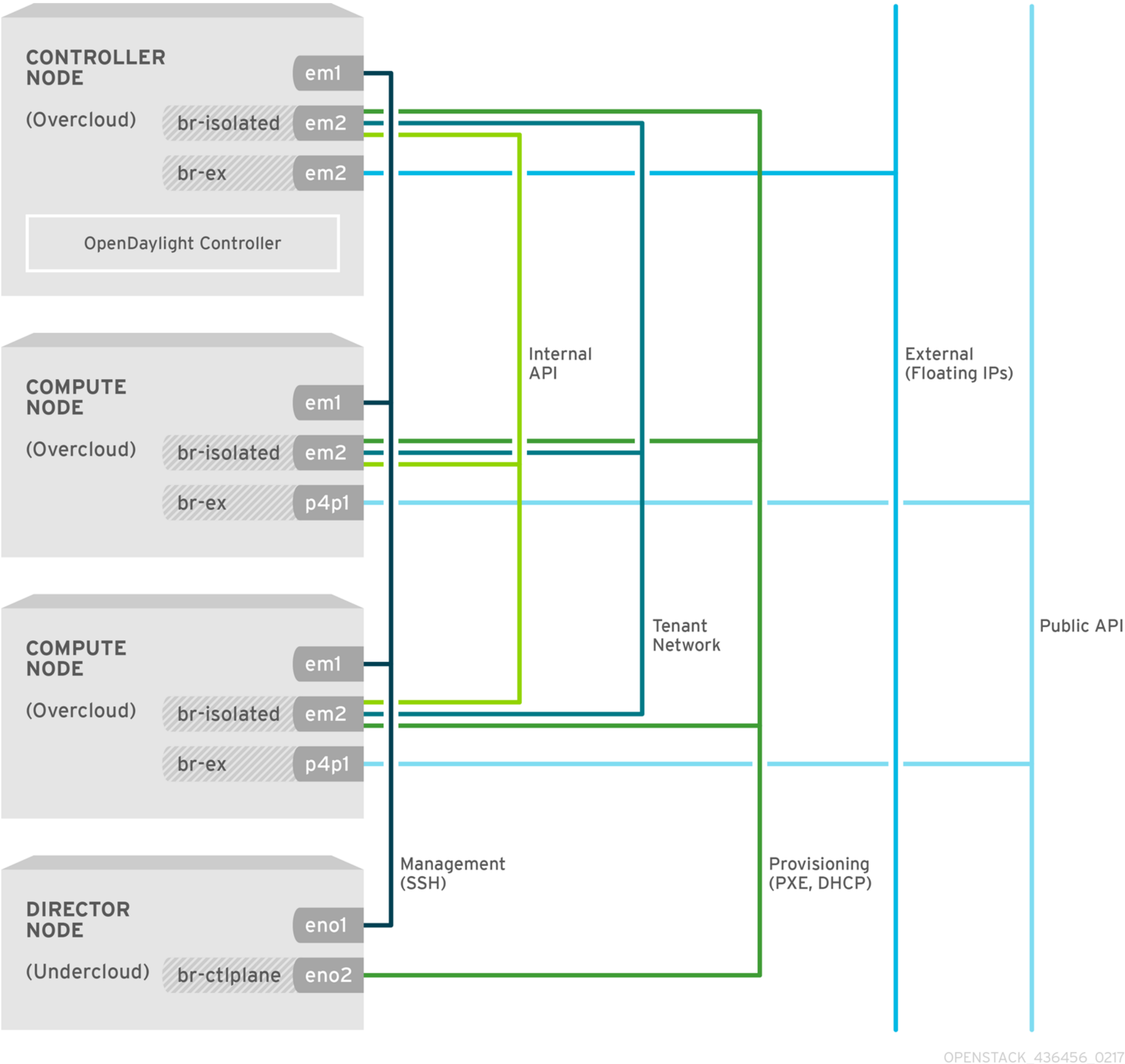
In this case, the environment files use abstracted numbered interfaces (**nic1**, **nic2**) and not the actual device names presented on the host operating system, for example, **eth0** or **eno2**. The hosts that belong to the same role do not require identical network interface device names. There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

The abstracted NIC scheme relies only on interfaces that are live and connected. In cases where the hosts have a different number of interfaces, it is sufficient 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.

6.2.4. Planning Networks, VLANs and IPs

This scenario uses network isolation to separate the Management, Provisioning, *Internal API*, Tenant, Public API, and Floating IPs network traffic.

Figure 6.2. 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	554,555-601	172.16.0.0/24
Public API	552	192.168.210.0/24
Floating IP	553	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 director also creates the **br-ex** bridge automatically with the relevant network interface attached to it.

Ensure that the 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 *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.

6.2.5. OpenDaylight configuration files used in this scenario

To deploy this scenario of OpenStack and OpenDaylight, the following deployment command was entered on the undercloud node:

```
$ 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.yaml \
  --log-file overcloud_install.log &> overcloud_install.log
```

This guide also shows the configuration files for this scenario, the configuration file content, and explanatory information about the configuration.

6.2.5.1. extra_env.yaml file.

The file has only one parameter.

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

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

6.2.5.2. undercloud.conf file

This file is in the **/home/stack/** directory.

> **NOTE**
>
> The file path points to customized versions of the configuration files.

```
[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
```

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

6.2.5.3. network-environment.yaml file

This is the main file for configuring the network. It is located in the **/home/stack/baremetal-vlan/** directory. The following file specifies the VLAN IDs and IP subnets for the different networks, and shows the provider mappings. The **controller.yaml** and **compute.yaml** files in the **nic-configs** directory are used to specify 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.
  OS::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 disabled, 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
  OS::TripleO::NodeUserData: /home/stack/baremetal-vlan/firstboot-config.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: 192.168.210.0/24
  # 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': '192.168.210.2', 'end': '192.168.210.12'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.210.1
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.5.1
  # Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.5.1
  InternalApiNetworkVlanID: 600
  TenantNetworkVlanID: 554
  ExternalNetworkVlanID: 552
  # 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: 'vlan'
  # 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:552:553,tenant:555:601'
  # 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%'
```

6.2.5.4. controller.yaml file

This file is in the **/home/stack/baremetal-vlan/nic-configs/** directory. This example defines the following switches: **br-isolated**, **br-vlan**, and **br-ex**. **nic2** is under **br-isolated** and **nic3** is under **br-ex**:

```
heat_template_version: pike

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 with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  DnsServers: # Override this with 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 with 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: interface
              name: nic1
              use_dhcp: false
```

```
-
 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: ovs_bridge
 name: br-ex
 use_dhcp: false
 dns_servers: {get_param: DnsServers}
 members:
   -
     type: interface
     name: nic4
     # 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}
-
 type: ovs_bridge
 name: br-vlan
 use_dhcp: false
 dns_servers: {get_param: DnsServers}
 members:
   -
     type: interface
```

```
              name: nic3
            -
              type: vlan
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}

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

6.2.5.5. compute.yaml file

This file is in the **/home/stack/baremetal-vlan/nic-configs/** directory. Most of the options in the Compute configuration are the same as in the Controller configuration. In this example, **nic4** is under **br-ex** to be used for External connectivity (Floating IP network)

```
heat_template_version: pike

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 with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this with parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this with 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 with 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: interface
              name: nic1
              use_dhcp: false
            -
              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}
                default: true
            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: ovs_bridge
            name: br-ex
            use_dhcp: false
            members:
              -
                type: interface
                name: nic4
          -
            type: ovs_bridge
            name: br-vlan
            use_dhcp: false
            dns_servers: {get_param: DnsServers}
            members:
              -
                type: interface
                name: nic3
              -
                type: vlan
                vlan_id: {get_param: TenantNetworkVlanID}
                addresses:
                  -
                    ip_netmask: {get_param: TenantIpSubnet}


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

6.2.6. Red Hat OpenStack Platform director configuration files used in this scenario

6.2.6.1. neutron.conf file

This file is in the **/etc/neutron/** directory and contains the following information:

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

6.2.6.2. ml2_conf.ini file

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

```
[DEFAULT]
[ml2]
type_drivers = vxlan,vlan,flat,gre
tenant_network_types = vlan
mechanism_drivers = opendaylight_v2
extension_drivers = qos,port_security
path_mtu = 0

[ml2_type_flat]
flat_networks = datacentre

[ml2_type_geneve]
[ml2_type_gre]
tunnel_id_ranges = 1:4094

[ml2_type_vlan]
network_vlan_ranges = datacentre:552:553,tenant:555:601

[ml2_type_vxlan]
vni_ranges = 1:4094
vxlan_group = 224.0.0.1

[securitygroup]
[ml2_odl]
password=<PASSWORD>
username=<USER>
url=http://172.17.0.10:8081/controller/nb/v2/neutron
```

1. Under the [ml2] section note that VXLAN is used as the networks' type and so is the **opendaylight_v2** mechanism driver.

2. Under [ml2_type_vlan], set the same mappings as in the **network-environment.yaml** file.

3. Under [ml2_odl], you should see the configuration accessing the OpenDaylightController.

You can use these details to confirm access to the OpenDaylight Controller:

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

CHAPTER 7. HIGH AVAILABILITY AND CLUSTERING WITH OPENDAYLIGHT

Red Hat OpenStack Platform 13 supports High Availability clustering for both neutron and the OpenDaylight Controller. The table below shows the architecture recommended to run a high availability cluster:

Node type	Number of nodes	Node mode
Neutron	3	active/active/active
OpenDaylight	3	active/active/active
Compute nodes (nova or OVS)	any	

The OpenDaylight role is composable, so it can be deployed on the same nodes as the neutron nodes, or on separate nodes. The setup is an all-active setup. All nodes can handle requests. If the receiving node cannot handle a request, the node forwards the request to another appropriate node. All nodes maintain synchronisation with each other. In Open vSwitch database schema (OVSDB) Southbound, available Controller nodes share the Open vSwitches, so that a specific node in the cluster handles each switch.

7.1. CONFIGURING OPENDAYLIGHT FOR HIGH AVAILABILITY AND CLUSTERING

Because the Red Hat OpenStack Platform director deploys the OpenDaylight Controller nodes, it has all the information required to configure clustering for OpenDaylight. Each OpenDaylight node requires an **akka.conf** configuration file that identifies the node *role* (its name in the cluster) and lists at least some of the other nodes in the cluster, the *seed* nodes. The nodes also require a **module-shards.conf** file that defines how data is replicated in the cluster. The Red Hat OpenStack Platform director makes the correct settings based on the selected deployment configuration. The **akka.conf** file depends on the nodes, while the **module-shards.conf** file depends on the nodes and the installed datastores (and hence the installed features, which we control to a large extent).

Example **akka.conf** file:

```
$ docker exec -it opendaylight_api cat
/var/lib/kolla/config_files/src/opt/opendaylight/configuration/initial/akka.conf


odl-cluster-data {
  akka {
    remote {
      netty.tcp {
        hostname = "192.0.2.1"
      }
    },
    cluster {
      seed-nodes = [
        "akka.tcp://opendaylight-cluster-data@192.0.2.1:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.2:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.3:2550"],
```

```
      roles = [ "member-1" ]
    }
  }
}
```

These example nodes are *seed nodes*. They do not need to reflect the current cluster setup as a whole. As long as one of the real nodes in the current cluster is reachable using the list of seed nodes, a starting-up node can join the cluster. In the configuration file, you can use names instead of IP addresses.

7.2. CLUSTER BEHAVIOUR

The cluster is not defined dynamically, which means that it does not adjust automatically. It is not possible to start a new node and connect it to an existing cluster by configuring only the new node. The cluster must be informed about nodes' additions and removals through the cluster administration RPCs.

The cluster is a leader/followers model. One of the active nodes is elected as the leader, and the remaining active nodes become followers. The cluster handles persistence according to the Raft consensus-based model. Following this principle, a transaction is only committed if the majority of the nodes in the cluster agree.

In OpenDaylight, if a node loses its connection with the cluster, its local transactions will no longer make forward progress. Eventually they will timeout (10 minutes by default) and the front-end actor will stop. All this applies per shard, so different shards can have different leaders. The behaviour results in one of the following:

- Lack of communication for less than ten minutes results in the minority nodes reconnecting with the majority leader. All the transactions are rolled back and the majority transactions are replayed.
- Lack of communication for more than ten minutes results in the minority nodes stopping working and recording the information into log messages. Read-only requests should still complete, but no changes persist and the nodes are not able to re-join the cluster autonomously.

This means that users must monitor the nodes. Users must check for availability and cluster synchronisation and restart them if they are out of synchronization for too long. For monitoring the nodes, users can use the Jolokia REST service. For more information, see Monitoring with Jolokia.

7.3. CLUSTER REQUIREMENTS

There are no specific networking requirements to support the cluster, such as bonding or MTUs. The cluster communications do not support high latencies, but latencies on the order of data-centre level are acceptable.

7.4. OPEN VSWITCH CONFIGURATION

Red Hat OpenStack Platform director configures each switch with all of the controllers automatically. The OVSDB supports sharing switches among the cluster nodes, to allow some level of load-balancing. However, each switch contacts all the nodes in the cluster and picks the one that answers first and makes it the master switch by default. This behaviour leads to *clustering* of the controller assignments when the fastest answering node handles most of the switches.

7.5. CLUSTER MONITORING

7.5.1. Monitoring with Jolokia

To monitor the status of the cluster, you must enable the **Jolokia** support in OpenDaylight.

Obtain a configuration datastore clustering report from the Jolokia address:

```
# curl -u <odl_username>
<odl_password>http://<odl_ip>:8081/jolokia/read/org.opendaylight.controller:type=DistributedConfigData
store,Category=ShardManager,name=shard-manager-config
```

Obtain an operational datastore clustering report from the Jolokia address:

```
# curl -u <odl_username>
<odl_password>http://<odl_ip>:8081/jolokia/read/org.opendaylight.controller:type=DistributedOperationa
lDatastore,Category=ShardManager,name=shard-manager-operational
```

The reports are JSON documents.

NOTE

You must change the **IP address** and the **member-1** values to match your environment. The IP address can point to a VIP, if there is no preference in which node will respond. However, addressing specific controllers provides more relevant results.

This description must indicate the same leader on each node.

NOTE

You can also monitor the cluster with the **Cluster Monitor Tool** that is being developed by the upstream OpenDaylight team. You can find it in the OpenDaylight Github repository.

The tool is not a part of Red Hat OpenStack Platform 13 and as such is not supported or provided by Red Hat.

7.6. UNDERSTANDING OPENDAYLIGHT PORTS

The official list of all OpenDaylight ports is available on the OpenDaylight wiki page. The ports relevant for this scenario are:

Port number	Used for
2550	clustering
6653	OpenFlow
6640, 6641	OVSDB
8087	neutron
8081	RESTCONF, Jolokia

Blocking traffic to these ports on the controller has the following effects:

Clustering

The clustered nodes will not be able to communicate. When running in clustered mode, each node must have at least one peer. If all traffic is blocked, the controller stops.

OpenFlow

The switches will not be able to reach the controller.

OVSDB

Open vSwitch will not be able to reach the controller. The controller will be able to initiate an active OVS connection, but any pings from the switch to the controller will fail and the switch will finally fail over to another controller.

neutron

Neutron will not be able to reach the controller.

RESTCONF

External tools using the REST endpoints will not be able to reach the controller. In this scenario, it only should affect the monitoring tools.

On the OpenDaylight side, the logs show only blocked traffic for clustering, because the other ports are used to talk to the ODL controller.

Blocking traffic to these ports on the target devices has the following effects:

Clustering

The clustered nodes will not be able to communicate. When running in clustered mode, each node must have at least one peer. If all traffic is blocked, the controller stops.

OpenFlow

The controller will not be able to push flows.

OVSDB

The controller will not be able to reach the switch (the controller will be able to respond to passive OVS connections).

In all cases in the latter situation, because OpenDaylight maintains its configuration and its operational status in distinct trees, the configuration still points to the unreachable devices, and the controller continues to try to connect to them.

7.7. UNDERSTANDING OPENDAYLIGHT FLOWS

Flow	Explanation
Neutron → ODL	Neutron to HA Proxy to ODL. PaceMaker manages the VIP (with three backing PIPs). The driver tries to keep TCP sessions open which may have an impact (https://review.openstack.org/#/c/440866/).
ODL → Neutron	There are no ODL-initiated communications.
ODL → ODL	ODL nodes communicate with each other on port 2550 (configurable).

Flow	Explanation
ODL → OVS	ODL communicates with switches using OVSDB (ports 6640 and 6641) and OpenFlow (port 6633). There is no VIP involved, ODL knows every switch's IP address and each ODL node knows about every switch.
OVS → ODL	ODL communicates with switches using OVSDB (ports 6640 and 6641) and OpenFlow (port 6633). There is no VIP involved, ODL configures every switch so that it knows about all the controllers. Notifications from the switches to the controller are sent to all nodes.

7.8. NEUTRON DHCP AGENT HA

The default setup runs the DHCP agent on all neutron nodes, along with the OVS agent. The roles are composable, so the agents can be separated from the controllers. The DHCP agent is important for HA only during the port bringing-up phase and during lease renewal. On port creation, neutron assigns IP and MAC addresses and configures all the DHCP agents appropriately, before the port comes up. During this phase, all DHCP agents answer the resulting DHCP requests.

To maximise the dataplane availability in the case of a DHCP agent failure, the leases are configured with long lease times, and the nodes are configured with short renewal delays. Thus, the DHCP agents are seldom needed, but when they are, the requesting nodes will quickly fail an unavailable DHCP agent and issue a broadcast request, picking up any remaining DHCP agent automatically.

The agents have their own process monitors. **systemd** starts the agents, and they create their namespaces and start the processes inside them. If an agent dies, the namespace stays up, **systemd** restarts the agent without terminating or restarting any other processes (it does not own them). Then the agent re-attaches to the namespace and re-uses it together with all running processes.

7.9. NEUTRON METADATA AGENT HA

In the reference implementation, the metadata services run on the controllers, that are combined with the network nodes, in the same namespace as the corresponding DHCP agent. A metadata proxy listens on port 80, and a static route redirects the traffic from the virtual machines to the proxy using the well-known metadata address. The proxy uses a Unix socket to talk to the metadata service, which is on the same node, and the latter talks to nova. With Unix sockets, we do not need to be able to route IP between the proxy and the service, so the metadata service is available even if the node is not routed. HA is handled using keepalive and VRRP elections. Failover time is 2-5s. The agents are handled in the same way as DHCP agents (with systemd and namespaces).

The metadata service in Red Hat OpenStack Platform 11 is a custom Python script while in Red Hat OpenStack Platform 13 it is HAProxy, which lowers the memory usage by 30. This is particularly significant because many users run one proxy per router, and hundreds if not thousands of routers per controller.

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 Carbon 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 7.4. For information on installing Red Hat Enterprise Linux, see the corresponding installation guide at Red Hat Enterprise Linux Installation Guide.
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.
NFV Documentation	For more details on planning your Red Hat OpenStack Platform deployment with NFV, see Network Functions Virtualization Planning and Configuration Guide.