Red Hat OpenStack Platform 17.1

Configuring Red Hat OpenStack Platform networking

Managing the OpenStack Networking service (neutron)

Last Updated: 2024-05-23

Red Hat OpenStack Platform 17.1 Configuring Red Hat OpenStack Platform networking

Managing the OpenStack Networking service (neutron)

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

A cookbook for common OpenStack Networking tasks.

Table of Contents

PREFACE

NOTE

You cannot apply a role-based access control (RBAC)-shared security group directly to an instance during instance creation. To apply an RBAC-shared security group to an instance you must first create the port, apply the shared security group to that port, and then assign that port to the instance. See Adding a security group to a port .

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.

2. Click the following link to open a the **Create Issue** page: Create Issue

3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.

4. Click **Create**.

CHAPTER 1. INTRODUCTION TO OPENSTACK NETWORKING

The Networking service (neutron) is the software-defined networking (SDN) component of Red Hat OpenStack Platform (RHOSP). The RHOSP Networking service manages internal and external traffic to and from virtual machine instances and provides core services such as routing, segmentation, DHCP, and metadata. It provides the API for virtual networking capabilities and management of switches, routers, ports, and firewalls.

1.1. MANAGING YOUR RHOSP NETWORKS

With the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) you can effectively meet your site’s networking goals. You can:

- **Provide connectivity to VM instances within a project.**
  Project networks primarily enable general (non-privileged) projects to manage networks without involving administrators. These networks are entirely virtual and require virtual routers to interact with other project networks and external networks such as the Internet. Project networks also usually provide DHCP and metadata services to VM (virtual machine) instances. RHOSP supports the following project network types: flat, VLAN, VXLAN, GRE, and GENEVE.

  For more information, see Managing project networks.

- **Connect VM instances to networks outside of a project.**
  Provider networks provide connectivity like project networks. But only administrative (privileged) users can manage those networks because they interface with the physical network infrastructure. RHOSP supports the following provider network types: flat and VLAN.

  Inside project networks, you can use pools of floating IP addresses or a single floating IP address to direct ingress traffic to your VM instances. Using bridge mappings, you can associate a physical network name (an interface label) to a bridge created with OVS or OVN to allow provider network traffic to reach the physical network.

  For more information, see Connecting VM instances to physical networks .

- **Create a network that is optimized for the edge.**
  Operators can create routed provider networks that are typically used in edge deployments, and rely on multiple layer 2 network segments instead of traditional networks that have only one segment.

  Routed provider networks simplify the cloud for end users because they see only one network. For cloud operators, routed provider networks deliver scalabilty and fault tolerance. For example, if a major error occurs, only one segment is impacted instead of the entire network failing.

  For more information, see Deploying routed provider networks .

- **Make your network resources highly available.**
  You can use availability zones (AZs) and Virtual Router Redundancy Protocol (VRRP) to keep your network resources highly available. Operators group network nodes that are attached to different power sources on different AZs. Next, operators schedule crucial services such as DHCP, L3, FW, and so on to be on separate AZs.

  RHOSP uses VRRP to make project routers and floating IP addresses highly available. An alternative to centralized routing, Distributed Virtual Routing (DVR) offers an alternative routing design based on VRRP that deploys the L3 agent and schedules routers on every Compute node.

For more information, see Using availability zones to make network resources highly available .

- **Secure your network at the port level.**
  Security groups provide a container for virtual firewall rules that control ingress (inbound to instances) and egress (outbound from instances) network traffic at the port level. Security groups use a default deny policy and only contain rules that allow specific traffic. Each port can reference one or more security groups in an additive fashion. The firewall driver translates security group rules to a configuration for the underlying packet filtering technology such as iptables.

  By default, security groups are stateful. In ML2/OVN deployments, you can also create stateless security groups. A stateless security group can provide significant performance benefits. Unlike stateful security groups, stateless security groups do not automatically allow returning traffic, so you must create a complimentary security group rule to allow the return of related traffic.

  For more information, see Configuring shared security groups.

- **Manage port traffic.**
  With allowed address pairs you identify a specific MAC address, IP address, or both to allow network traffic to pass through a port regardless of the subnet. When you define allowed address pairs, you are able to use protocols like VRRP (Virtual Router Redundancy Protocol) that float an IP address between two VM instances to enable fast data plane failover.

  For more information, see Configuring allowed address pairs.

- **Optimize large overlay networks.**
  Using the L2 Population driver you can enable broadcast, multicast, and unicast traffic to scale out on large overlay networks.

  For more information, see Configuring the L2 population driver .

- **Set ingress and egress limits for traffic on VM instances.**
  You can offer varying service levels for instances by using quality of service (QoS) policies to apply rate limits to egress and ingress traffic. You can apply QoS policies to individual ports. You can also apply QoS policies to a project network, where ports with no specific policy attached inherit the policy.

  For more information, see Configuring Quality of Service (QoS) policies.

- **Manage the amount of network resources RHOSP projects can create.**
  With the Networking service quota options you can set limits on the amount of network resources project users can create. This includes resources such as ports, subnets, networks, and so on.

  For more information, see Managing project quotas.

- **Optimize your VM instances for Network Functions Virtualization (NFV).**
  Instances can send and receive VLAN-tagged traffic over a single virtual NIC. This is particularly useful for NFV applications (VNFs) that expect VLAN-tagged traffic, allowing a single virtual NIC to serve multiple customers or services.

  In a VLAN transparent network, you set up VLAN tagging in the VM instances. The VLAN tags are transferred over the network and consumed by the VM instances on the same VLAN, and ignored by other instances and devices. VLAN trunks support VLAN-aware instances by combining VLANs into a single trunked port.

  For more information, see VLAN-aware instances.

- **Control which projects can attach instances to a shared network.**
  Using role-based access control (RBAC) policies in the RHOSP Networking service, cloud administrators can remove the ability for some projects to create networks and can instead allow them to attach to pre-existing networks that correspond to their project.

  For more information, see Configuring RBAC policies .

- **Control network access to and from instances.**
  You can control network and protocol access to and from instances by using security groups. Security groups are sets of IP filter rules that, for example, allow users to perform an ICMP ping on an instance, and run SSH to connect to an instance. The security group rules are applied to all instances within a project.

  For more information, see Configuring security groups.

- **Logging traffic flow events into and out of an instance.**
  You can create packet logs for security groups to monitor traffic flows into and out of a virtual machine (VM) instance. Each log generates a stream of data about packet flow events and appends it to a common log file on the Compute host from which the VM instance was launched.

  For more information, see Logging security group actions.

1.2. NETWORKING SERVICE COMPONENTS

The Red Hat OpenStack Platform (RHOSP) Networking service (neutron) includes the following components:

- API server
  The RHOSP networking API includes support for Layer 2 networking and IP Address Management (IPAM), as well as an extension for a Layer 3 router construct that enables routing between Layer 2 networks and gateways to external networks. RHOSP networking includes a growing list of plug-ins that enable interoperability with various commercial and open source network technologies, including routers, switches, virtual switches and software-defined networking (SDN) controllers.

- Modular Layer 2 (ML2) plug-in and agents
  ML2 plugs and unplugs ports, creates networks or subnets, and provides IP addressing.

- Messaging queue
  Accepts and routes RPC requests between RHOSP services to complete API operations.

1.3. MODULAR LAYER 2 (ML2) NETWORKING

Modular Layer 2 (ML2) is the Red Hat OpenStack Platform (RHOSP) networking core plug-in. The ML2 modular design enables the concurrent operation of mixed network technologies through mechanism drivers. Open Virtual Network (OVN) is the default mechanism driver used with ML2.

The ML2 framework distinguishes between the two kinds of drivers that can be configured:

Type drivers

Define how an RHOSP network is technically realized.
Each available network type is managed by an ML2 type driver, and they maintain any required type-specific network state. Validating the type-specific information for provider networks, type drivers are responsible for the allocation of a free segment in project networks. Examples of type drivers are GENEVE, GRE, VXLAN, and so on.

Mechanism drivers

Define the mechanism to access an RHOSP network of a certain type.
The mechanism driver takes the information established by the type driver and applies it to the networking mechanisms that have been enabled. Examples of mechanism drivers are Open Virtual Networking (OVN) and Open vSwitch (OVS).

Mechanism drivers can employ L2 agents, and by using RPC interact directly with external devices or controllers. You can use multiple mechanism and type drivers simultaneously to access different ports of the same virtual network.

Additional resources

- Section 1.8, "Modular Layer 2 (ML2) type and mechanism driver compatibility"

1.4. ML2 NETWORK TYPES

You can operate multiple network segments at the same time. ML2 supports the use and interconnection of multiple network segments. You don't have to bind a port to a network segment because ML2 binds ports to segements with connectivity. Depending on the mechanism driver, ML2 supports the following network segment types:

- Flat
- VLAN
- GENEVE tunnels
- VXLAN and GRE tunnels

Flat

All virtual machine (VM) instances reside on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation occurs.

VLAN

With RHOSP networking users can create multiple provider or project networks using VLAN IDs (802.1Q tagged) that correspond to VLANs present in the physical network. This allows instances to communicate with each other across the environment. They can also communicate with dedicated servers, firewalls, load balancers and other network infrastructure on the same Layer 2 VLAN.
You can use VLANs to segment network traffic for computers running on the same switch. This means that you can logically divide your switch by configuring the ports to be members of different networks — they are basically mini-LANs that you can use to separate traffic for security reasons.

For example, if your switch has 24 ports in total, you can assign ports 1-6 to VLAN200, and ports 7-18 to VLAN201. As a result, computers connected to VLAN200 are completely separate from those on VLAN201; they cannot communicate directly, and if they wanted to, the traffic must pass through a router as if they were two separate physical switches. Firewalls can also be useful for governing which VLANs can communicate with each other.

GENEVE tunnels

GENEVE recognizes and accommodates changing capabilities and needs of different devices in network virtualization. It provides a framework for tunneling rather than being prescriptive about the entire system. Geneve defines the content of the metadata flexibly that is added during encapsulation and tries to adapt to various virtualization scenarios. It uses UDP as its transport protocol and is dynamic in size using extensible option headers. Geneve supports unicast, multicast, and broadcast. The GENEVE type driver is compatible with the ML2/OVN mechanism driver.

VXLAN and GRE tunnels

VXLAN and GRE use network overlays to support private communication between instances. An RHOSP networking router is required to enable traffic to traverse outside of the GRE or VXLAN project network. A router is also required to connect directly-connected project networks with external networks, including the internet; the router provides the ability to connect to instances directly from an external network using floating IP addresses. VXLAN and GRE type drivers are compatible with the ML2/OVS mechanism driver.

Additional resources

- Section 1.8, “Modular Layer 2 (ML2) type and mechanism driver compatibility”

1.5. MODULAR LAYER 2 (ML2) MECHANISM DRIVERS

Modular Layer 2 (ML2) plug-ins are implemented as mechanisms with a common code base. This approach enables code reuse and eliminates much of the complexity around code maintenance and testing.

You enable mechanism drivers using the Orchestration service (heat) parameter, **NeutronMechanismDrivers**. Here is an example from a heat custom environment file:

```
parameter_defaults:
  ...
  NeutronMechanismDrivers: ansible,ovn,baremetal
  ...
```

The order in which you specify the mechanism drivers matters. In the earlier example, if you want to bind a port using the baremetal mechanism driver, then you must specify **baremetal** before **ansible**. Otherwise, the ansible driver will bind the port, because it precedes **baremetal** in the list of values for **NeutronMechanismDrivers**.

Red Hat chose ML2/OVN as the default mechanism driver for all new deployments starting with RHOSP 15 because it offers immediate advantages over the ML2/OVS mechanism driver for most customers today. Those advantages multiply with each release while we continue to enhance and improve the ML2/OVN feature set.

Support is available for the deprecated ML2/OVS mechanism driver through the RHOSP 17 releases. During this time, the ML2/OVS driver remains in maintenance mode, receiving bug fixes and normal support, and most new feature development happens in the ML2/OVN mechanism driver.

In RHOSP 18.0, Red Hat plans to completely remove the ML2/OVS mechanism driver and stop supporting it.

If your existing Red Hat OpenStack Platform (RHOSP) deployment uses the ML2/OVS mechanism driver, start now to evaluate a plan to migrate to the mechanism driver. Migration is supported in RHOSP 16.2 and will be supported in RHOSP 17.1. Migration tools are available in RHOSP 17.0 for test purposes only.

Red Hat requires that you file a proactive support case before attempting a migration from ML2/OVS to ML2/OVN. Red Hat does not support migrations without the proactive support case. See How to open a proactive case for a planned activity on Red Hat OpenStack Platform?

Additional resources

- Neutron in *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform*
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

1.6. OPEN VSWITCH

Open vSwitch (OVS) is a software-defined networking (SDN) virtual switch similar to the Linux software bridge. OVS provides switching services to virtualized networks with support for industry standard OpenFlow and sFlow. OVS can also integrate with physical switches using layer 2 features, such as STP, LACP, and 802.1Q VLAN tagging. Open vSwitch version 1.11.0-1.el6 or later also supports tunneling with VXLAN and GRE.

NOTE

To mitigate the risk of network loops in OVS, only a single interface or a single bond can be a member of a given bridge. If you require multiple bonds or interfaces, you can configure multiple bridges.

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

1.7. OPEN VIRTUAL NETWORK (OVN)

Open Virtual Network (OVN), is a system to support logical network abstraction in virtual machine and container environments. Sometimes called open source virtual networking for Open vSwitch, OVN complements the existing capabilities of OVS to add native support for logical network abstractions, such as logical L2 and L3 overlays, security groups and services such as DHCP.

A physical network comprises physical wires, switches, and routers. A virtual network extends a physical network into a hypervisor or container platform, bridging VMs or containers into the physical network. An OVN logical network is a network implemented in software that is insulated from physical networks by tunnels or other encapsulations. This allows IP and other address spaces used in logical networks to overlap with those used on physical networks without causing conflicts. Logical network topologies can be arranged without regard for the topologies of the physical networks on which they run. Thus, VMs that are part of a logical network can migrate from one physical machine to another without network disruption.

The encapsulation layer prevents VMs and containers connected to a logical network from communicating with nodes on physical networks. For clustering VMs and containers, this can be acceptable or even desirable, but in many cases VMs and containers do need connectivity to physical networks. OVN provides multiple forms of gateways for this purpose. An OVN deployment consists of several components:

Cloud Management System (CMS)

integrates OVN into a physical network by managing the OVN logical network elements and connecting the OVN logical network infrastructure to physical network elements. Some examples include OpenStack and OpenShift.

OVN databases

stores data representing the OVN logical and physical networks.

Hypervisors

run Open vSwitch and translate the OVN logical network into OpenFlow on a physical or virtual machine.

Gateways

extends a tunnel-based OVN logical network into a physical network by forwarding packets between tunnels and the physical network infrastructure.

1.8. MODULAR LAYER 2 (ML2) TYPE AND MECHANISM DRIVER COMPATIBILITY

Refer to the following table when planning your Red Hat OpenStack Platform (RHOSP) data networks to determine the network types each Modular Layer 2 (ML2) mechanism driver supports.

Table 1.1. Network types supported by ML2 mechanism drivers

Mechanism driver	Supports these type drivers				
	Flat	GRE	VLAN	VXLAN	GENEVE
Open Virtual Network (OVN)	Yes	No	Yes	Yes [1]	Yes
Open vSwitch (OVS)	Yes	Yes	Yes	Yes	No

[1] ML2/OVN VXLAN support is limited to 4096 networks and 4096 ports per network. Also, ACLs that rely on the ingress port do not work with ML2/OVN and VXLAN, because the ingress port is not passed.

1.9. EXTENSION DRIVERS FOR THE RHOSP NETWORKING SERVICE

The Red Hat OpenStack Platform (RHOSP) Networking service (neutron) is extensible. Extensions serve two purposes: they allow the introduction of new features in the API without requiring a version change and they allow the introduction of vendor specific niche functionality. Applications can programmatically list available extensions by performing a GET on the /**extensions** URI. Note that this is a versioned request; that is, an extension available in one API version might not be available in another.

The ML2 plug-in also supports extension drivers that allows other pluggable drivers to extend the core resources implemented in the ML2 plug-in for network objects. Examples of extension drivers include support for QoS, port security, and so on.

CHAPTER 2. WORKING WITH ML2/OVN

Red Hat OpenStack Platform (RHOSP) networks are managed by the Networking service (neutron). The core of the Networking service is the Modular Layer 2 (ML2) plug-in, and the default mechanism driver for RHOSP ML2 plug-in is the Open Virtual Networking (OVN) mechanism driver.

Earlier RHOSP versions used the Open vSwitch (OVS) mechanism driver by default, but Red Hat recommends the ML2/OVN mechanism driver for most deployments.

2.1. LIST OF COMPONENTS IN THE RHOSP OVN ARCHITECTURE

The RHOSP OVN architecture replaces the OVS Modular Layer 2 (ML2) mechanism driver with the OVN ML2 mechanism driver to support the Networking API. OVN provides networking services for the Red Hat OpenStack platform.

As illustrated in Figure 2.1, the OVN architecture consists of the following components and services:

ML2 plug-in with OVN mechanism driver

The ML2 plug-in translates the OpenStack-specific networking configuration into the platform-neutral OVN logical networking configuration. It typically runs on the Controller node.

OVN northbound (NB) database (ovn-nb)

This database stores the logical OVN networking configuration from the OVN ML2 plugin. It typically runs on the Controller node and listens on TCP port **6641**.

OVN northbound service (ovn-northd)

This service converts the logical networking configuration from the OVN NB database to the logical data path flows and populates these on the OVN Southbound database. It typically runs on the Controller node.

OVN southbound (SB) database (ovn-sb)

This database stores the converted logical data path flows. It typically runs on the Controller node and listens on TCP port **6642**.

OVN controller (ovn-controller)

This controller connects to the OVN SB database and acts as the open vSwitch controller to control and monitor network traffic. It runs on all Compute and gateway nodes where **OS::Tripleo::Services::OVNController** is defined.

OVN metadata agent (ovn-metadata-agent)

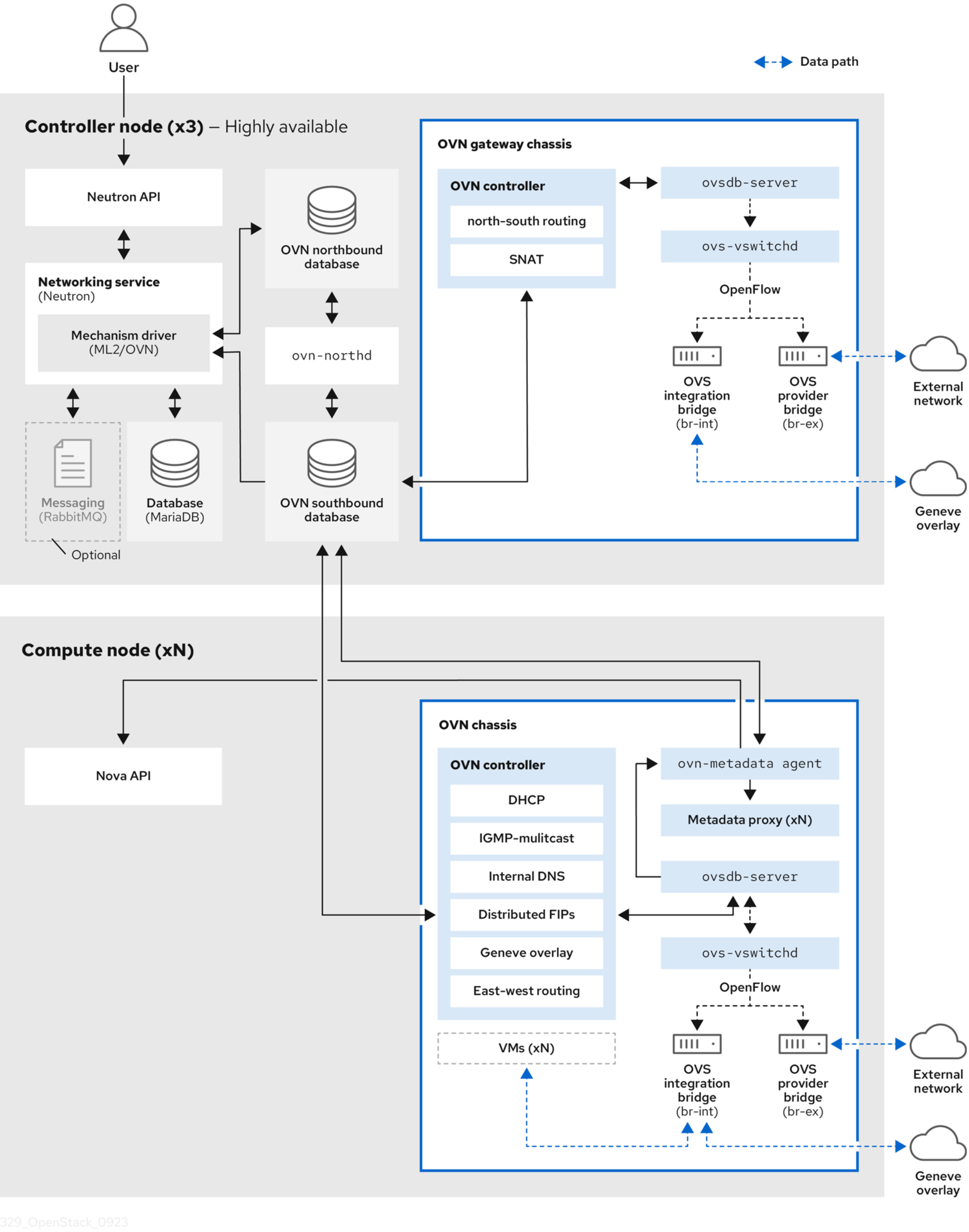
This agent creates the **haproxy** instances for managing the OVS interfaces, network namespaces and HAProxy processes used to proxy metadata API requests. The agent runs on all Compute and gateway nodes where **OS::TripleO::Services::OVNMetadataAgent** is defined.

OVS database server (OVSDB)

Hosts the OVN Northbound and Southbound databases. Also interacts with **ovs-vswitchd** to host the OVS database **conf.db**.

> NOTE
>
> The schema file for the NB database is located in **/usr/share/ovn/ovn-nb.ovsschema**, and the SB database schema file is in **/usr/share/ovn/ovn-sb.ovsschema**.

Figure 2.1. OVN architecture in a RHOSP environment



2.2. ML2/OVN DATABASES

In Red Hat OpenStack Platform ML2/OVN deployments, network configuration information passes between processes through shared distributed databases. You can inspect these databases to verify the status of the network and identify issues.

OVN northbound database

The northbound database (**OVN_Northbound**) serves as the interface between OVN and a cloud management system such as Red Hat OpenStack Platform (RHOSP). RHOSP produces the contents of the northbound database.

The northbound database contains the current desired state of the network, presented as a collection of logical ports, logical switches, logical routers, and more. Every RHOSP Networking service (neutron) object is represented in a table in the northbound database.

OVN southbound database

The southbound database (**OVN_Southbound**) holds the logical and physical configuration state for OVN system to support virtual network abstraction. The **ovn-controller** uses the information in this database to configure OVS to satisfy Networking service (neutron) requirements.

2.3. THE OVN-CONTROLLER SERVICE ON COMPUTE NODES

The **ovn-controller** service runs on each Compute node and connects to the OVN southbound (SB) database server to retrieve the logical flows. The **ovn-controller** translates these logical flows into physical OpenFlow flows and adds the flows to the OVS bridge (**br-int**).

To communicate with **ovs-vswitchd** and install the OpenFlow flows, the **ovn-controller** connects to one of the active **ovsdb-server** servers (which host **conf.db**) using the UNIX socket path that was passed when **ovn-controller** was started (for example **unix:/var/run/openvswitch/db.sock**).

The **ovn-controller** service expects certain key-value pairs in the **external_ids** column of the **Open_vSwitch** table; **puppet-ovn** uses **puppet-vswitch** to populate these fields. The following example shows the key-value pairs that **puppet-vswitch** configures in the **external_ids** column:

```
hostname=<HOST NAME>
ovn-encap-ip=<IP OF THE NODE>
ovn-encap-type=geneve
ovn-remote=tcp:OVN_DBS_VIP:6642
```

2.4. OVN METADATA AGENT ON COMPUTE NODES

The OVN metadata agent is configured in the **tripleo-heat-templates/deployment/ovn/ovn-metadata-container-puppet.yaml** file and included in the default Compute role through **OS::TripleO::Services::OVNMetadataAgent**. As such, the OVN metadata agent with default parameters is deployed as part of the OVN deployment.

OpenStack guest instances access the Networking metadata service available at the link-local IP address: 169.254.169.254. The **neutron-ovn-metadata-agent** has access to the host networks where the Compute metadata API exists. Each HAProxy is in a network namespace that is not able to reach the appropriate host network. HaProxy adds the necessary headers to the metadata API request and then forwards the request to the **neutron-ovn-metadata-agent** over a UNIX domain socket.

The OVN Networking service creates a unique network namespace for each virtual network that enables the metadata service. Each network accessed by the instances on the Compute node has a corresponding metadata namespace (ovnmeta-<network_uuid>).

2.5. THE OVN COMPOSABLE SERVICE

Red Hat OpenStack Platform usually consists of nodes in pre-defined roles, such as nodes in Controller roles, Compute roles, and different storage role types. Each of these default roles contains a set of services that are defined in the core heat template collection.

In a default Red Hat OpenStack (RHOSP) deployment, the ML2/OVN composable service, ovn-dbs, runs on Controller nodes. Because the service is composable, you can assign it to another role, such as a Networker role. By choosing to assign the ML2/OVN service to another role you can reduce the load on the Controller node, and implement a high-availability strategy by isolating the Networking service on Networker nodes.

Related information

- Deploying a custom role with ML2/OVN
- SR-IOV with ML2/OVN and native OVN DHCP

2.6. LAYER 3 HIGH AVAILABILITY WITH OVN

OVN supports Layer 3 high availability (L3 HA) without any special configuration.

NOTE

When you create a router, do not use **--ha** option because OVN routers are highly available by default. **Openstack router create** commands that include the **--ha** option fail.

OVN automatically schedules the router port to all available gateway nodes that can act as an L3 gateway on the specified external network. OVN L3 HA uses the **gateway_chassis** column in the OVN **Logical_Router_Port** table. Most functionality is managed by OpenFlow rules with bundled active_passive outputs. The **ovn-controller** handles the Address Resolution Protocol (ARP) responder and router enablement and disablement. Gratuitous ARPs for FIPs and router external addresses are also periodically sent by the **ovn-controller**.

NOTE

L3HA uses OVN to balance the routers back to the original gateway nodes to avoid any nodes becoming a bottleneck.

BFD monitoring

OVN uses the Bidirectional Forwarding Detection (BFD) protocol to monitor the availability of the gateway nodes. This protocol is encapsulated on top of the Geneve tunnels established from node to node.

Each gateway node monitors all the other gateway nodes in a star topology in the deployment. Gateway nodes also monitor the compute nodes to let the gateways enable and disable routing of packets and ARP responses and announcements.

Each compute node uses BFD to monitor each gateway node and automatically steers external traffic, such as source and destination Network Address Translation (SNAT and DNAT), through the active gateway node for a given router. Compute nodes do not need to monitor other compute nodes.

NOTE

External network failures are not detected as would happen with an ML2-OVS configuration.

L3 HA for OVN supports the following failure modes:

- The gateway node becomes disconnected from the network (tunneling interface).
- **ovs-vswitchd** stops (**ovs-switchd** is responsible for BFD signaling)
- **ovn-controller** stops (**ovn-controller** removes itself as a registered node).

NOTE

This BFD monitoring mechanism only works for link failures, not for routing failures.

2.7. ACTIVE-ACTIVE CLUSTERED DATABASE SERVICE MODEL

Red Hat OpenStack Platform (RHOSP) ML2/OVN deployments use a clustered database service model that applies the Raft consensus algorithm to enhance performance of OVS database protocol traffic and provide faster, more reliable failover handling. Starting in RHOSP 17.0, the clustered database service model replaces the pacemaker-based, active/backup model.

A clustered database operates on a cluster of at least three database servers on different hosts. Servers use the Raft consensus algorithm to synchronize writes and share network traffic continuously across the cluster. The cluster elects one server as the leader. All servers in the cluster can handle database read operations, which mitigates potential bottlenecks on the control plane. Write operations are handled by the cluster leader.

If a server fails, a new cluster leader is elected and the traffic is redistributed among the remaining operational servers. The clustered database service model handles failovers more efficiently than the pacemaker-based model did. This mitigates related downtime and complications that can occur with longer failover times.

The leader election process requires a majority, so the fault tolerance capacity is limited by the highest odd number in the cluster. For example, a three-server cluster continues to operate if one server fails. A five-server cluster tolerates up to two failures. Increasing the number of servers to an even number does not increase fault tolerance. For example, a four-server cluster cannot tolerate more failures than a three-server cluster.

Most RHOSP deployments use three servers.

Clusters larger than five servers also work, with every two added servers allowing the cluster to tolerate an additional failure, but write performance decreases.

For information on monitoring the status of the database servers, see Monitoring OVN database status.

2.8. DEPLOYING A CUSTOM ROLE WITH ML2/OVN

In a default Red Hat OpenStack (RHOSP) deployment, the ML2/OVN composable service runs on Controller nodes. You can optionally use supported custom roles like those described in the following examples.

Networker

Run the OVN composable services on dedicated networker nodes.

Networker with SR-IOV

Run the OVN composable services on dedicated networker nodes with SR-IOV.

Controller with SR-IOV

Run the OVN composable services on SR-IOV capable controller nodes.

You can also generate your own custom roles.

Limitations

The following limitations apply to the use of SR-IOV with ML2/OVN and native OVN DHCP in this release.

- All external ports are scheduled on a single gateway node because there is only one HA Chassis Group for all of the ports.
- North/south routing on VF(direct) ports on VLAN tenant networks does not work with SR-IOV because the external ports are not colocated with the logical router's gateway ports. See https://bugs.launchpad.net/neutron/+bug/1875852.

Prerequisites

- You know how to deploy custom roles.
  For more information see Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide.

Procedure

1. Log in to the undercloud host as the **stack** user and source the **stackrc** file.

```
$ source stackrc
```

2. Choose the custom roles file that is appropriate for your deployment. Use it directly in the deploy command if it suits your needs as-is. Or you can generate your own custom roles file that combines other custom roles files.

Deployment	Role	Role File
Networker role	Networker	**Networker.yaml**
Networker role with SR-IOV	NetworkerSriov	**NetworkerSriov.yaml**
Co-located control and networker with SR-IOV	ControllerSriov	**ControllerSriov.yaml**

3. (Optional) Generate a new custom roles data file that combines one of the custom roles files listed earlier with other custom roles files.
   Follow the instructions in Creating a roles_data file in the *Customizing your Red Hat OpenStack Platform deployment* guide. Include the appropriate source role files depending on your deployment.

4. (Optional) To identify specific nodes for the role, you can create a specific hardware flavor and assign the flavor to specific nodes. Then use an environment file to define the flavor for the role, and to specify a node count.
   For more information, see the example in Creating a new role in the *Customizing your Red Hat OpenStack Platform deployment* guide.

5. Create an environment file as appropriate for your deployment.

Deployment	Sample Environment File
Networker role	neutron-ovn-dvr-ha.yaml
Networker role with SR-IOV	ovn-sriov.yaml

6. Include the following settings as appropriate for your deployment.

<table>
<tr><th>Deployment</th><th>Settings</th></tr>
<tr><td>Networker role</td><td>

```
ControllerParameters:
  OVNCMSOptions: ""
ControllerSriovParameters:
    OVNCMSOptions: ""
NetworkerParameters:
  OVNCMSOptions: "enable-chassis-as-gw"
NetworkerSriovParameters:
  OVNCMSOptions: ""
```

</td></tr>
<tr><td>Networker role with SR-IOV</td><td>

```
OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None

ControllerParameters:
  OVNCMSOptions: ""
ControllerSriovParameters:
    OVNCMSOptions: ""
NetworkerParameters:
  OVNCMSOptions: ""
NetworkerSriovParameters:
  OVNCMSOptions: "enable-chassis-as-gw"
```

</td></tr>
<tr><td>Co-located control and networker with SR-IOV</td><td>

```
OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None

ControllerParameters:
  OVNCMSOptions: ""
ControllerSriovParameters:
    OVNCMSOptions: "enable-chassis-as-gw"
NetworkerParameters:
  OVNCMSOptions: ""
NetworkerSriovParameters:
  OVNCMSOptions: ""
```

</td></tr>
</table>

7. Run the deployment command and include the core heat templates, other environment files, and the custom roles data file in your deployment command with the **-r** option.

   IMPORTANT

   The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

   ### Example

   ```
   $ openstack overcloud deploy --templates <core_heat_templates> \
   -e <other_environment_files> \
   -e /home/stack/templates/my-neutron-environment.yaml
   -r mycustom_roles_file.yaml
   ```

Verification steps

1. Log in to the Controller or Networker node as the **tripleo-admin** user:

   ### Example

   ```
   ssh tripleo-admin@controller-0
   ```

2. Ensure that **ovn_metadata_agent** is running.

   ```
   $ sudo podman ps | grep ovn_metadata
   ```

   ### Sample output

   ```
   a65125d9588d  undercloud-0.ctlplane.localdomain:8787/rh-osbs ...
   openstack-neutron-metadata-agent-ovn ...
   kolla_start  23 hours ago  Up 21 hours ago  ovn_metadata_agent
   ```

3. Ensure that Controller nodes with OVN services or dedicated Networker nodes have been configured as gateways for OVS.

   ```
   $ sudo ovs-vsctl get Open_Vswitch . external_ids:ovn-cms-options
   ```

   ### Sample output

   ```
   enable-chassis-as-gw
   ```

Additional verification steps for SR-IOV deployments

1. Log in to a Compute node as the **tripleo-admin** user:

   ### Example

   ```
   ssh tripleo-admin@compute-0
   ```

2. Ensure that **neutron_sriov_agent** is running on Compute nodes.

```
sudo podman ps | grep neutron_sriov_agent
```

Sample output

```
f54cbbf4523a  undercloud-0.ctlplane.localdomain:8787 ...
openstack-neutron-sriov-agent ...
kolla_start  23 hours ago  Up 21 hours ago  neutron_sriov_agent
```

3. Ensure that network-available SR-IOV NICs have been successfully detected.

```
$ sudo podman exec -uroot galera-bundle-podman-0 mysql nova \
-e 'select hypervisor_hostname,pci_stats from compute_nodes;'
```

Sample output

```
computesriov-1.localdomain {... {"dev_type": "type-PF", "physical_network"
: "datacentre", "trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF",
"physical_network": "datacentre", "trusted": "true", "parent_ifname":
"enp7s0f3"}, "count": 5}, ...}

computesriov-0.localdomain {... {"dev_type": "type-PF", "physical_network":
"datacentre", "trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF",
"physical_network": "datacentre", "trusted": "true", "parent_ifname":
"enp7s0f3"}, "count": 5}, ...}
```

Additional resources

- Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide
- overcloud deploy in the *Command line interface reference*

2.9. SR-IOV WITH ML2/OVN AND NATIVE OVN DHCP

You can deploy a custom role to use SR-IOV in an ML2/OVN deployment with native OVN DHCP. See Section 2.8, “Deploying a custom role with ML2/OVN” .

Limitations

The following limitations apply to the use of SR-IOV with ML2/OVN and native OVN DHCP in this release.

- All external ports are scheduled on a single gateway node because there is only one HA Chassis Group for all of the ports.
- North/south routing on VF(direct) ports on VLAN tenant networks does not work with SR-IOV because the external ports are not colocated with the logical router’s gateway ports. See https://bugs.launchpad.net/neutron/+bug/1875852.

Additional resources

- Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide.

CHAPTER 3. MANAGING PROJECT NETWORKS

Project networks help you to isolate network traffic for cloud computing. Steps to create a project network include planning and creating the network, and adding subnets and routers.

3.1. VLAN PLANNING

When you plan your Red Hat OpenStack Platform deployment, you start with a number of subnets, from which you allocate individual IP addresses. When you use multiple subnets you can segregate traffic between systems into VLANs.

For example, it is ideal that your management or API traffic is not on the same network as systems that serve web traffic. Traffic between VLANs travels through a router where you can implement firewalls to govern traffic flow.

You must plan your VLANs as part of your overall plan that includes traffic isolation, high availability, and IP address utilization for the various types of virtual networking resources in your deployment.

NOTE

The maximum number of VLANs in a single network, or in one OVS agent for a network node, is 4094. In situations where you require more than the maximum number of VLANs, you can create several provider networks (VXLAN networks) and several network nodes, one per network. Each node can contain up to 4094 private networks.

3.2. TYPES OF NETWORK TRAFFIC

You can allocate separate VLANs for the different types of network traffic that you want to host. For example, you can have separate VLANs for each of these types of networks. Only the External network must be routable to the external physical network. In this release, director provides DHCP services.

NOTE

You do not require all of the isolated VLANs in this section for every OpenStack deployment. For example, if your cloud users do not create ad hoc virtual networks on demand, then you may not require a project network. If you want each VM to connect directly to the same switch as any other physical system, connect your Compute nodes directly to a provider network and configure your instances to use that provider network directly.

- **Provisioning network** - This VLAN is dedicated to deploying new nodes using director over PXE boot. OpenStack Orchestration (heat) installs OpenStack onto the overcloud bare metal servers. These servers attach to the physical network to receive the platform installation image from the undercloud infrastructure.

- **Internal API network** - The OpenStack services use the Internal API network for communication, including API communication, RPC messages, and database communication. In addition, this network is used for operational messages between controller nodes. When planning your IP address allocation, note that each API service requires its own IP address. Specifically, you must plan IP addresses for each of the following services:

    - vip-msg (ampq)

    - vip-keystone-int

  - vip-glance-int
  - vip-cinder-int
  - vip-nova-int
  - vip-neutron-int
  - vip-horizon-int
  - vip-heat-int
  - vip-ceilometer-int
  - vip-swift-int
  - vip-keystone-pub
  - vip-glance-pub
  - vip-cinder-pub
  - vip-nova-pub
  - vip-neutron-pub
  - vip-horizon-pub
  - vip-heat-pub
  - vip-ceilometer-pub
  - vip-swift-pub

- **Storage** - Block Storage, NFS, iSCSI, and other storage services. Isolate this network to separate physical Ethernet links for performance reasons.

- **Storage Management** - OpenStack Object Storage (swift) uses this network to synchronise data objects between participating replica nodes. The proxy service acts as the 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, since they do not interact with Ceph directly but rather use the front end service. Note that the RBD driver is an exception; this traffic connects directly to Ceph.

- **Project networks** - Neutron provides each project with their own networks using either VLAN segregation (where each project network is a network VLAN), or tunneling using VXLAN or GRE. Network traffic is isolated within each project network. Each project network has an IP subnet associated with it, and multiple project networks may use the same addresses.

- **External** - The External network hosts the public API endpoints and connections to the Dashboard (horizon). You can also use this network for SNAT. In a production deployment, it is common to use a separate network for floating IP addresses and NAT.

- **Provider networks** - Use provider networks to attach instances to existing network infrastructure. You can use provider networks to map directly to an existing physical network in the data center, using flat networking or VLAN tags. This allows an instance to share the same layer-2 network as a system external to the OpenStack Networking infrastructure.

3.3. IP ADDRESS CONSUMPTION

The following systems consume IP addresses from your allocated range:

- **Physical nodes** - Each physical NIC requires one IP address. It is common practice to dedicate physical NICs to specific functions. For example, allocate management and NFS traffic to distinct physical NICs, sometimes with multiple NICs connecting across to different switches for redundancy purposes.
- **Virtual IPs (VIPs) for High Availability**- Plan to allocate between one and three VIPs for each network that controller nodes share.

3.4. VIRTUAL NETWORKING

The following virtual resources consume IP addresses in OpenStack Networking. These resources are considered local to the cloud infrastructure, and do not need to be reachable by systems in the external physical network:

- **Project networks** - Each project network requires a subnet that it can use to allocate IP addresses to instances.
- **Virtual routers** - Each router interface plugging into a subnet requires one IP address. If you want to use DHCP, each router interface requires two IP addresses.
- **Instances** - Each instance requires an address from the project subnet that hosts the instance. If you require ingress traffic, you must allocate a floating IP address to the instance from the designated external network.
- **Management traffic** - Includes OpenStack Services and API traffic. All services share a small number of VIPs. API, RPC and database services communicate on the internal API VIP.

3.5. ADDING NETWORK ROUTING

To allow traffic to be routed to and from your new network, you must add its subnet as an interface to an existing virtual router:

1. In the dashboard, select **Project** > **Network** > **Routers**
2. Select your virtual router name in the **Routers** list, and click **Add Interface**.
   In the Subnet list, select the name of your new subnet. You can optionally specify an IP address for the interface in this field.
3. Click **Add Interface**.
   Instances on your network can now communicate with systems outside the subnet.

3.6. EXAMPLE NETWORK PLAN

This example shows a number of networks that accommodate multiple subnets, with each subnet being assigned a range of IP addresses:

Table 3.1. Example subnet plan

Subnet name	Address range	Number of addresses	Subnet Mask
Provisioning network	192.168.100.1 - 192.168.100.250	250	255.255.255.0
Internal API network	172.16.1.10 - 172.16.1.250	241	255.255.255.0
Storage	172.16.2.10 - 172.16.2.250	241	255.255.255.0
Storage Management	172.16.3.10 - 172.16.3.250	241	255.255.255.0
Tenant network (GRE/VXLAN)	172.16.4.10 - 172.16.4.250	241	255.255.255.0
External network (incl. floating IPs)	10.1.2.10 - 10.1.3.222	469	255.255.254.0
Provider network (infrastructure)	10.10.3.10 - 10.10.3.250	241	255.255.252.0

3.7. CREATING A NETWORK

Create a network so that your instances can communicate with each other and receive IP addresses using DHCP. For more information about external network connections, see *Bridging the physical network*.

When creating networks, it is important to know that networks can host multiple subnets. This is useful if you intend to host distinctly different systems in the same network, and prefer a measure of isolation between them. For example, you can designate that only webserver traffic is present on one subnet, while database traffic traverses another. Subnets are isolated from each other, and any instance that wants to communicate with another subnet must have their traffic directed by a router. Consider placing systems that require a high volume of traffic amongst themselves in the same subnet, so that they do not require routing, and can avoid the subsequent latency and load.

1. In the dashboard, select **Project > Network > Networks**
2. Click **+Create Network** and specify the following values:

Field	Description
Network Name	Descriptive name, based on the role that the network will perform. If you are integrating the network with an external VLAN, consider appending the VLAN ID number to the name. For example, **webservers_122**, if you are hosting HTTP web servers in this subnet, and your VLAN tag is **122**. Or you might use **internal-only** if you intend to keep the network traffic private, and not integrate the network with an external network.

Field	Description
Admin State	Controls whether the network is immediately available. Use this field to create the network in a Down state, where it is logically present but inactive. This is useful if you do not intend to enter the network into production immediately.
Create Subnet	Determines whether to create a subnet. For example, you might not want to create a subnet if you intend to keep this network as a placeholder without network connectivity.

3. Click the **Next** button, and specify the following values in the **Subnet** tab:

Field	Description
Subnet Name	Enter a descriptive name for the subnet.
Network Address	Enter the address in CIDR format, which contains the IP address range and subnet mask in one value. To determine the address, calculate the number of bits masked in the subnet mask and append that value to the IP address range. For example, the subnet mask 255.255.255.0 has 24 masked bits. To use this mask with the IPv4 address range 192.168.122.0, specify the address 192.168.122.0/24.
IP Version	Specifies the internet protocol version, where valid types are IPv4 or IPv6. The IP address range in the Network Address field must match whichever version you select.
Gateway IP	IP address of the router interface for your default gateway. This address is the next hop for routing any traffic destined for an external location, and must be within the range that you specify in the Network Address field. For example, if your CIDR network address is 192.168.122.0/24, then your default gateway is likely to be 192.168.122.1.
Disable Gateway	Disables forwarding and isolates the subnet.

4. Click **Next** to specify **DHCP** options:

- **Enable DHCP** - Enables DHCP services for this subnet. You can use DHCP to automate the distribution of IP settings to your instances.

- **IPv6 Address** - Configuration Modes. If you create an IPv6 network, you must specify how to allocate IPv6 addresses and additional information:

    - **No Options Specified** - Select this option if you want to set IP addresses manually, or if you use a non OpenStack-aware method for address allocation.
    - **SLAAC (Stateless Address Autoconfiguration)** - Instances generate IPv6 addresses based on Router Advertisement (RA) messages sent from the OpenStack Networking router. Use this configuration to create an OpenStack Networking subnet with ra_mode set to slaac and address_mode set to slaac.
    - **DHCPv6 stateful** - Instances receive IPv6 addresses as well as additional options (for example, DNS) from the OpenStack Networking DHCPv6 service. Use this configuration to create a subnet with ra_mode set to dhcpv6-stateful and address_mode set to dhcpv6-stateful.
    - **DHCPv6 stateless** - Instances generate IPv6 addresses based on Router Advertisement (RA) messages sent from the OpenStack Networking router. Additional options (for example, DNS) are allocated from the OpenStack Networking DHCPv6 service. Use this configuration to create a subnet with ra_mode set to dhcpv6-stateless and address_mode set to dhcpv6-stateless.

- **Allocation Pools** - Range of IP addresses that you want DHCP to assign. For example, the value 192.168.22.100,192.168.22.150 considers all up addresses in that range as available for allocation.
- **DNS Name Servers** - IP addresses of the DNS servers available on the network. DHCP distributes these addresses to the instances for name resolution.

  IMPORTANT

  For strategic services such as DNS, it is a best practice not to host them on your cloud. For example, if your cloud hosts DNS and your cloud becomes inoperable, DNS is unavailable and the cloud components cannot do lookups on each other.

- **Host Routes** - Static host routes. First, specify the destination network in CIDR format, followed by the next hop that you want to use for routing (for example, 192.168.23.0/24, 10.1.31.1). Provide this value if you need to distribute static routes to instances.

5. Click **Create**.
   You can view the complete network in the **Networks** tab. You can also click **Edit** to change any options as needed. When you create instances, you can configure them now to use its subnet, and they receive any specified DHCP options.

3.8. WORKING WITH SUBNETS

Use subnets to grant network connectivity to instances. Each instance is assigned to a subnet as part of the instance creation process, therefore it’s important to consider proper placement of instances to best accommodate their connectivity requirements.

You can create subnets only in pre-existing networks. Remember that project networks in OpenStack Networking can host multiple subnets. This is useful if you intend to host distinctly different systems in the same network, and prefer a measure of isolation between them.

For example, you can designate that only webserver traffic is present on one subnet, while database traffic traverse another.

Subnets are isolated from each other, and any instance that wants to communicate with another subnet must have their traffic directed by a router. Therefore, you can lessen network latency and load by grouping systems in the same subnet that require a high volume of traffic between each other.

3.9. CREATING A SUBNET

To create a subnet, follow these steps:

1. In the dashboard, select **Project > Network > Networks**, and click the name of your network in the **Networks** view.

2. Click **Create Subnet**, and specify the following values:

Field	Description
Subnet Name	Descriptive subnet name.
Network Address	Address in CIDR format, which contains the IP address range and subnet mask in one value. To determine the CIDR address, calculate the number of bits masked in the subnet mask and append that value to the IP address range. For example, the subnet mask 255.255.255.0 has 24 masked bits. To use this mask with the IPv4 address range 192.168.122.0, specify the address 192.168.122.0/24.
IP Version	Internet protocol version, where valid types are IPv4 or IPv6. The IP address range in the Network Address field must match whichever protocol version you select.
Gateway IP	IP address of the router interface for your default gateway. This address is the next hop for routing any traffic destined for an external location, and must be within the range that you specify in the Network Address field. For example, if your CIDR network address is 192.168.122.0/24, then your default gateway is likely to be 192.168.122.1.
Disable Gateway	Disables forwarding and isolates the subnet.

3. Click **Next** to specify **DHCP** options:

   - **Enable DHCP** - Enables DHCP services for this subnet. You can use DHCP to automate the distribution of IP settings to your instances.

   - **IPv6 Address** - Configuration Modes. If you create an IPv6 network, you must specify how to allocate IPv6 addresses and additional information:

     - **No Options Specified** - Select this option if you want to set IP addresses manually, or if you use a non OpenStack-aware method for address allocation.

  - **SLAAC (Stateless Address Autoconfiguration)** - Instances generate IPv6 addresses based on Router Advertisement (RA) messages sent from the OpenStack Networking router. Use this configuration to create an OpenStack Networking subnet with ra_mode set to slaac and address_mode set to slaac.

  - **DHCPv6 stateful** - Instances receive IPv6 addresses as well as additional options (for example, DNS) from the OpenStack Networking DHCPv6 service. Use this configuration to create a subnet with ra_mode set to dhcpv6-stateful and address_mode set to dhcpv6-stateful.

  - **DHCPv6 stateless** - Instances generate IPv6 addresses based on Router Advertisement (RA) messages sent from the OpenStack Networking router. Additional options (for example, DNS) are allocated from the OpenStack Networking DHCPv6 service. Use this configuration to create a subnet with ra_mode set to dhcpv6-stateless and address_mode set to dhcpv6-stateless.

- **Allocation Pools** - Range of IP addresses that you want DHCP to assign. For example, the value 192.168.22.100,192.168.22.150 considers all up addresses in that range as available for allocation.

- **DNS Name Servers** - IP addresses of the DNS servers available on the network. DHCP distributes these addresses to the instances for name resolution.

- **Host Routes** - Static host routes. First, specify the destination network in CIDR format, followed by the next hop that you want to use for routing (for example, 192.168.23.0/24, 10.1.31.1). Provide this value if you need to distribute static routes to instances.

4. Click **Create**.
   You can view the subnet in the **Subnets** list. You can also click **Edit** to change any options as needed. When you create instances, you can configure them now to use its subnet, and they receive any specified DHCP options.

3.10. ADDING A ROUTER

OpenStack Networking provides routing services using an SDN-based virtual router. Routers are a requirement for your instances to communicate with external subnets, including those in the physical network. Routers and subnets connect using interfaces, with each subnet requiring its own interface to the router.

The default gateway of a router defines the next hop for any traffic received by the router. Its network is typically configured to route traffic to the external physical network using a virtual bridge.

To create a router, complete the following steps:

1. In the dashboard, select **Project > Network > Routers**, and click **Create Router**.

2. Enter a descriptive name for the new router, and click **Create router**.

3. Click **Set Gateway** next to the entry for the new router in the **Routers** list.

4. In the **External Network** list, specify the network that you want to receive traffic destined for an external location.

5. Click **Set Gateway**.
   After you add a router, you must configure any subnets you have created to send traffic using this router. You do this by creating interfaces between the subnet and the router.

IMPORTANT

The default routes for subnets must not be overwritten. When the default route for a subnet is removed, the L3 agent automatically removes the corresponding route in the router namespace too, and network traffic cannot flow to and from the associated subnet. If the existing router namespace route has been removed, to fix this problem, perform these steps:

1. Disassociate all floating IPs on the subnet.
2. Detach the router from the subnet.
3. Re-attach the router to the subnet.
4. Re-attach all floating IPs.

3.11. PURGING ALL RESOURCES AND DELETING A PROJECT

Use the **openstack project purge** command to delete all resources that belong to a particular project as well as deleting the project, too.

For example, to purge the resources of the **test-project** project, and then delete the project, run the following commands:

```
# openstack project list
+----------------------------------+--------------+
| ID                               | Name         |
+----------------------------------+--------------+
| 02e501908c5b438dbc73536c10c9aac0 | test-project |
| 519e6344f82e4c079c8e2eabb690023b | services     |
| 80bf5732752a41128e612fe615c886c6 | demo         |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin        |
+----------------------------------+--------------+

# openstack project purge --project 02e501908c5b438dbc73536c10c9aac0
```

3.12. DELETING A ROUTER

You can delete a router if it has no connected interfaces.

To remove its interfaces and delete a router, complete the following steps:

1. In the dashboard, select **Project > Network > Routers**, and click the name of the router that you want to delete.
2. Select the interfaces of type **Internal Interface**, and click **Delete Interfaces**.
3. From the Routers list, select the target router and click **Delete Routers**.

3.13. DELETING A SUBNET

You can delete a subnet if it is no longer in use. However, if any instances are still configured to use the subnet, the deletion attempt fails and the dashboard displays an error message.

Complete the following steps to delete a specific subnet in a network:

1. In the dashboard, select **Project > Network > Networks**
2. Click the name of your network.
3. Select the target subnet, and click **Delete Subnets**.

3.14. DELETING A NETWORK

There are occasions where it becomes necessary to delete a network that was previously created, perhaps as housekeeping or as part of a decommissioning process. You must first remove or detach any interfaces where the network is still in use, before you can successfully delete a network.

To delete a network in your project, together with any dependent interfaces, complete the following steps:

1. In the dashboard, select **Project > Network > Networks**
   Remove all router interfaces associated with the target network subnets.

   To remove an interface, find the ID number of the network that you want to delete by clicking on your target network in the **Networks** list, and looking at the ID field. All the subnets associated with the network share this value in the **Network ID** field.

2. Navigate to **Project > Network > Routers**, click the name of your virtual router in the **Routers** list, and locate the interface attached to the subnet that you want to delete.
   You can distinguish this subnet from the other subnets by the IP address that served as the gateway IP. You can further validate the distinction by ensuring that the network ID of the interface matches the ID that you noted in the previous step.

3. Click the **Delete Interface** button for the interface that you want to delete.

4. Select **Project > Network > Networks**, and click the name of your network.

5. Click the **Delete Subnet** button for the subnet that you want to delete.

   NOTE

   If you are still unable to remove the subnet at this point, ensure it is not already being used by any instances.

6. Select **Project > Network > Networks**, and select the network you would like to delete.

7. Click **Delete Networks**.

CHAPTER 4. CONNECTING VM INSTANCES TO PHYSICAL NETWORKS

You can directly connect your VM instances to an external network using flat and VLAN provider networks.

4.1. OVERVIEW OF THE OPENSTACK NETWORKING TOPOLOGY

OpenStack Networking (neutron) has two categories of services distributed across a number of node types.

- **Neutron server** - This service runs the OpenStack Networking API server, which provides the API for end-users and services to interact with OpenStack Networking. This server also integrates with the underlying database to store and retrieve project network, router, and loadbalancer details, among others.
- **Neutron agents** - These are the services that perform the network functions for OpenStack Networking:
    - **neutron-dhcp-agent** - manages DHCP IP addressing for project private networks.
    - **neutron-l3-agent** - performs layer 3 routing between project private networks, the external network, and others.
- **Compute node** - This node hosts the hypervisor that runs the virtual machines, also known as instances. A Compute node must be wired directly to the network in order to provide external connectivity for instances. This node is typically where the l2 agents run, such as **neutron-openvswitch-agent**.

Additional resources

- Section 4.2, “Placement of OpenStack Networking services”

4.2. PLACEMENT OF OPENSTACK NETWORKING SERVICES

The OpenStack Networking services can either run together on the same physical server, or on separate dedicated servers, which are named according to their roles:

- *Controller node* - The server that runs API service.
- *Network node* - The server that runs the OpenStack Networking agents.
- *Compute node* - The hypervisor server that hosts the instances.

The steps in this chapter apply to an environment that contains these three node types. If your deployment has both the Controller and Network node roles on the same physical node, then you must perform the steps from both sections on that server. This also applies for a High Availability (HA) environment, where all three nodes might be running the Controller node and Network node services with HA. As a result, you must complete the steps in sections applicable to Controller and Network nodes on all three nodes.

Additional resources

- Section 4.1, “Overview of the OpenStack Networking topology”

4.3. CONFIGURING FLAT PROVIDER NETWORKS

You can use flat provider networks to connect instances directly to the external network. This is useful if you have multiple physical networks and separate physical interfaces, and intend to connect each Compute and Network node to those external networks.

Prerequisites

- You have multiple physical networks.
  This example uses physical networks called **physnet1**, and **physnet2**, respectively.

- You have separate physical interfaces.
  This example uses separate physical interfaces, **eth0** and **eth1**, respectively.

Procedure

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-modules-environment.yaml
   ```

   TIP

   The Red Hat OpenStack Platform Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file* , which is a special type of template that provides customization for your orchestration templates.

2. In the YAML environment file under **parameter_defaults**, use the **NeutronBridgeMappings** to specify which OVS bridges are used for accessing external networks.

   Example

   ```
   parameter_defaults:
     NeutronBridgeMappings: 'physnet1:br-net1,physnet2:br-net2'
   ```

3. In the custom NIC configuration template for the Controller and Compute nodes, configure the bridges with interfaces attached.

   Example

   ```
   ...
                 - type: ovs_bridge
                   name: br-net1
                   mtu: 1500
                   use_dhcp: false
                   members:
                   - type: interface
                     name: eth0
                     mtu: 1500
                     use_dhcp: false
                     primary: true
   ```

```
              - type: ovs_bridge
                name: br-net2
                mtu: 1500
                use_dhcp: false
                members:
                - type: interface
                  name: eth1
                  mtu: 1500
                  use_dhcp: false
                  primary: true
...
```

4. Run the **openstack overcloud deploy** command and include the templates and the environment files, including this modified custom NIC template and the new environment file.

> **IMPORTANT**
>
> The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```

Verification

1. Create an external network (**public1**) as a flat network and associate it with the configured physical network (**physnet1**).
   Configure it as a shared network (using **--share**) to let other users create VM instances that connect to the external network directly.

Example

```
# openstack network create --share --provider-network-type flat --provider-physical-network
physnet1 --external public01
```

2. Create a subnet (**public_subnet**) using the **openstack subnet create** command.

Example

```
# openstack subnet create --no-dhcp --allocation-pool
start=192.168.100.20,end=192.168.100.100 --gateway 192.168.100.1 --network public01
public_subnet
```

3. Create a VM instance and connect it directly to the newly-created external network.

Example

```
$ openstack server create --image rhel --flavor my_flavor --network public01 my_instance
```

Additional resources

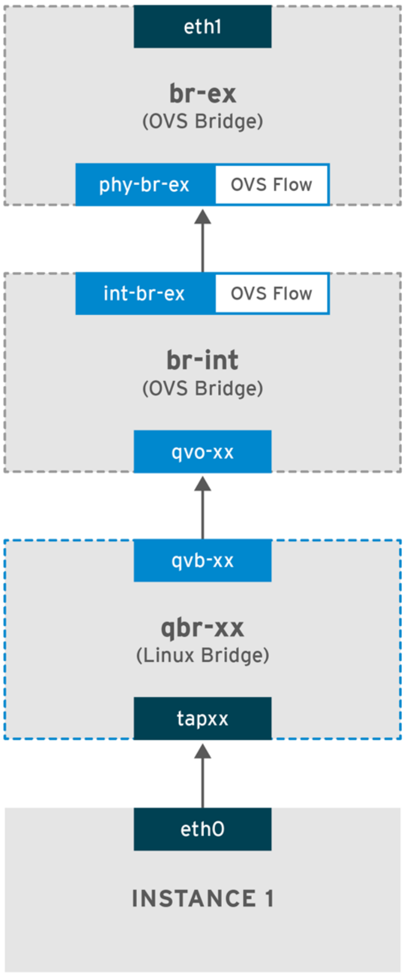
- Environment files in the *Installing and managing Red Hat OpenStack Platform with director* guide
- Including environment files in overcloud creation in the *Installing and managing Red Hat OpenStack Platform with director* guide
- network create in the *Command line interface reference*
- subnet create in the *Command line interface reference*
- server create in the *Command line interface reference*

4.4. HOW DOES THE FLAT PROVIDER NETWORK PACKET FLOW WORK?

This section describes in detail how traffic flows to and from an instance with flat provider network configuration.

The flow of outgoing traffic in a flat provider network

The following diagram describes the packet flow for traffic leaving an instance and arriving directly at an external network. After you configure the **br-ex** external bridge, add the physical interface to the bridge, and spawn an instance to a Compute node, the resulting configuration of interfaces and bridges resembles the configuration in the following diagram (if using the **iptables_hybrid** firewall driver):



OPENSTACK_450456_0617

1. Packets leave the **eth0** interface of the instance and arrive at the linux bridge **qbr-xx**.

2. Bridge **qbr-xx** is connected to **br-int** using veth pair **qvb-xx <-> qvo-xxx**. This is because the bridge is used to apply the inbound/outbound firewall rules defined by the security group.

3. Interface **qvb-xx** is connected to the **qbr-xx** linux bridge, and **qvoxx** is connected to the **br-int** Open vSwitch (OVS) bridge.

An example configuration of `qbr-xx`Linux bridge:

```
 # brctl show
qbr269d4d73-e7  8000.061943266ebb no  qvb269d4d73-e7
      tap269d4d73-e7
```

The configuration of **qvo-xx** on **br-int**:

```
# ovs-vsctl show
 Bridge br-int
     fail_mode: secure
        Interface "qvof63599ba-8f"
     Port "qvo269d4d73-e7"
        tag: 5
        Interface "qvo269d4d73-e7"
```

NOTE

Port **qvo-xx** is tagged with the internal VLAN tag associated with the flat provider network. In this example, the VLAN tag is **5**. When the packet reaches **qvo-xx**, the VLAN tag is appended to the packet header.

The packet is then moved to the **br-ex** OVS bridge using the patch-peer **int-br-ex <-> phy-br-ex**.

Example configuration of the patch-peer on **br-int**:

```
# ovs-vsctl show
   Bridge br-int
      fail_mode: secure
     Port int-br-ex
        Interface int-br-ex
           type: patch
           options: {peer=phy-br-ex}
```

Example configuration of the patch-peer on **br-ex**:

```
   Bridge br-ex
      Port phy-br-ex
         Interface phy-br-ex
            type: patch
            options: {peer=int-br-ex}
      Port br-ex
         Interface br-ex
            type: internal
```

When this packet reaches **phy-br-ex** on **br-ex**, an OVS flow inside **br-ex** strips the VLAN tag (5) and forwards it to the physical interface.

In the following example, the output shows the port number of **phy-br-ex** as **2**.

```
 # ovs-ofctl show br-ex
OFPT_FEATURES_REPLY (xid=0x2): dpid:00003440b5c90dc6
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE

 2(phy-br-ex): addr:ba:b5:7b:ae:5c:a2
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

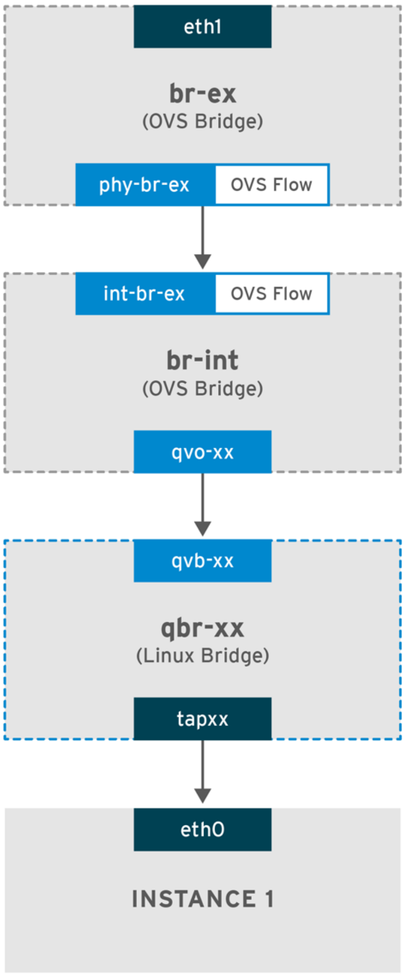
The following output shows any packet that arrives on **phy-br-ex** (**in_port=2**) with a VLAN tag of **5** (**dl_vlan=5**). In addition, an OVS flow in br-ex strips the VLAN tag and forwards the packet to the physical interface.

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4703.491s, table=0, n_packets=3620, n_bytes=333744, idle_age=0, priority=1
actions=NORMAL
 cookie=0x0, duration=3890.038s, table=0, n_packets=13, n_bytes=1714, idle_age=3764,
priority=4,in_port=2,dl_vlan=5 actions=strip_vlan,NORMAL
 cookie=0x0, duration=4702.644s, table=0, n_packets=10650, n_bytes=447632, idle_age=0,
priority=2,in_port=2 actions=drop
```

If the physical interface is another VLAN-tagged interface, then the physical interface adds the tag to the packet.

The flow of incoming traffic in a flat provider network

This section contains information about the flow of incoming traffic from the external network until it arrives at the interface of the instance.



OPENSTACK_450456_0617

1. Incoming traffic arrives at **eth1** on the physical node.

2. The packet passes to the **br-ex** bridge.

3. The packet moves to **br-int** via the patch-peer **phy-br-ex <--> int-br-ex**.

In the following example, **int-br-ex** uses port number **15**. See the entry containing **15(int-br-ex)**:

```
 # ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:00004e67212f644d
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 15(int-br-ex): addr:12:4e:44:a9:50:f4
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

Observing the traffic flow on br-int

1. When the packet arrives at **int-br-ex**, an OVS flow rule within the **br-int** bridge amends the packet to add the internal VLAN tag **5**. See the entry for **actions=mod_vlan_vid:5**:

```
 # ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5351.536s, table=0, n_packets=12118, n_bytes=510456, idle_age=0,
priority=1 actions=NORMAL
 cookie=0x0, duration=4537.553s, table=0, n_packets=3489, n_bytes=321696, idle_age=0,
priority=3,in_port=15,vlan_tci=0x0000 actions=mod_vlan_vid:5,NORMAL
 cookie=0x0, duration=5350.365s, table=0, n_packets=628, n_bytes=57892, idle_age=4538,
priority=2,in_port=15 actions=drop
 cookie=0x0, duration=5351.432s, table=23, n_packets=0, n_bytes=0, idle_age=5351,
priority=0 actions=drop
```

2. The second rule manages packets that arrive on int-br-ex (in_port=15) with no VLAN tag (vlan_tci=0x0000): This rule adds VLAN tag 5 to the packet (**actions=mod_vlan_vid:5,NORMAL**) and forwards it to **qvoxxx**.

3. **qvoxxx** accepts the packet and forwards it to **qvbxx**, after stripping away the VLAN tag.

4. The packet then reaches the instance.

NOTE

VLAN tag 5 is an example VLAN that was used on a test Compute node with a flat provider network; this value was assigned automatically by **neutron-openvswitch-agent**. This value may be different for your own flat provider network, and can differ for the same network on two separate Compute nodes.

Additional resources

- Section 4.5, “Troubleshooting instance-physical network connections on flat provider networks”

4.5. TROUBLESHOOTING INSTANCE-PHYSICAL NETWORK CONNECTIONS ON FLAT PROVIDER NETWORKS

The output provided in "How does the flat provider network packet flow work?" provides sufficient debugging information for troubleshooting a flat provider network, should anything go wrong. The following steps contain further information about the troubleshooting process.

Procedure

1. Review **bridge_mappings**.
Verify that the physical network name you use is consistent with the contents of the **bridge_mapping** configuration.

Example

In this example, the physical network name is, **physnet1**.

```
$ openstack network show provider-flat
```

Sample output

```
...
| provider:physical_network | physnet1
...
```

Example

In this example, the contents of the **bridge_mapping** configuration is also, **physnet1**:

```
$ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
```

Sample output

```
bridge_mappings = physnet1:br-ex
```

2. Review the network configuration.
   Confirm that the network is created as **external**, and uses the **flat** type:

Example

In this example, details about the network, **provider-flat**, is queried:

```
$ openstack network show provider-flat
```

Sample output

```
...
| provider:network_type     | flat                                 |
| router:external           | True                                 |
...
```

3. Review the patch-peer.
   Verify that **br-int** and **br-ex** are connected using a patch-peer **int-br-ex <--> phy-br-ex**.

```
$ ovs-vsctl show
```

Sample output

```
  Bridge br-int
      fail_mode: secure
     Port int-br-ex
         Interface int-br-ex
             type: patch
             options: {peer=phy-br-ex}
```

Sample output

Configuration of the patch-peer on **br-ex**:

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port br-ex
            Interface br-ex
                type: internal
```

This connection is created when you restart the **neutron-openvswitch-agent** service, if **bridge_mapping** is correctly configured in **/etc/neutron/plugins/ml2/openvswitch_agent.ini**.

Re-check the **bridge_mapping** setting if the connection is not created after you restart the service.

4. Review the network flows.
   Run **ovs-ofctl dump-flows br-ex** and **ovs-ofctl dump-flows br-int**, and review whether the flows strip the internal VLAN IDs for outgoing packets, and add VLAN IDs for incoming packets. This flow is first added when you spawn an instance to this network on a specific Compute node.

   a. If this flow is not created after spawning the instance, verify that the network is created as **flat**, is **external**, and that the **physical_network** name is correct. In addition, review the **bridge_mapping** settings.

   b. Finally, review the **ifcfg-br-ex** and **ifcfg-ethx** configuration. Ensure that **ethX** is added as a port within **br-ex**, and that **ifcfg-br-ex** and **ifcfg-ethx** have an **UP** flag in the output of **ip a**.

Sample output

The following output shows **eth1** is a port in **br-ex**:

```
Bridge br-ex
    Port phy-br-ex
        Interface phy-br-ex
            type: patch
            options: {peer=int-br-ex}
    Port "eth1"
        Interface "eth1"
```

Example

The following example demonstrates that **eth1** is configured as an OVS port, and that the kernel knows to transfer all packets from the interface, and send them to the OVS bridge **br-ex**. This can be observed in the entry, **master ovs-system**.

```
$ ip a
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP qlen 1000
```

Additional resources

- Section 4.4, “How does the flat provider network packet flow work?”
- Configuring bridge mappings

4.6. CONFIGURING VLAN PROVIDER NETWORKS

When you connect multiple VLAN-tagged interfaces on a single NIC to multiple provider networks, these new VLAN provider networks can connect VM instances directly to external networks.

Prerequisites

- You have a physical network, with a range of VLANs.
  This example uses a physical network called **physnet1**, with a range of VLANs, **171-172**.

- Your Network nodes and Compute nodes are connected to a physical network using a physical interface.
  This example uses Network nodes and Compute nodes that are connected to a physical network, **physnet1**, using a physical interface, **eth1**.
- The switch ports that these interfaces connect to must be configured to trunk the required VLAN ranges.

Procedure

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   #### Example

   ```
   $ vi /home/stack/templates/my-modules-environment.yaml
   ```

   #### TIP

   The Red Hat OpenStack Platform Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your orchestration templates.

2. In the YAML environment file under **parameter_defaults**, use **NeutronTypeDrivers** to specify your network type drivers.

   #### Example

   ```
   parameter_defaults:
     NeutronTypeDrivers: vxlan,flat,vlan
   ```

3. Configure the **NeutronNetworkVLANRanges** setting to reflect the physical network and VLAN ranges in use:

   #### Example

   ```
   parameter_defaults:
     NeutronTypeDrivers: 'vxlan,flat,vlan'
     NeutronNetworkVLANRanges: 'physnet1:171:172'
   ```

4. Create an external network bridge (*br-ex*), and associate a port (*eth1*) with it.
   This example configures *eth1* to use *br-ex*:

   #### Example

   ```
   parameter_defaults:
     NeutronTypeDrivers: 'vxlan,flat,vlan'
     NeutronNetworkVLANRanges: 'physnet1:171:172'
     NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-int'
   ```

5. Run the **openstack overcloud deploy** command and include the core templates and the environment files, including this new environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-environment.yaml
```

Verification

1. Create the external networks as type **vlan**, and associate them with the configured **physical_network**.
   Run the following example command to create two networks: one for VLAN 171, and another for VLAN 172:

   ### Example

   ```
   $ openstack network create \
     --provider-network-type vlan \
     --provider-physical-network physnet1 \
     --provider-segment 171 \
     provider-vlan171

   $ openstack network create \
     --provider-network-type vlan \
     --provider-physical-network physnet1 \
     --provider-segment 172 \
     provider-vlan172
   ```

2. Create a number of subnets and configure them to use the external network.
   You can use either **openstack subnet create** or the dashboard to create these subnets. Ensure that the external subnet details you have received from your network administrator are correctly associated with each VLAN.

   In this example, VLAN 171 uses subnet **10.65.217.0/24** and VLAN 172 uses **10.65.218.0/24**:

   ### Example

   ```
   $ openstack subnet create \
     --network provider-vlan171 \
     --subnet-range 10.65.217.0/24 \
     --dhcp \
     --gateway 10.65.217.254 \
     subnet-provider-171

   $ openstack subnet create \
     --network provider-vlan172 \
     --subnet-range 10.65.218.0/24 \
   ```

```
--dhcp \
--gateway 10.65.218.254 \
subnet-provider-172
```

Additional resources

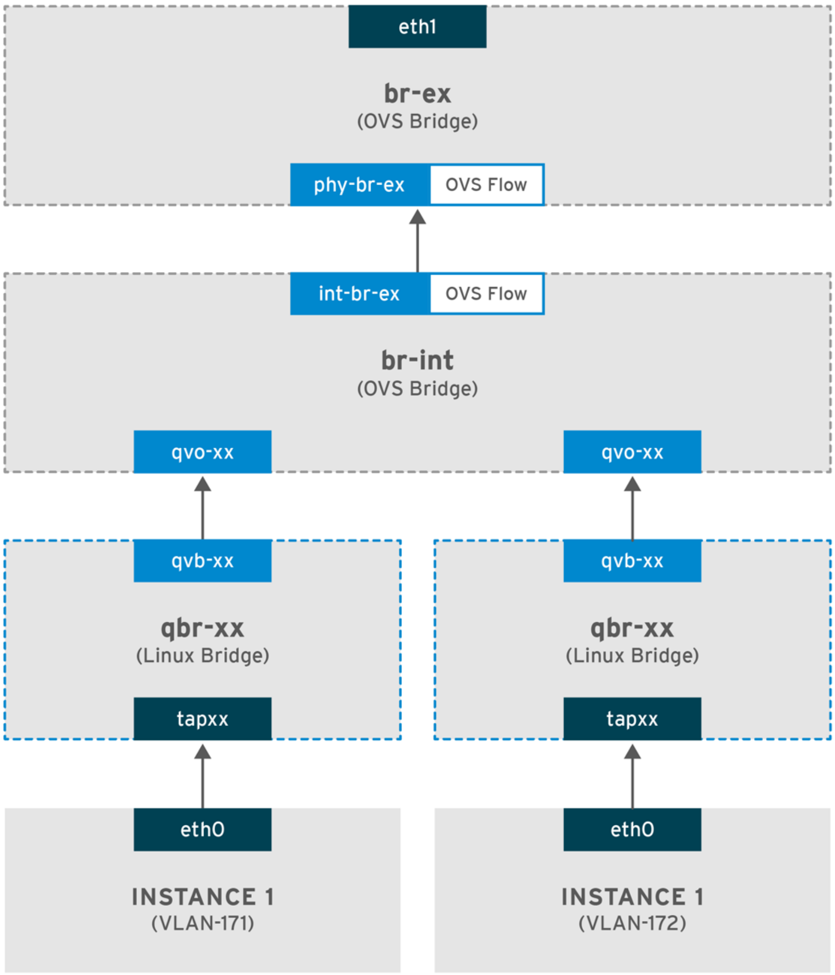
- Custom network interface templates in the *Installing and managing Red Hat OpenStack Platform with director* guide
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- network create in the *Command line interface reference*
- subnet create in the *Command line interface reference*

4.7. HOW DOES THE VLAN PROVIDER NETWORK PACKET FLOW WORK?

This section describes in detail how traffic flows to and from an instance with VLAN provider network configuration.

The flow of outgoing traffic in a VLAN provider network

The following diagram describes the packet flow for traffic leaving an instance and arriving directly to a VLAN provider external network. This example uses two instances attached to the two VLAN networks (171 and 172). After you configure *br-ex*, add a physical interface to it, and spawn an instance to a Compute node, the resulting configuration of interfaces and bridges resembles the configuration in the following diagram:



OPENSTACK_450456_0617

1. Packets leaving the *eth0* interface of the instance arrive at the linux bridge *qbr-xx* connected to the instance.

2. *qbr-xx* is connected to *br-int* using veth pair *qvbxx* <→ *qvoxxx*.

3. *qvbxx* is connected to the linux bridge *qbr-xx* and *qvoxx* is connected to the Open vSwitch bridge *br-int*.

Example configuration of *qbr-xx* on the Linux bridge.

This example features two instances and two corresponding linux bridges:

```
# brctl show
bridge name bridge id  STP enabled interfaces
qbr84878b78-63  8000.e6b3df9451e0 no  qvb84878b78-63
    tap84878b78-63

qbr86257b61-5d  8000.3a3c888eeae6 no  qvb86257b61-5d
    tap86257b61-5d
```

The configuration of *qvoxx* on *br-int*:

```
          options: {peer=phy-br-ex}
     Port "qvo86257b61-5d"
       tag: 3
```

```
            Interface "qvo86257b61-5d"
        Port "qvo84878b78-63"
            tag: 2
            Interface "qvo84878b78-63"
```

- **qvoxx** is tagged with the internal VLAN tag associated with the VLAN provider network. In this example, the internal VLAN tag 2 is associated with the VLAN provider network **provider-171** and VLAN tag 3 is associated with VLAN provider network **provider-172**. When the packet reaches *qvoxx*, the this VLAN tag is added to the packet header.

- The packet is then moved to the *br-ex* OVS bridge using patch-peer **int-br-ex** <→ **phy-br-ex**. Example patch-peer on *br-int*:

```
    Bridge br-int
        fail_mode: secure
       Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
```

Example configuration of the patch peer on *br-ex*:

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port br-ex
            Interface br-ex
                type: internal
```

- When this packet reaches *phy-br-ex* on *br-ex*, an OVS flow inside *br-ex* replaces the internal VLAN tag with the actual VLAN tag associated with the VLAN provider network.

The output of the following command shows that the port number of *phy-br-ex* is **4**:

```
# ovs-ofctl show br-ex
 4(phy-br-ex): addr:32:e7:a1:6b:90:3e
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

The following command shows any packet that arrives on phy-br-ex (**in_port=4**) which has VLAN tag 2 (**dl_vlan=2**). Open vSwitch replaces the VLAN tag with 171 (**actions=mod_vlan_vid:171,NORMAL**) and forwards the packet to the physical interface. The command also shows any packet that arrives on phy-br-ex (**in_port=4**) which has VLAN tag 3 (**dl_vlan=3**). Open vSwitch replaces the VLAN tag with 172 (**actions=mod_vlan_vid:172,NORMAL**) and forwards the packet to the physical interface. The neutron-openvswitch-agent adds these rules.

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6527.527s, table=0, n_packets=29211, n_bytes=2725576, idle_age=0,
priority=1 actions=NORMAL
```

```
 cookie=0x0, duration=2939.172s, table=0, n_packets=117, n_bytes=8296, idle_age=58,
priority=4,in_port=4,dl_vlan=3 actions=mod_vlan_vid:172,NORMAL
 cookie=0x0, duration=6111.389s, table=0, n_packets=145, n_bytes=9368, idle_age=98,
priority=4,in_port=4,dl_vlan=2 actions=mod_vlan_vid:171,NORMAL
 cookie=0x0, duration=6526.675s, table=0, n_packets=82, n_bytes=6700, idle_age=2462,
priority=2,in_port=4 actions=drop
```

- This packet is then forwarded to physical interface *eth1*.

The flow of incoming traffic in a VLAN provider network

The following example flow was tested on a Compute node using VLAN tag 2 for provider network provider-171 and VLAN tag 3 for provider network provider-172. The flow uses port 18 on the integration bridge br-int.

Your VLAN provider network may require a different configuration. Also, the configuration requirement for a network may differ between two different Compute nodes.

The output of the following command shows *int-br-ex* with port number 18:

```
# ovs-ofctl show br-int
 18(int-br-ex): addr:fe:b7:cb:03:c5:c1
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

The output of the following command shows the flow rules on br-int.

```
# ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6770.572s, table=0, n_packets=1239, n_bytes=127795, idle_age=106,
priority=1 actions=NORMAL

 cookie=0x0, duration=3181.679s, table=0, n_packets=2605, n_bytes=246456, idle_age=0,
 priority=3,in_port=18,dl_vlan=172 actions=mod_vlan_vid:3,NORMAL

 cookie=0x0, duration=6353.898s, table=0, n_packets=5077, n_bytes=482582, idle_age=0,
 priority=3,in_port=18,dl_vlan=171 actions=mod_vlan_vid:2,NORMAL

 cookie=0x0, duration=6769.391s, table=0, n_packets=22301, n_bytes=2013101, idle_age=0,
priority=2,in_port=18 actions=drop

 cookie=0x0, duration=6770.463s, table=23, n_packets=0, n_bytes=0, idle_age=6770, priority=0
actions=drop
```

Incoming flow example

This example demonstrates the following br-int OVS flow:

```
cookie=0x0, duration=3181.679s, table=0, n_packets=2605, n_bytes=246456, idle_age=0,
priority=3,in_port=18,dl_vlan=172 actions=mod_vlan_vid:3,NORMAL
```

- A packet with VLAN tag 172 from the external network reaches the *br-ex* bridge via *eth1* on the physical node.

- The packet moves to *br-int* via the patch-peer **phy-br-ex <-> int-br-ex**.

- The packet matches the flow’s criteria (**in_port=18,dl_vlan=172**).
- The flow actions (**actions=mod_vlan_vid:3,NORMAL**) replace the VLAN tag 172 with internal VLAN tag 3 and forwards the packet to the instance with normal Layer 2 processing.

Additional resources

- Section 4.4, “How does the flat provider network packet flow work?”

4.8. TROUBLESHOOTING INSTANCE-PHYSICAL NETWORK CONNECTIONS ON VLAN PROVIDER NETWORKS

Refer to the packet flow described in "How does the VLAN provider network packet flow work?" when troubleshooting connectivity in a VLAN provider network. In addition, review the following configuration options:

Procedure

1. Verify that physical network name used in the **bridge_mapping** configuration matches the physical network name.

   #### Example

   ```
   $ openstack network show provider-vlan171
   ```

   #### Sample output

   ```
   ...
   | provider:physical_network | physnet1
   ...
   ```

   #### Example

   ```
   $ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
   ```

   #### Sample output

   In this sample output, the physical network name, **physnet1**, matches the name used in the **bridge_mapping** configuration:

   ```
   bridge_mappings = physnet1:br-ex
   ```

2. Confirm that the network was created as **external**, is type **vlan**, and uses the correct **segmentation_id** value:

   #### Example

   ```
   $ openstack network show provider-vlan171
   ```

   #### Sample output

   ```
   ...
   ```

```
| provider:network_type      | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 171                                  |
...
```

3. Review the patch-peer.
   Verify that **br-int** and **br-ex** are connected using a patch-peer **int-br-ex <--> phy-br-ex**.

   ```
   $ ovs-vsctl show
   ```

   This connection is created while restarting **neutron-openvswitch-agent**, provided that the **bridge_mapping** is correctly configured in **/etc/neutron/plugins/ml2/openvswitch_agent.ini**.

   Recheck the **bridge_mapping** setting if this is not created even after restarting the service.

4. Review the network flows.

   a. To review the flow of outgoing packets, run **ovs-ofctl dump-flows br-ex** and **ovs-ofctl dump-flows br-int**, and verify that the flows map the internal VLAN IDs to the external VLAN ID (**segmentation_id**).

   b. For incoming packets, map the external VLAN ID to the internal VLAN ID.
   This flow is added by the neutron OVS agent when you spawn an instance to this network for the first time.

   c. If this flow is not created after spawning the instance, ensure that the network is created as **vlan**, is **external**, and that the **physical_network** name is correct. In addition, re-check the **bridge_mapping** settings.

   d. Finally, re-check the **ifcfg-br-ex** and **ifcfg-ethx** configuration.
   Ensure that **br-ex** includes port **ethX**, and that both **ifcfg-br-ex** and **ifcfg-ethx** have an **UP** flag in the output of the **ip a** command.

   Example

   ```
   $ ovs-vsctl show
   ```

   In this sample output, **eth1** is a port in **br-ex**:

   ```
       Bridge br-ex
           Port phy-br-ex
               Interface phy-br-ex
                   type: patch
                   options: {peer=int-br-ex}
           Port "eth1"
               Interface "eth1"
   ```

   Example

   ```
   $ ip a
   ```

   Sample output

In this sample output, **eth1** has been added as a port, and that the kernel is configured to move all packets from the interface to the OVS bridge **br-ex**. This is demonstrated by the entry, **master ovs-system**.

```
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP qlen 1000
```

Additional resources

- Section 4.7, "How does the VLAN provider network packet flow work?"

4.9. ENABLING MULTICAST SNOOPING FOR PROVIDER NETWORKS IN AN ML2/OVS DEPLOYMENT

To prevent flooding multicast packets to every port in a Red Hat OpenStack Platform (RHOSP) provider network, you must enable multicast snooping. In RHOSP deployments that use the Modular Layer 2 plug-in with the Open vSwitch mechanism driver (ML2/OVS), you do this by declaring the RHOSP Orchestration (heat) **NeutronEnableIgmpSnooping** parameter in a YAML-formatted environment file.

IMPORTANT

You should thoroughly test and understand any multicast snooping configuration before applying it to a production environment. Misconfiguration can break multicasting or cause erratic network behavior.

Prerequisites

- Your configuration must only use ML2/OVS provider networks.
- Your physical routers must also have IGMP snooping enabled.
  That is, the physical router must send IGMP query packets on the provider network to solicit regular IGMP reports from multicast group members to maintain the snooping cache in OVS (and for physical networking).
- An RHOSP Networking service security group rule must be in place to allow inbound IGMP to the VM instances (or port security disabled).
  In this example, a rule is created for the **ping_ssh** security group:

  #### Example

  ```
  $ openstack security group rule create --protocol igmp --ingress ping_ssh
  ```

Procedure

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   #### Example

   ```
   $ vi /home/stack/templates/my-ovs-environment.yaml
   ```

TIP

The Orchestration service (heat) uses a set of plans called templates to install and configure your environment. You can customize aspects of the overcloud with a custom environment file, which is a special type of template that provides customization for your heat templates.

2. In the YAML environment file under **parameter_defaults**, set **NeutronEnableIgmpSnooping** to true.

```
parameter_defaults:
  NeutronEnableIgmpSnooping: true
  ...
```

IMPORTANT

Ensure that you add a whitespace character between the colon (:) and **true**.

3. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-ovs-
environment.yaml
```

Verification

- Verify that the multicast snooping is enabled.

Example

```
# sudo ovs-vsctl list bridge br-int
```

Sample output

```
...
mcast_snooping_enable: true
...
other_config: {mac-table-size="50000", mcast-snooping-disable-flood-unregistered=True}
...
```

Additional resources

- Neutron in *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform*

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Networking (neutron) Parameters in the *Overcloud parameters* guide
- Creating a security group in the *Creating and managing instances* guide

4.10. ENABLING MULTICAST IN AN ML2/OVN DEPLOYMENT

To support multicast traffic, modify the deployment's security configuration to allow multicast traffic to reach the virtual machine (VM) instances in the multicast group. To prevent multicast traffic flooding, enable IGMP snooping.

IMPORTANT

Test and understand any multicast snooping configuration before applying it to a production environment. Misconfiguration can break multicasting or cause erratic network behavior.

Prerequisites

- An OpenStack deployment with the ML2/OVN mechanism driver.

Procedure

1. Configure security to allow multicast traffic to the appropriate VM instances. For instance, create a pair of security group rules to allow IGMP traffic from the IGMP querier to enter and exit the VM instances, and a third rule to allow multicast traffic.

   Example

   A security group *mySG* allows IGMP traffic to enter and exit the VM instances.

   ```
   openstack security group rule create --protocol igmp --ingress mySG

   openstack security group rule create --protocol igmp --egress mySG
   ```

   Another rule allows multicast traffic to reach VM instances.

   ```
   openstack security group rule create  --protocol udp mySG
   ```

   As an alternative to setting security group rules, some operators choose to selectively disable port security on the network. If you choose to disable port security, consider and plan for any related security risks.

2. Set the heat parameter **NeutronEnableIgmpSnooping: True** in an environment file on the undercloud node. For instance, add the following lines to ovn-extras.yaml.

   Example

   ```
   parameter_defaults:
       NeutronEnableIgmpSnooping: True
   ```

3. Include the environment file in the **openstack overcloud deploy** command with any other environment files that are relevant to your environment and deploy the overcloud.

```
$ openstack overcloud deploy \
--templates \
…
-e <other_overcloud_environment_files> \

-e ovn-extras.yaml \
…
```

Replace **<other_overcloud_environment_files>** with the list of environment files that are part of your existing deployment.

Verification

1. Verify that the multicast snooping is enabled. List the northbound database Logical_Switch table.

```
$ ovn-nbctl list Logical_Switch
```

Sample output

```
_uuid         : d6a2fbcd-aaa4-4b9e-8274-184238d66a15
other_config  : {mcast_flood_unregistered="false", mcast_snoop="true"}
...
```

The Networking Service (neutron) igmp_snooping_enable configuration is translated into the mcast_snoop option set in the other_config column of the Logical_Switch table in the OVN Northbound Database. Note that mcast_flood_unregistered is always “false”.

2. Show the IGMP groups.

```
$ ovn-sbctl list IGMP_group
```

Sample output

```
_uuid    : 2d6cae4c-bd82-4b31-9c63-2d17cbeadc4e
address  : "225.0.0.120"
chassis  : 34e25681-f73f-43ac-a3a4-7da2a710ecd3
datapath : eaf0f5cc-a2c8-4c30-8def-2bc1ec9dcabc
ports    : [5eaf9dd5-eae5-4749-ac60-4c1451901c56, 8a69efc5-38c5-48fb-bbab-
30f2bf9b8d45]
...
```

Additional resources

- Neutron in *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform*
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

4.11. ENABLING COMPUTE METADATA ACCESS

Instances connected as described in this chapter are directly attached to the provider external networks, and have external routers configured as their default gateway. No OpenStack Networking (neutron) routers are used. This means that *neutron* routers cannot be used to proxy metadata requests from instances to the *nova-metadata* server, which may result in failures while running *cloud-init*. However, this issue can be resolved by configuring the dhcp agent to proxy metadata requests. You can enable this functionality in **/etc/neutron/dhcp_agent.ini**. For example:

```
enable_isolated_metadata = True
```

4.12. FLOATING IP ADDRESSES

You can use the same network to allocate floating IP addresses to instances, even if the floating IPs are already associated with private networks. The addresses that you allocate as floating IPs from this network are bound to the *qrouter-xxx* namespace on the Network node, and perform *DNAT-SNAT* to the associated private IP address. In contrast, the IP addresses that you allocate for direct external network access are bound directly inside the instance, and allow the instance to communicate directly with external network.

CHAPTER 5. MANAGING FLOATING IP ADDRESSES

In addition to a having a private, fixed IP address, VM instances can have a public, or floating IP address to communicate with other networks. The information in this section describes how to create and manage floating IPs with the Red Hat OpenStack Platform (RHOSP) Networking service (neutron).

5.1. CREATING FLOATING IP POOLS

You can use floating IP addresses to direct ingress network traffic to your OpenStack instances. First, you must define a pool of validly routable external IP addresses, which you can then assign to instances dynamically. OpenStack Networking routes all incoming traffic destined for that floating IP to the instance that you associate with the floating IP.

> NOTE
>
> OpenStack Networking allocates floating IP addresses to all projects (tenants) from the same IP ranges in CIDR format. As a result, all projects can consume floating IPs from every floating IP subnet. You can manage this behavior using quotas for specific projects. For example, you can set the default to **10** for **ProjectA** and **ProjectB**, while setting the quota for **ProjectC** to **0**.

Procedure

- When you create an external subnet, you can also define the floating IP allocation pool.

  ```
  $ openstack subnet create --no-dhcp --allocation-pool
  start=IP_ADDRESS,end=IP_ADDRESS --gateway IP_ADDRESS --network
  SUBNET_RANGE NETWORK_NAME
  ```

  If the subnet hosts only floating IP addresses, consider disabling DHCP allocation with the **--no-dhcp** option in the **openstack subnet create** command.

  ### Example

  ```
  $ openstack subnet create --no-dhcp --allocation_pool
  start=192.168.100.20,end=192.168.100.100 --gateway 192.168.100.1 --network
  192.168.100.0/24 public
  ```

Verification

- You can verify that the pool is configured properly by assigning a random floating IP to an instance. (See the later link that follows.)

Additional resources

- subnet create in the *Command line interface reference*
- Assigning a random floating IP

5.2. ASSIGNING A SPECIFIC FLOATING IP

You can assign a specific floating IP address to a VM instance.

Procedure

- Allocate a floating IP address to an instance by using the **openstack server add floating ip** command.

  #### Example

  ```
  $ openstack server add floating ip prod-serv1 192.0.2.200
  ```

Validation steps

- Confirm that your floating IP is associated with your instance by using the **openstack server show** command.

  #### Example

  ```
  $ openstack server show prod-serv1
  ```

  #### Sample output

  ```
  +-----------------------------+------------------------------------------+
  | Field                       | Value                                    |
  +-----------------------------+------------------------------------------+
  | OS-DCF:diskConfig           | MANUAL                                   |
  | OS-EXT-AZ:availability_zone | nova                                     |
  | OS-EXT-STS:power_state      | Running                                  |
  | OS-EXT-STS:task_state       | None                                     |
  | OS-EXT-STS:vm_state         | active                                   |
  | OS-SRV-USG:launched_at      | 2021-08-11T14:45:37.000000               |
  | OS-SRV-USG:terminated_at    | None                                     |
  | accessIPv4                  |                                          |
  | accessIPv6                  |                                          |
  | addresses                   | public=198.51.100.56,192.0.2.200         |
  |                             |                                          |
  | config_drive                |                                          |
  | created                     | 2021-08-11T14:44:54Z                     |
  | flavor                      | review-ephemeral                         |
  |                             | (8130dd45-78f6-44dc-8173-4d6426b8e520)   |
  | hostId                      | 2308c8d8f60ed5394b1525122fb5bf8ea55c78b8 |
  |                             | 0ec6157eca4488c9                         |
  | id                          | aef3ca09-887d-4d20-872d-1d1b49081958     |
  | image                       | rhel8                                    |
  |                             | (20724bfe-93a9-4341-a5a3-78b37b3a5dfb)   |
  | key_name                    | example-keypair                          |
  | name                        | prod-serv1                               |
  | progress                    | 0                                        |
  | project_id                  | bd7a8c4a19424cf09a82627566b434fa         |
  | properties                  |                                          |
  | security_groups             | name='default'                           |
  | status                      | ACTIVE                                   |
  | updated                     | 2021-08-11T14:45:37Z                     |
  | user_id                     | 4b7e19a0d723310fd92911eb2fe59743a3a5cd32 |
  |                             | 45f76ffced91096196f646b5                 |
  | volumes_attached            |                                          |
  +-----------------------------+------------------------------------------+
  ```

Additional resources

- server add floating ip in the *Command line interface reference*
- server show in the *Command line interface reference*
- Assigning a random floating IP

5.3. CREATING AN ADVANCED NETWORK

Advanced network options are available for administrators, when creating a network in the Dashboard from the **Admin** view. Use these options to specify projects and to define the network type that you want to use.

Procedure

1. In the dashboard, select **Admin** > **Networks** > **Create Network** > **Project**.
2. Select the project that you want to host the new network with the **Project** drop-down list.
3. Review the options in **Provider Network Type**:
    - **Local** - Traffic remains on the local Compute host and is effectively isolated from any external networks.
    - **Flat** - Traffic remains on a single network and can also be shared with the host. No VLAN tagging or other network segregation takes place.
    - **VLAN** - Create a network using a VLAN ID that corresponds to a VLAN present in the physical network. This option allows instances to communicate with systems on the same layer 2 VLAN.
    - **GRE** - Use a network overlay that spans multiple nodes for private communication between instances. Traffic egressing the overlay must be routed.
    - **VXLAN** - Similar to GRE, and uses a network overlay to span multiple nodes for private communication between instances. Traffic egressing the overlay must be routed.
4. Click **Create Network**.
   Review the Project Network Topology to validate that the network has been successfully created.

Additional resources

- Assigning a specific floating IP
- Assigning a random floating IP

5.4. ASSIGNING A RANDOM FLOATING IP

You can dynamically allocate floating IP addresses to VM instances from a pool of external IP addresses.

Prerequisites

- A pool of routable external IP addresses.
  For more information, see Section 5.1, “Creating floating IP pools” .

Procedure

1. Enter the following command to allocate a floating IP address from the pool. In this example, the network is named **public**.

   #### Example

   ```
   $ openstack floating ip create public
   ```

   #### Sample output

   In the following example, the newly allocated floating IP is **192.0.2.200**. You can assign it to an instance.

   ```
   +---------------------+--------------------------------------------------+
   | Field               | Value                                            |
   +---------------------+--------------------------------------------------+
   | fixed_ip_address    | None                                             |
   | floating_ip_address | 192.0.2.200                                      |
   | floating_network_id | f0dcc603-f693-4258-a940-0a31fd4b80d9             |
   | id                  | 6352284c-c5df-4792-b168-e6f6348e2620             |
   | port_id             | None                                             |
   | router_id           | None                                             |
   | status              | ACTIVE                                           |
   +---------------------+--------------------------------------------------+
   ```

2. Enter the following command to locate your instance:

   ```
   $ openstack server list
   ```

   #### Sample output

   ```
   +-------------+-------------+--------+-------------+-------+-------------+
   | ID          | Name        | Status | Networks    | Image | Flavor      |
   +-------------+-------------+--------+-------------+-------+-------------+
   | aef3ca09-88 | prod-serv1  | ACTIVE | public=198. | rhel8 | review-     |
   | 7d-4d20-872 |             |        | 51.100.56   |       | ephemeral   |
   | d-1d1b49081 |             |        |             |       |             |
   | 958         |             |        |             |       |             |
   |             |             |        |             |       |             |
   +-------------+-------------+--------+-------------+-------+-------------+
   ```

3. Associate the instance name or ID with the floating IP.

   #### Example

   ```
   $ openstack server add floating ip prod-serv1 192.0.2.200
   ```

Validation steps

- Enter the following command to confirm that your floating IP is associated with your instance.

  Example

  ```
  $ openstack server show prod-serv1
  ```

  Sample output

  ```
  +-----------------------------+------------------------------------------+
  | Field                       | Value                                    |
  +-----------------------------+------------------------------------------+
  | OS-DCF:diskConfig           | MANUAL                                   |
  | OS-EXT-AZ:availability_zone | nova                                     |
  | OS-EXT-STS:power_state      | Running                                  |
  | OS-EXT-STS:task_state       | None                                     |
  | OS-EXT-STS:vm_state         | active                                   |
  | OS-SRV-USG:launched_at      | 2021-08-11T14:45:37.000000               |
  | OS-SRV-USG:terminated_at    | None                                     |
  | accessIPv4                  |                                          |
  | accessIPv6                  |                                          |
  | addresses                   | public=198.51.100.56,192.0.2.200         |
  |                             |                                          |
  | config_drive                |                                          |
  | created                     | 2021-08-11T14:44:54Z                     |
  | flavor                      | review-ephemeral                         |
  |                             | (8130dd45-78f6-44dc-8173-4d6426b8e520)   |
  | hostId                      | 2308c8d8f60ed5394b1525122fb5bf8ea55c78b8 |
  |                             | 0ec6157eca4488c9                         |
  | id                          | aef3ca09-887d-4d20-872d-1d1b49081958     |
  | image                       | rhel8                                    |
  |                             | (20724bfe-93a9-4341-a5a3-78b37b3a5dfb)   |
  | key_name                    | example-keypair                          |
  | name                        | prod-serv1                               |
  | progress                    | 0                                        |
  | project_id                  | bd7a8c4a19424cf09a82627566b434fa         |
  | properties                  |                                          |
  | security_groups             | name='default'                           |
  | status                      | ACTIVE                                   |
  | updated                     | 2021-08-11T14:45:37Z                     |
  | user_id                     | 4b7e19a0d723310fd92911eb2fe59743a3a5cd32 |
  |                             | 45f76ffced91096196f646b5                 |
  | volumes_attached            |                                          |
  +-----------------------------+------------------------------------------+
  ```

Additional resources

- floating ip create in the *Command line interface reference*
- server add floating ip in the *Command line interface reference*
- server show in the *Command line interface reference*
- Creating floating IP pools

5.5. CREATING MULTIPLE FLOATING IP POOLS

OpenStack Networking supports one floating IP pool for each L3 agent. Therefore, you must scale your L3 agents to create additional floating IP pools.

Procedure

- Make sure that in **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf** the property **handle_internal_only_routers** is set to **True** for only one L3 agent in your environment. This option configures the L3 agent to manage only non-external routers.

Additional resources

- Creating floating IP pools
- Assigning a random floating IP

5.6. CONFIGURING FLOATING IP PORT FORWARDING

To enable users to set up port forwarding for floating IPs, you must enable the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) port_forwarding` service plug-in.

Prerequisites

- You must have RHOSP administrator privileges.
- The **port_forwarding** service plug-in requires that you also set the **router** service plug-in.

Procedure

1. Log in to the undercloud host as the stack user.
2. Source the stackrc undercloud credentials file:

```
$ source ~/stackrc
```

3. In a custom environment YAML file, set the **port_forwarding** service plug-in:

```
parameter_defaults:
  NeutronPluginExtensions: "router,port_forwarding"
```

> **NOTE**
>
> The **port_forwarding** service plug-in requires that you also set the **router** service plug-in.

4. If you use the ML2/OVS mechanism driver with the Networking service, you must also set the **port_forwarding** extension for the OVS L3 agent:

```
parameter_defaults:
  NeutronPluginExtensions: "router,port_forwarding"
  NeutronL3AgentExtensions: "port_forwarding"
```

5. Deploy your overcloud and include the core heat templates, environment files, and this new custom environment file.

   IMPORTANT

   The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

   ```
   $ openstack overcloud deploy --templates \
    -e <your_environment_files> \
    -e /home/stack/templates/my-environment.yaml
   ```

   RHOSP users can now set up port forwarding for floating IPs. For more information, see Section 5.7, “Creating port forwarding for a floating IP” .

Verification

1. Source the overcloud credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Ensure that the Networking service has successfully loaded the **port_forwarding** and **router** service plug-ins:

   ```
   $ openstack extension list --network -c Name -c Alias --max-width 74 | \
   grep -i -e 'Neutron L3 Router' -i -e floating-ip-port-forwarding
   ```

   Sample output

   A successful verification produces output similar to the following:

   ```
   | Floating IP Port Forwarding     | floating-ip-port-forwarding      |
   | Neutron L3 Router               | router                           |
   ```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

5.7. CREATING PORT FORWARDING FOR A FLOATING IP

You can use Red Hat OpenStack Platform Networking service (neutron) to set up port forwarding for a floating IP.

Prerequisites

- The Networking service must be running with the **port_forwarding** service plug-in loaded. For information, see Section 5.6, “Configuring floating IP port forwarding” .

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

2. Use the following command to create port forwarding for a floating IP:

   ```
   $ openstack floating ip port forwarding create \
   --internal-ip-address <internal-ip-address> \
   --port <port> \
   --internal-protocol-port <port-number> \
   --external-protocol-port <port-number> \
   --protocol <protocol> \
   <floating-ip>
   ```

   - Replace **<internal-ip-address>** with the internal, destination IP address.
     This is the IP address that is associated with the instance on which the application is running.

   - Replace **<port>** with the name or ID of the Networking service port to which the instance is attached.

   - Replace **<port-number>** in **--internal-protocol-port** with the internal, destination port number.
     This is the port number that the application running in the instance uses.

   - Replace **<port-number>** in **--external-protocol-port** with the external, source port number.
     This is the port number that the application running outside of your RHOSP cloud uses.

   - Replace **<protocol>** with the protocol, such as TCP or UDP, used by the application that receives the port-forwarded traffic.

   - Replace **<floating-ip>** with the floating IP whose specified port traffic you want to forward.

     #### Example

     This example creates port fowarding for an instance that is attached to the floating IP **198.51.100.47**. The floating IP uses the Networking service port **1adfdb09-e8c6-4708-b5aa-11f50fc22d62**. When the Networking service detects incoming, external traffic addressed to **198.51.100.47:80**, it forwards the traffic to the internal IP address, **203.0.113.107**, on TCP port, **8080**:

     ```
     $ openstack floating ip port forwarding create \
     --internal-ip-address 203.0.113.107 \
     --port 1adfdb09-e8c6-4708-b5aa-11f50fc22d62 \
     --internal-protocol-port 8080 \
     --external-protocol-port 80 \
     --protocol tcp \
     198.51.100.47
     ```

Verification

- Confirm that the Networking service has established forwarding for the floating IP port.

Example

The following example verifies successful port forwarding for the floating IP **198.51.100.47**:

```
$ openstack floating ip port forwarding list 198.51.100.47 --max-width 74
```

Sample output

The output shows that traffic sent to the floating IP **198.51.100.47** on TCP port 80 is forwarded to port **8080** on the instance with the internal address **203.0.113.107**:

```
+----------+------------------+---------------------+---------------+---------------+----------+-------------
+
| ID       | Internal Port ID | Internal IP Address | Internal Port | External Port | Protocol |
Description |
+----------+------------------+---------------------+---------------+---------------+----------+-------------
+
| 5cf204c7 | 1adfdb09-e8c6-47 | 203.0.113.107       |          8080 |            80 | tcp      |             |
| -6825-45 | 08-b5aa-11f50fc2 |                     |               |               |          |             |
| de-84ec- | 2d62             |                     |               |               |          |             |
| 2eb507be |                  |                     |               |               |          |             |
| 543e     |                  |                     |               |               |          |             |
+----------+------------------+---------------------+---------------+---------------+----------+-------------
+
```

Additional resources

- floating ip port forwarding create in the *Command line interface reference*

5.8. BRIDGING THE PHYSICAL NETWORK

Bridge your virtual network to the physical network to enable connectivity to and from virtual instances.

In this procedure, the example physical interface, **eth0**, is mapped to the bridge, **br-ex**; the virtual bridge acts as the intermediary between the physical network and any virtual networks.

As a result, all traffic traversing **eth0** uses the configured Open vSwitch to reach instances.

To map a physical NIC to the virtual Open vSwitch bridge, complete the following steps:

Procedure

1. Open **/etc/sysconfig/network-scripts/ifcfg-eth0** in a text editor, and update the following parameters with values appropriate for the network at your site:

   - IPADDR
   - NETMASK GATEWAY
   - DNS1 (name server)
     Here is an example:

     ```
     # vi /etc/sysconfig/network-scripts/ifcfg-eth0
     DEVICE=eth0
     TYPE=OVSPort
     ```

```
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

2. Open **/etc/sysconfig/network-scripts/ifcfg-br-ex** in a text editor and update the virtual bridge parameters with the IP address values that were previously allocated to eth0:

   ```
   # vi /etc/sysconfig/network-scripts/ifcfg-br-ex
   DEVICE=br-ex
   DEVICETYPE=ovs
   TYPE=OVSBridge
   BOOTPROTO=static
   IPADDR=192.168.120.10
   NETMASK=255.255.255.0
   GATEWAY=192.168.120.1
   DNS1=192.168.120.1
   ONBOOT=yes
   ```

   You can now assign floating IP addresses to instances and make them available to the physical network.

Additional resources

- Configuring bridge mappings

5.9. ADDING AN INTERFACE

You can use interfaces to interconnect routers with subnets so that routers can direct any traffic that instances send to destinations outside of their intermediate subnet.

To add a router interface and connect the new interface to a subnet, complete these steps:

> **NOTE**
>
> This procedure uses the Network Topology feature. Using this feature, you can see a graphical representation of all your virtual routers and networks while you to perform network management tasks.

1. In the dashboard, select **Project > Network > Network Topology**

2. Locate the router that you want to manage, hover your mouse over it, and click **Add Interface**.

3. Specify the Subnet that you want to connect to the router.
   You can also specify an IP address. The address is useful for testing and troubleshooting purposes, since a successful ping to this interface indicates that the traffic is routing as expected.

4. Click **Add interface**.
   The Network Topology diagram automatically updates to reflect the new interface connection between the router and subnet.

5.10. DELETING AN INTERFACE

You can remove an interface to a subnet if you no longer require the router to direct traffic for the subnet.

To delete an interface, complete the following steps:

1. In the dashboard, select **Project > Network > Routers**
2. Click the name of the router that hosts the interface that you want to delete.
3. Select the interface type (**Internal Interface**), and click **Delete Interfaces**.

CHAPTER 6. MONITORING AND TROUBLESHOOTING NETWORKS

The diagnostic process of monitoring and troubleshooting network connectivity in Red Hat OpenStack Platform is similar to the diagnostic process for physical networks. If you use VLANs, you can consider the virtual infrastructure as a trunked extension of the physical network, rather than a wholly separate environment. There are some differences between troubleshooting an ML2/OVS network and the default, ML2/OVN network.

6.1. BASIC PING TESTING

The **ping** command is a useful tool for analyzing network connectivity problems. The results serve as a basic indicator of network connectivity, but might not entirely exclude all connectivity issues, such as a firewall blocking the actual application traffic. The ping command sends traffic to specific destinations, and then reports back whether the attempts were successful.

NOTE

The ping command is an ICMP operation. To use **ping**, you must allow ICMP traffic to traverse any intermediary firewalls.

Ping tests are most useful when run from the machine experiencing network issues, so it may be necessary to connect to the command line via the VNC management console if the machine seems to be completely offline.

For example, the following ping test command validates multiple layers of network infrastructure in order to succeed; name resolution, IP routing, and network switching must all function correctly:

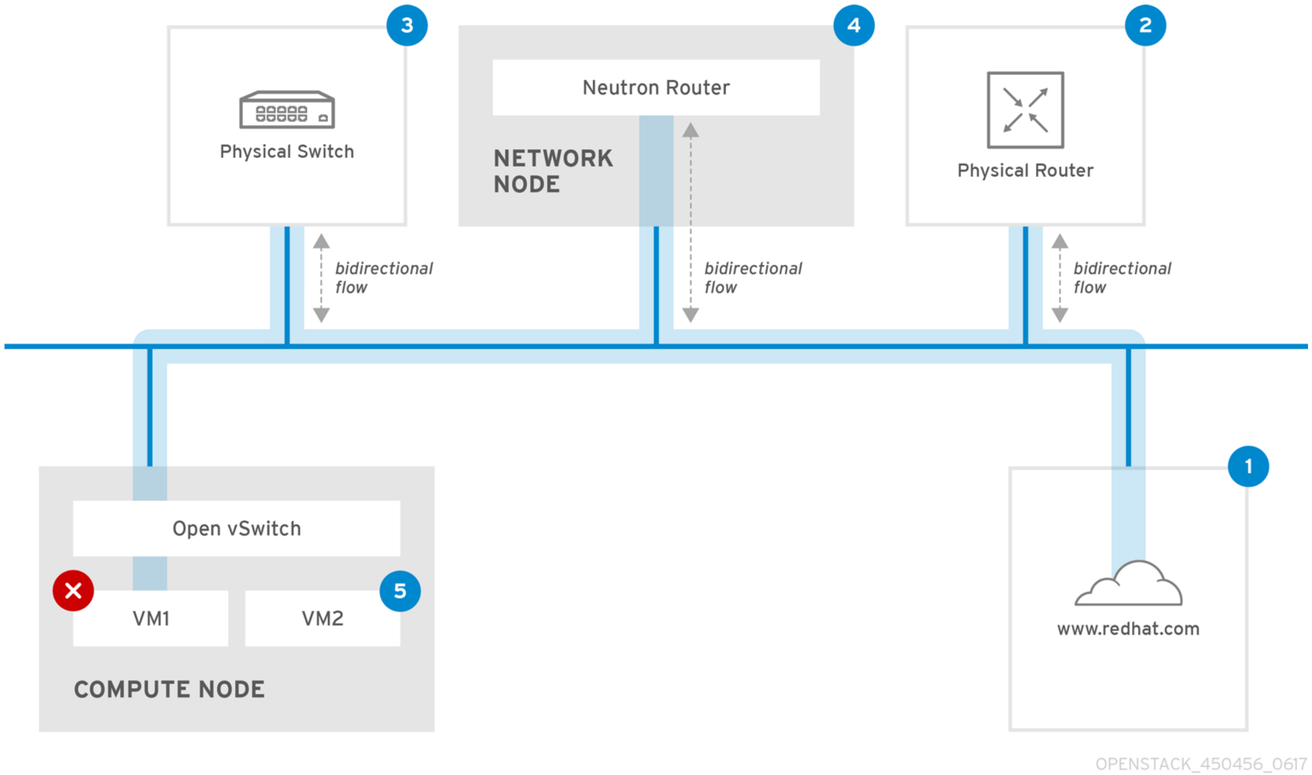
```
$ ping www.example.com

PING e1890.b.akamaiedge.net (125.56.247.214) 56(84) bytes of data.
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=1
ttl=54 time=13.4 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=2
ttl=54 time=13.5 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=3
ttl=54 time=13.4 ms
^C
```

You can terminate the ping command with Ctrl-c, after which a summary of the results is presented. Zero percent packet loss indicates that the connection was stable and did not time out.

```
--- e1890.b.akamaiedge.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.461/13.498/13.541/0.100 ms
```

The results of a ping test can be very revealing, depending on which destination you test. For example, in the following diagram VM1 is experiencing some form of connectivity issue. The possible destinations are numbered in blue, and the conclusions drawn from a successful or failed result are presented:



1. **The internet** - a common first step is to send a ping test to an internet location, such as www.example.com.

   - *Success*: This test indicates that all the various network points in between the machine and the Internet are functioning correctly. This includes the virtual and physical network infrastructure.

   - *Failure*: There are various ways in which a ping test to a distant internet location can fail. If other machines on your network are able to successfully ping the internet, that proves the internet connection is working, and the issue is likely within the configuration of the local machine.

2. **Physical router** - This is the router interface that the network administrator designates to direct traffic onward to external destinations.

   - *Success*: Ping tests to the physical router can determine whether the local network and underlying switches are functioning. These packets do not traverse the router, so they do not prove whether there is a routing issue present on the default gateway.

   - *Failure*: This indicates that the problem lies between VM1 and the default gateway. The router/switches might be down, or you may be using an incorrect default gateway. Compare the configuration with that on another server that you know is functioning correctly. Try pinging another server on the local network.

3. **Neutron router** - This is the virtual SDN (Software-defined Networking) router that Red Hat OpenStack Platform uses to direct the traffic of virtual machines.

   - *Success*: Firewall is allowing ICMP traffic, the Networking node is online.

   - *Failure*: Confirm whether ICMP traffic is permitted in the security group of the instance. Check that the Networking node is online, confirm that all the required services are running, and review the L3 agent log (*/var/log/neutron/l3-agent.log*).

4. **Physical switch** - The physical switch manages traffic between nodes on the same physical network.
   - *Success*: Traffic sent by a VM to the physical switch must pass through the virtual network infrastructure, indicating that this segment is functioning correctly.
   - *Failure*: Check that the physical switch port is configured to trunk the required VLANs.
5. **VM2** - Attempt to ping a VM on the same subnet, on the same Compute node.
   - *Success*: The NIC driver and basic IP configuration on VM1 are functional.
   - *Failure*: Validate the network configuration on VM1. Or, firewall on VM2 might simply be blocking ping traffic. In addition, verify the virtual switching configuration and review the Open vSwitch log files.

6.2. VIEWING CURRENT PORT STATUS

A basic troubleshooting task is to create an inventory of all of the ports attached to a router and determine the port status (**DOWN** or **ACTIVE**).

Procedure

1. To view all the ports that attach to the router named **r1**, run the following command:

```
# openstack port list --router r1
```

Sample output

```
+--------------------------------------+------+-------------------+-------------------------------------------
-----------------------------------------+
| id                                   | name | mac_address       | fixed_ips
|
+--------------------------------------+------+-------------------+-------------------------------------------
-----------------------------------------+
| b58d26f0-cc03-43c1-ab23-ccdb1018252a |      | fa:16:3e:94:a7:df | {"subnet_id": "a592fdba-
babd-48e0-96e8-2dd9117614d3", "ip_address": "192.168.200.1"} |
| c45e998d-98a1-4b23-bb41-5d24797a12a4 |      | fa:16:3e:ee:6a:f7 | {"subnet_id": "43f8f625-
c773-4f18-a691-fd4ebfb3be54", "ip_address": "172.24.4.225"}  |
+--------------------------------------+------+-------------------+-------------------------------------------
-----------------------------------------+
```

2. To view the details of each port, run the following command. Include the port ID of the port that you want to view. The result includes the port status, indicated in the following example as having an **ACTIVE** state:

```
# openstack port show b58d26f0-cc03-43c1-ab23-ccdb1018252a
```

Sample output

```
+-----------------------+---------------------------------------------------------------------------------------
+
| Field                 | Value                                                                                 |
+-----------------------+---------------------------------------------------------------------------------------
```

```
+
| admin_state_up        | True                                                                        |
| allowed_address_pairs |                                                                             |
| binding:host_id       | node.example.com                                                      |
| binding:profile       | {}                                                                    |
| binding:vif_details   | {"port_filter": true, "ovs_hybrid_plug": true}                              |
| binding:vif_type      | ovs                                                                   |
| binding:vnic_type     | normal                                                                |
| device_id             | 49c6ebdc-0e62-49ad-a9ca-58cea464472f                                        |
| device_owner          | network:router_interface                                              |
| extra_dhcp_opts       |                                                                       |
| fixed_ips             | {"subnet_id": "a592fdba-babd-48e0-96e8-2dd9117614d3", "ip_address":
"192.168.200.1"} |
| id                    | b58d26f0-cc03-43c1-ab23-ccdb1018252a                                  |
| mac_address           | fa:16:3e:94:a7:df                                                     |
| name                  |                                                                       |
| network_id            | 63c24160-47ac-4140-903d-8f9a670b0ca4
|
| security_groups       |                                                                       |
| status                | ACTIVE                                                                |
| tenant_id             | d588d1112e0f496fb6cac22f9be45d49                                      |
+-----------------------+----------------------------------------------------------------------------------------
+
```

3. Perform step 2 for each port to determine its status.

6.3. TROUBLESHOOTING CONNECTIVITY TO VLAN PROVIDER NETWORKS

OpenStack Networking can trunk VLAN networks through to the SDN switches. Support for VLAN-tagged provider networks means that virtual instances can integrate with server subnets in the physical network.

Procedure

1. Ping the gateway with **ping <gateway-IP-address>**.
   Consider this example, in which a network is created with these commands:

   ```
   # openstack network create --provider-network-type vlan --provider-physical-network phy-
   eno1 --provider-segment 120 provider
   # openstack subnet create --no-dhcp --allocation-pool
   start=192.168.120.1,end=192.168.120.153 --gateway 192.168.120.254 --network  provider
   public_subnet
   ```

   In this example, the gateway IP address is **192.168.120.254**.

   ```
   $ ping 192.168.120.254
   ```

2. If the ping fails, do the following:

   a. Confirm that you have network flow for the associated VLAN.
      It is possible that the VLAN ID has not been set. In this example, OpenStack Networking is configured to trunk VLAN 120 to the provider network. (See **--provider:segmentation_id=120** in the example in step 1.)

b. Confirm the VLAN flow on the bridge interface using the command, **ovs-ofctl dump-flows <bridge-name>**.
   In this example the bridge is named **br-ex**:

```
# ovs-ofctl dump-flows br-ex

 NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=987.521s, table=0, n_packets=67897, n_bytes=14065247,
idle_age=0, priority=1 actions=NORMAL
  cookie=0x0, duration=986.979s, table=0, n_packets=8, n_bytes=648, idle_age=977,
priority=2,in_port=12 actions=drop
```

6.4. REVIEWING THE VLAN CONFIGURATION AND LOG FILES

To help validate or troubleshoot a deployment, you can:

- verify the registration and status of Red Hat Openstack Platform (RHOSP) Networking service (neutron) agents.
- validate network configuration values such as VLAN ranges.

Procedure

1. Use the **openstack network agent list** command to verify that the RHOSP Networking service agents are up and registered with the correct host names.

```
(overcloud)[stack@undercloud~]$ openstack network agent list
+--------------------------------------+--------------------+----------------------+-------+----------------+
| id                                   | agent_type         | host                 | alive | admin_state_up |
+--------------------------------------+--------------------+----------------------+-------+----------------+
| a08397a8-6600-437d-9013-b2c5b3730c0c | Metadata agent     | rhelosp.example.com  | :-)
| True           |
| a5153cd2-5881-4fc8-b0ad-be0c97734e6a | L3 agent           | rhelosp.example.com  | :-)   |
True           |
| b54f0be7-c555-43da-ad19-5593a075ddf0 | DHCP agent         | rhelosp.example.com  | :-)
| True           |
| d2be3cb0-4010-4458-b459-c5eb0d4d354b | Open vSwitch agent | rhelosp.example.com  |
:-)   | True           |
+--------------------------------------+--------------------+----------------------+-------+----------------+
```

2. Review **/var/log/containers/neutron/openvswitch-agent.log**. Look for confirmation that the creation process used the **ovs-ofctl** command to configure VLAN trunking.

3. Validate **external_network_bridge** in the **/etc/neutron/l3_agent.ini** file. If there is a hardcoded value in the **external_network_bridge** parameter, you cannot use a provider network with the L3-agent, and you cannot create the necessary flows. The **external_network_bridge** value must be in the format `external_network_bridge = "" `.

4. Check the **network_vlan_ranges** value in the **/etc/neutron/plugin.ini** file. For provider networks, do not specify the numeric VLAN ID. Specify IDs only when using VLAN isolated project networks.

5. Validate the **OVS agent configuration file bridge mappings**, to confirm that the bridge mapped to **phy-eno1** exists and is properly connected to **eno1**.

6.5. PERFORMING BASIC ICMP TESTING WITHIN THE ML2/OVN NAMESPACE

As a basic troubleshooting step, you can attempt to ping an instance from an OVN metadata interface that is on the same layer 2 network.

Prerequisites

- RHOSP deployment, with ML2/OVN as the Networking service (neutron) default mechanism driver.

Procedure

1. Log in to the overcloud using your Red Hat OpenStack Platform credentials.

2. Run the **openstack server list** command to obtain the name of a VM instance.

3. Run the **openstack server show** command to determine the Compute node on which the instance is running.

   Example

   ```
   $ openstack server show my_instance -c OS-EXT-SRV-ATTR:host \
   -c addresses
   ```

   Sample output

   ```
   +----------------------+-------------------------------------------------+
   | Field                | Value                                           |
   +----------------------+-------------------------------------------------+
   | OS-EXT-SRV-ATTR:host | compute0.ctlplane.example.com                   |
   | addresses            | finance-network1=192.0.2.2; provider-           |
   |                      | storage=198.51.100.13                           |
   +----------------------+-------------------------------------------------+
   ```

4. Log in to the Compute node host.

   Example

   ```
   $ ssh tripleo-admin@compute0.ctlplane
   ```

5. Run the **ip netns list** command to see the OVN metadata namespaces.

   Sample output

   ```
   ovnmeta-07384836-6ab1-4539-b23a-c581cf072011 (id: 1)
   ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa (id: 0)
   ```

6. Using the metadata namespace run an **ip netns exec** command to ping the associated network.

   Example

```
$ sudo ip netns exec ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa \
ping 192.0.2.2
```

Sample output

```
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.470 ms
64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.183 ms
64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.296 ms
64 bytes from 192.0.2.2: icmp_seq=5 ttl=64 time=0.307 ms
^C
--- 192.0.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 122ms
rtt min/avg/max/mdev = 0.183/0.347/0.483/0.116 ms
```

Additional resources

- server show in the *Command line interface reference*

6.6. TROUBLESHOOTING FROM WITHIN PROJECT NETWORKS (ML2/OVS)

In Red Hat Openstack Platform (RHOSP) ML2/OVS networks, all project traffic is contained within network namespaces so that projects can configure networks without interfering with each other. For example, network namespaces allow different projects to have the same subnet range of 192.168.1.1/24 without interference between them.

Prerequisites

- RHOSP deployment, with ML2/OVS as the Networking service (neutron) default mechanism driver.

Procedure

1. Determine which network namespace contains the network, by listing all of the project networks using the **openstack network list** command:

```
$ openstack network list
```

In this output, note that the ID for the **web-servers** network (**9cb32fe0-d7fb-432c-b116-f483c6497b08**). The command appends the network ID to the network namespace, which enables you to identify the namespace in the next step.

Sample output

```
+--------------------------------------+-------------+-------------------------------------------------------+
| id                                   | name        | subnets                                               |
+--------------------------------------+-------------+-------------------------------------------------------+
| 9cb32fe0-d7fb-432c-b116-f483c6497b08 | web-servers | 453d6769-fcde-4796-a205-
66ee01680bba 192.168.212.0/24 |
| a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81 | private     | c1e58160-707f-44a7-bf94-
8694f29e74d3 10.0.0.0/24      |
| baadd774-87e9-4e97-a055-326bb422b29b | private     | 340c58e1-7fe7-4cf2-96a7-
96a0a4ff3231 192.168.200.0/24 |
| 24ba3a36-5645-4f46-be47-f6af2a7d8af2 | public      | 35f3d2cb-6e4b-4527-a932-
952a395c4bb3 172.24.4.224/28  |
+--------------------------------------+-------------+------------------------------------------------------+
```

2. List all the network namespaces using the **ip netns list** command:

   ```
   # ip netns list
   ```

   The output contains a namespace that matches the **web-servers** network ID.

   In this output, the namespace is **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08**.

   Sample output

   ```
   qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08
   qrouter-31680a1c-9b3e-4906-bd69-cb39ed5faa01
   qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b
   qdhcp-a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81
   qrouter-e9281608-52a6-4576-86a6-92955df46f56
   ```

3. Examine the configuration of the **web-servers** network by running commands within the namespace, prefixing the troubleshooting commands with **ip netns exec <namespace>**.
   In this example, the **route -n** command is used.

   Example

   ```
   # ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b route -n
   ```

   Sample output

   ```
   Kernel IP routing table
   Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
   0.0.0.0         172.24.4.225    0.0.0.0         UG    0      0        0 qg-8d128f89-87
   172.24.4.224    0.0.0.0         255.255.255.240 U     0      0        0 qg-8d128f89-87
   192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 qr-8efd6357-96
   ```

6.7. PERFORMING ADVANCED ICMP TESTING WITHIN THE NAMESPACE (ML2/OVS)

You can troubleshoot Red Hat Openstack Platform (RHOSP) ML2/OVS networks, using a combination of **tcpdump** and **ping** commands.

Prerequisites

- RHOSP deployment, with ML2/OVS as the Networking service (neutron) default mechanism driver.

Procedure

1. Capture ICMP traffic using the **tcpdump** command:

Example

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b tcpdump -qnntpi any icmp
```

2. In a separate command line window, perform a ping test to an external network:

Example

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b ping www.example.com
```

3. In the terminal running the **tcpdump** session, observe detailed results of the ping test.

Sample output

```
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
IP (tos 0xc0, ttl 64, id 55447, offset 0, flags [none], proto ICMP (1), length 88)
    172.24.4.228 > 172.24.4.228: ICMP host 192.168.200.20 unreachable, length 68
 IP (tos 0x0, ttl 64, id 22976, offset 0, flags [DF], proto UDP (17), length 60)
    172.24.4.228.40278 > 192.168.200.21: [bad udp cksum 0xfa7b -> 0xe235!] UDP, length 32
```

NOTE

When you perform a **tcpdump** analysis of traffic, you see the responding packets heading to the router interface rather than to the VM instance. This is expected behavior, as the **qrouter** performs Destination Network Address Translation (DNAT) on the return packets.

6.8. CREATING ALIASES FOR OVN TROUBLESHOOTING COMMANDS

You run OVN commands, such as **ovn-nbctl show**, in the **ovn_controller** container. The container runs on the Controller node and Compute nodes. To simplify your access to the commands, create and source a script that defines aliases.

Prerequisites

- Red Hat OpenStack Platform deployment with ML2/OVN as the default mechanism driver.

Procedure

1. Log in to the Controller host as a user that has the necessary privileges to access the OVN containers.

Example

```
$ ssh tripleo-admin@controller-0.ctlplane
```

2. Create a shell script file that contains the **ovn** commands that you want to run.

Example

```
vi ~/bin/ovn-alias.sh
```

3. Add the **ovn** commands, and save the script file.

   Example

   In this example, the **ovn-sbctl**, **ovn-nbctl**, and **ovn-trace** commands have been added to an alias file:

   ```
   REMOTE_IP=$(sudo ovs-vsctl get open . external_ids:ovn-remote)
   NBDB=$(echo $REMOTE_IP | sed 's/6642/6641/g')
   SBDB=$REMOTE_IP
   alias ovn-sbctl="sudo podman exec ovn_controller ovn-sbctl --db=$SBDB"
   alias ovn-nbctl="sudo podman exec ovn_controller ovn-nbctl --db=$NBDB"
   alias ovn-trace="sudo podman exec ovn_controller ovn-trace --db=$SBDB"
   ```

4. Repeat the steps in this procedure on the Compute host.

Validation

1. Source the script file.

   Example

   ```
   # source ovn-alias.sh
   ```

2. Run a command to confirm that your script file works properly.

   Example

   ```
   # ovn-nbctl show
   ```

   Sample output

   ```
   switch 26ce22db-1795-41bd-b561-9827cbd81778 (neutron-f8e79863-6c58-43d0-8f7d-
   8ec4a423e13b) (aka internal_network)
    port 1913c3ae-8475-4b60-a479-df7bcce8d9c8
        addresses: ["fa:16:3e:33:c1:fc 192.168.254.76"]
    port 1aabaee3-b944-4da2-bf0a-573215d3f3d9
        addresses: ["fa:16:3e:16:cb:ce 192.168.254.74"]
    port 7e000980-59f9-4a0f-b76a-4fdf4e86f27b
        type: localport
        addresses: ["fa:16:3e:c9:30:ed 192.168.254.2"]
   ```

Additional resources

- **ovn-nbctl --help** command
- **ovn-sbctl --help** command
- **ovn-trace --help** command

6.9. MONITORING OVN LOGICAL FLOWS

OVN uses logical flows that are tables of flows with a priority, match, and actions. These logical flows are distributed to the **ovn-controller** running on each Red Hat Openstack Platform (RHOSP) Compute node. Use the **ovn-sbctl lflow-list** command on the Controller node to view the full set of logical flows.

Prerequisites

- RHOSP deployment with ML2/OVN as the Networking service (neutron) default mechanism driver.
- Create an alias file for the OVN database commands.
  See, Section 6.8, “Creating aliases for OVN troubleshooting commands” .

Procedure

1. Log in to the Controller host as a user that has the necessary privileges to access the OVN containers.

   Example

   ```
   $ ssh tripleo-admin@controller-0.ctlplane
   ```

2. Source the alias file for the OVN database commands.
   For more information, see Section 6.8, “Creating aliases for OVN troubleshooting commands” .

   Example

   ```
   source ~/ovn-alias.sh
   ```

3. View the logical flows:

   ```
   $ ovn-sbctl lflow-list
   ```

4. Inspect the output.

   Sample output

   ```
   Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: ingress
     table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
     table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
     table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port1" && eth.src ==
   {00:00:00:00:00:01}), action=(next;)
     table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port2" && eth.src ==
   {00:00:00:00:00:02}), action=(next;)
     table=1 (ls_in_port_sec_ip  ), priority=0    , match=(1), action=(next;)
     table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
   00:00:00:00:00:01 && arp.sha == 00:00:00:00:00:01), action=(next;)
     table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
   00:00:00:00:00:01 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
   00:00:00:00:00:01) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:01)))), action=
   (next;)
     table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
   00:00:00:00:00:02 && arp.sha == 00:00:00:00:00:02), action=(next;)
     table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
   00:00:00:00:00:02 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
   ```

```
00:00:00:00:00:02) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:02)))), action=
(next;)
  table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port1" && (arp || nd)),
action=(drop;)
  table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port2" && (arp || nd)),
action=(drop;)
  table=2 (ls_in_port_sec_nd  ), priority=0    , match=(1), action=(next;)
  table=3 (ls_in_pre_acl      ), priority=0, match=(1), action=(next;)
  table=4 (ls_in_pre_lb       ), priority=0    , match=(1), action=(next;)
  table=5 (ls_in_pre_stateful ), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
  table=5 (ls_in_pre_stateful ), priority=0    , match=(1), action=(next;)
  table=6 (ls_in_acl          ), priority=0    , match=(1), action=(next;)
  table=7 (ls_in_qos_mark     ), priority=0    , match=(1), action=(next;)
  table=8 (ls_in_lb           ), priority=0    , match=(1), action=(next;)
  table=9 (ls_in_stateful     ), priority=100  , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
  table=9 (ls_in_stateful     ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
  table=9 (ls_in_stateful     ), priority=0    , match=(1), action=(next;)
  table=10(ls_in_arp_rsp      ), priority=0    , match=(1), action=(next;)
  table=11(ls_in_dhcp_options ), priority=0    , match=(1), action=(next;)
  table=12(ls_in_dhcp_response), priority=0    , match=(1), action=(next;)
  table=13(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast), action=(outport =
"_MC_flood"; output;)
  table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:01), action=
(outport = "sw0-port1"; output;)
  table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:02), action=
(outport = "sw0-port2"; output;)
Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: egress
  table=0 (ls_out_pre_lb      ), priority=0    , match=(1), action=(next;)
  table=1 (ls_out_pre_acl     ), priority=0    , match=(1), action=(next;)
  table=2 (ls_out_pre_stateful), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
  table=2 (ls_out_pre_stateful), priority=0    , match=(1), action=(next;)
  table=3 (ls_out_lb          ), priority=0    , match=(1), action=(next;)
  table=4 (ls_out_acl         ), priority=0    , match=(1), action=(next;)
  table=5 (ls_out_qos_mark    ), priority=0    , match=(1), action=(next;)
  table=6 (ls_out_stateful    ), priority=100  , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
  table=6 (ls_out_stateful    ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
  table=6 (ls_out_stateful    ), priority=0    , match=(1), action=(next;)
  table=7 (ls_out_port_sec_ip ), priority=0    , match=(1), action=(next;)
  table=8 (ls_out_port_sec_l2 ), priority=100  , match=(eth.mcast), action=(output;)
  table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
  table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port2" && eth.dst ==
{00:00:00:00:00:02}), action=(output;)
```

Key differences between OVN and OpenFlow include:

- OVN ports are logical entities that reside somewhere on a network, not physical ports on a single switch.
- OVN gives each table in the pipeline a name in addition to its number. The name describes the purpose of that stage in the pipeline.
- The OVN match syntax supports complex Boolean expressions.

- The actions supported in OVN logical flows extend beyond those of OpenFlow. You can implement higher level features, such as DHCP, in the OVN logical flow syntax.

5. Run an OVN trace.
   The **ovn-trace** command can simulate how a packet travels through the OVN logical flows, or help you determine why a packet is dropped. Provide the **ovn-trace** command with the following parameters:

   DATAPATH

   The logical switch or logical router where the simulated packet starts.

   MICROFLOW

   The simulated packet, in the syntax used by the **ovn-sb** database.

Example

This example displays the **--minimal** output option on a simulated packet and shows that the packet reaches its destination:

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

Sample output

```
#
reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x
0000
    output("sw0-port2");
```

Example

In more detail, the **--summary** output for this same simulated packet shows the full execution pipeline:

```
$ ovn-trace --summary sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

Sample output

The sample output shows:

- The packet enters the **sw0** network from the **sw0-port1** port and runs the ingress pipeline.

- The **outport** variable is set to **sw0-port2** indicating that the intended destination for this packet is **sw0-port2**.

- The packet is output from the ingress pipeline, which brings it to the egress pipeline for **sw0** with the **outport** variable set to **sw0-port2**.

- The output action is executed in the egress pipeline, which outputs the packet to the current value of the **outport** variable, which is **sw0-port2**.

  ```
  #
  reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type
  =0x0000
  ```

```
ingress(dp="sw0", inport="sw0-port1") {
    outport = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", outport="sw0-port2") {
        output;
        /* output to "sw0-port2", type "" */;
    };
};
```

Additional resources

- Section 6.8, "Creating aliases for OVN troubleshooting commands"
- **ovn-sbctl --help** command
- **ovn-trace --help** command

6.10. MONITORING OPENFLOWS

You can use **ovs-ofctl dump-flows** command to monitor the OpenFlow flows on a logical switch in your Red Hat Openstack Platform (RHOSP) network.

Prerequisites

- RHOSP deployment with ML2/OVN as the Networking service (neutron) default mechanism driver.

Procedure

1. Log in to the Controller host as a user that has the necessary privileges to access the OVN containers.

   #### Example

   ```
   $ ssh tripleo-admin@controller-0.ctlplane
   ```

2. Run the **ovs-ofctl dump-flows** command.

   #### Example

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

3. Inspect the output, which resembles the following output.

   #### Sample output

   ```
   $ ovs-ofctl dump-flows br-int
   NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=72.132s, table=0, n_packets=0, n_bytes=0, idle_age=72,
   priority=10,in_port=1,dl_src=00:00:00:00:00:01 actions=resubmit(,1)
    cookie=0x0, duration=60.565s, table=0, n_packets=0, n_bytes=0, idle_age=60,
   priority=10,in_port=2,dl_src=00:00:00:00:00:02 actions=resubmit(,1)
   ```

```
 cookie=0x0, duration=28.127s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0
actions=drop
 cookie=0x0, duration=13.887s, table=1, n_packets=0, n_bytes=0, idle_age=13,
priority=0,in_port=1 actions=output:2
 cookie=0x0, duration=4.023s, table=1, n_packets=0, n_bytes=0, idle_age=4,
priority=0,in_port=2 actions=output:1
```

Additional resources

- **ovs-ofctl --help** command

6.11. MONITORING OVN DATABASE STATUS

You can use the **ovs-appctl** command to monitor connections between OVN database servers.

Prerequisites

- RHOSP deployment with ML2/OVN as the Networking service (neutron) default mechanism driver.

Procedure

1. Log in to a Controller host as a user that has the necessary privileges to access the OVN containers.
   Monitoring from a server on a single Controller host provides the information you need to to verify basic cluster health and to diagnose many types of problems. For a very thorough analysis, perform this procedure on all Controllers.

   Example

   ```
   $ ssh tripleo-admin@compute-0
   ```

2. Run the **ovs-appctl** command.

   Example: northbound database

   ```
   $ ovs-appctl -t /var/lib/openvswitch/ovn/ovnnb_db.ctl cluster/status OVN_Northbound
   ```

   Example: southbound database

   ```
   ovs-appctl -t /var/lib/openvswitch/ovn/ovnsb_db.ctl cluster/status OVN_Southbound
   ```

3. Inspect the output, which resembles the following output.

   Sample output: southbound database

   This sample output was generated on server 1114, which was a follower at the time.

   ```
   1114
   Name: OVN_Southbound
   Cluster ID: 017a (017add73-58f1-4fcd-ae35-bacc0f07ce57)
   Server ID: 1114 (1114865d-4f42-443a-b758-d4431fc35748)
   Address: tcp:[fd00:fd00:fd00:2000::4a]:6644
   ```

```
Status: cluster member
Role: follower
Term: 90
Leader: ca6e
Vote: ca6e

Last Election started 27881511 ms ago, reason: leadership_transfer
Last Election won: 27881503 ms ago
Election timer: 16000
Log: [51470, 51737]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->ca6e ->0f90 <-ca6e <-0f90
Disconnections: 0
Servers:
    1114 (1114 at tcp:[fd00:fd00:fd00:2000::4a]:6644) (self)
    ca6e (ca6e at tcp:[fd00:fd00:fd00:2000::18f]:6644) last msg 5141 ms ago
    0f90 (0f90 at tcp:[fd00:fd00:fd00:2000::2e0]:6644) last msg 22106129 ms ago
```

Diagnostic indications from sample output

A right-pointing arrow (→) represents outbound connection from this server to another A left-pointing arrow (←) represents inbound connection from another server to this server.

All servers are active and connected

Connections: ->ca6e ->0f90 <-ca6e <-0f90
This three-node cluster appears healthy. The server 1114 has inbound and outbound connections with the other two servers, ca6e and 0f90.

A server is disconnected from the cluster

Connections: ->ca6e (->0f90) <-ca6e
The incoming connection from server 0f90 is not listed. The parenthesis around the outgoing connection indicate that outbound messages to 0f90 failed. For most situations, connecting to any server in the cluster provides enough information to determine whether there are issues with the cluster. Running the diagnostics on all servers provides more detailed information and might detect problems that you cannot detect from a single server.

The cluster has lost quorum

```
Role: candidate
...
Leader: unknown
```

This server is a candidate and the leader is unknown.

The ovsdb-server is down on this node

```
2024-03-27T22:10:28Z|00001|unixctl|WARN|failed to connect to
/var/lib/openvswitch/ovn/ovnsb_db.ctl
ovs-appctl: cannot connect to "/var/lib/openvswitch/ovn/ovnsb_db.ctl" (Connection
refused)

<exits with non-zero status>
```

In this case, you cannot get all the information you need from a single server. For example, you cannot determine whether the other servers are running. If the server is down, run ovs-appctl on another server.

Time since last message to leader from each follower (only updated on leader)

```
Servers:
   1114 (1114 at tcp:[fd00:fd00:fd00:2000::4a]:6644) next_index=51737
match_index=51736 last msg 224 ms ago
   ca6e (ca6e at tcp:[fd00:fd00:fd00:2000::18f]:6644) (self) next_index=51470
match_index=51736
   0f90 (0f90 at tcp:[fd00:fd00:fd00:2000::2e0]:6644) next_index=51737
match_index=51736 last msg 224 ms ago
```

Log on to the cluster leader host and run ovs-appctl. Note that a new leader can be elected at any time.

Additional resources

- **ovs-appctl --help** command

6.12. VALIDATING YOUR ML2/OVN DEPLOYMENT

Validating the ML2/OVN networks on your Red Hat OpenStack Platform (RHOSP) deployment consists of creating a test network and subnet and performing diagnostic tasks such as verifying that specfic containers are running.

Prerequisites

- New deployment of RHOSP, with ML2/OVN as the Networking service (neutron) default mechanism driver.

- Create an alias file for the OVN database commands.
  See, Section 6.8, “Creating aliases for OVN troubleshooting commands” .

Procedure

1. Create a test network and subnet.

   ```
   NETWORK_ID=\
   $(openstack network create internal_network | awk '/\| id/ {print $4}')

   openstack subnet create internal_subnet \
   --network $NETWORK_ID \
   --dns-nameserver 8.8.8.8 \
   --subnet-range 192.168.254.0/24
   ```

   If you encounter errors, perform the steps that follow.

2. Verify that the relevant containers are running on the Controller host:

   a. Log in to the Controller host as a user that has the necessary privileges to access the OVN containers.

Example

```
$ ssh tripleo-admin@controller-0.ctlplane
```

b. Enter the following command:

```
$ sudo podman ps -a --format="{{.Names}}"|grep ovn
```

As shown in the following sample, the output should list the OVN containers:

Sample output

```
container-puppet-ovn_controller
ovn_cluster_north_db_server
ovn_cluster_south_db_server
ovn_cluster_northd
ovn_controller
```

3. Verify that the relevant containers are running on the Compute host:

   a. Log in to the Compute host as a user that has the necessary privileges to access the OVN containers.

   ### Example

   ```
   $ ssh tripleo-admin@compute-0.ctlplane
   ```

   b. Enter the following command:

   ```
   $ sudo podman ps -a --format="{{.Names}}"|grep ovn
   ```

   As shown in the following sample, the output should list the OVN containers:

   ### Sample output

   ```
   container-puppet-ovn_controller
   ovn_metadata_agent
   ovn_controller
   ```

4. Inspect log files for error messages.

   ```
   grep -r ERR /var/log/containers/openvswitch/ /var/log/containers/neutron/
   ```

5. Source an alias file to run the OVN database commands.
   For more information, see Section 6.8, "Creating aliases for OVN troubleshooting commands" .

   ### Example

   ```
   $ source ~/ovn-alias.sh
   ```

6. Query the northbound and southbound databases to check for responsiveness.

```
# ovn-nbctl show
# ovn-sbctl show
```

7. Attempt to ping an instance from an OVN metadata interface that is on the same layer 2 network.
   For more information, see Section 6.5, “Performing basic ICMP testing within the ML2/OVN namespace”.

8. If you need to contact Red Hat for support, perform the steps described in this Red Hat Solution, How to collect all required logs for Red Hat Support to investigate an OpenStack issue.

Additional resources

- network create in the *Command line interface reference*
- subnet create in the *Command line interface reference*
- Section 6.8, “Creating aliases for OVN troubleshooting commands”
- **ovn-nbctl --help** command
- **ovn-sbctl --help** command

6.13. SETTING THE LOGGING MODE FOR ML2/OVN

Set ML2/OVN logging to debug mode for additional troubleshooting information. Set logging back to info mode to use less disk space when you do not need additional debugging information.

Prerequisites

- Red Hat OpenStack Platform deployment with ML2/OVN as the default mechanism driver.

Procedure

1. Log in to the Controller or Compute node where you want to set the logging mode as a user that has the necessary privileges to access the OVN containers.

   Example

   ```
   $ ssh tripleo-admin@controller-0.ctlplane
   ```

2. Set the ML2/OVN logging mode.

   Debug logging mode

   ```
   $ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set dbg
   ```

   Info logging mode

   ```
   $ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set info
   ```

Verification

Verification

- Confirm that the **ovn-controller** container log now contains debug messages:

  ```
  $ sudo grep DBG /var/log/containers/openvswitch/ovn-controller.log
  ```

  Sample output

  You should see recent log messages that contain the string **|DBG|**:

  ```
  2022-09-29T20:52:54.638Z|00170|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-
  int.mgmt: received: OFPT_ECHO_REQUEST (OF1.5) (xid=0x0): 0 bytes of payload
  2022-09-29T20:52:54.638Z|00171|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-
  int.mgmt: sent (Success): OFPT_ECHO_REPLY (OF1.5) (xid=0x0): 0 bytes of payload
  ```

- Confirm that the ovn-controller container log contains a string similar to the following:

  ```
  ...received request vlog/set["info"], id=0
  ```

Additional resources

- Section 6.15, “ML2/OVN log files”

6.14. FIXING OVN CONTROLLERS THAT FAIL TO REGISTER ON EDGE SITES

Issue

OVN controllers on Red Hat OpenStack Platform (RHOSP) edge sites fail to register.

> **NOTE**
>
> This error can occur on RHOSP 17.1 ML2/OVN deployments that were updated from an *earlier* RHOSP version—RHOSP 16.1.7 and earlier or RHOSP 16.2.0.

Sample error

The error encountered is similar to the following:

```
2021-04-12T09:14:48.994Z|04754|ovsdb_idl|WARN|transaction error: {"details":"Transaction
causes multiple rows in \"Encap\" table to have identical values (geneve and \"10.14.2.7\") for
index on columns \"type\" and \"ip\".  First row, with UUID 3973cad5-eb8a-4f29-85c3-
c105d861c0e0, was inserted by this transaction.  Second row, with UUID f06b71a8-4162-475b-
8542-d27db3a9097a, existed in the database before this transaction and was not modified by the
transaction.","error":"constraint violation"}
```

Cause

If the **ovn-controller** process replaces the hostname, it registers another chassis entry which includes another encap entry. For more information, see BZ#1948472.

Resolution

Follow these steps to resolve the problem:

1. If you have not already, create aliases for the necessary OVN database commands that you will use later in this procedure.
   For more information, see Creating aliases for OVN troubleshooting commands .

2. Log in to the Controller host as a user that has the necessary privileges to access the OVN containers.

   Example

   ```
   $ ssh tripleo-admin@controller-0.ctlplane
   ```

3. Obtain the IP address from the **/var/log/containers/openvswitch/ovn-controller.log**

4. Confirm that the IP address is correct:

   ```
   ovn-sbctl list encap |grep -a3 <IP address from ovn-controller.log>
   ```

5. Delete the chassis that contains IP address:

   ```
   ovn-sbctl chassis-del <chassis-id>
   ```

6. Check the **Chassis_Private** table to confirm that chassis has been removed:

   ```
   ovn-sbctl find Chassis_private chassis="[]"
   ```

7. If any entries are reported, remove them with the following command:

   ```
   $ ovn-sbctl destroy Chassis_Private <listed_id>
   ```

8. Restart the following containers:

   - **tripleo_ovn_controller**
   - **tripleo_ovn_metadata_agent**

     ```
     $ sudo systemctl restart tripleo_ovn_controller
     $ sudo systemctl restart tripleo_ovn_metadata_agent
     ```

Verification

- Confirm that OVN agents are running:

  ```
  $ openstack network agent list -c "Agent Type" -c State -c Binary
  ```

  Sample output

  ```
  +------------------------------+-------+----------------------------+
  | Agent Type                   | State | Binary                     |
  +------------------------------+-------+----------------------------+
  | OVN Controller Gateway agent | UP    | ovn-controller             |
  | OVN Controller Gateway agent | UP    | ovn-controller             |
  | OVN Controller agent         | UP    | ovn-controller             |
  ```

```
| OVN Metadata agent          | UP    | neutron-ovn-metadata-agent |
| OVN Controller Gateway agent | UP    | ovn-controller             |
+------------------------------+-------+----------------------------+
```

6.15. ML2/OVN LOG FILES

Log files track events related to the deployment and operation of the ML2/OVN mechanism driver.

Table 6.1. ML2/OVN log files per node

Nodes	Log	Path /var/log/containers/openvswitch...
Controller, Compute, Networking	OVS northbound database server	**.../ovn-controller.log**
Controller	OVS northbound database server	**.../ovsdb-server-nb.log**
Controller	OVS southbound database server	**.../ovsdb-server-sb.log**
Controller	OVN northbound database server	**.../ovn-northd.log**

CHAPTER 7. CONFIGURING PHYSICAL SWITCHES FOR OPENSTACK NETWORKING

This chapter documents the common physical switch configuration steps required for OpenStack Networking. Vendor-specific configuration is included for certain switches.

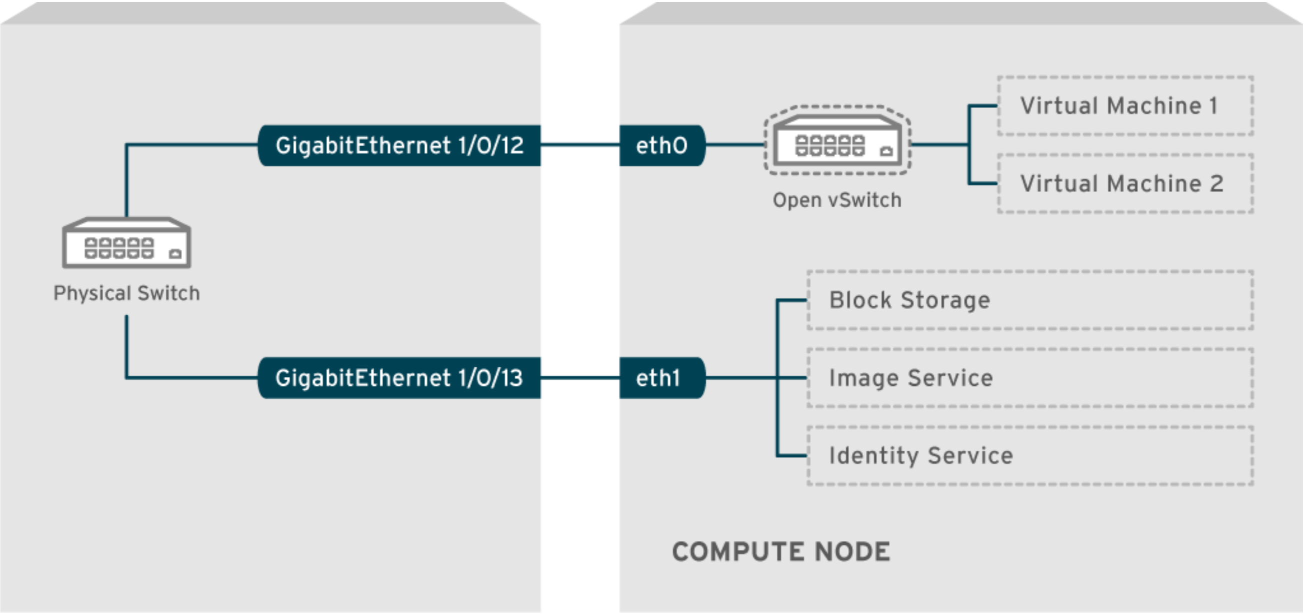
7.1. PLANNING YOUR PHYSICAL NETWORK ENVIRONMENT

The physical network adapters in your OpenStack nodes carry different types of network traffic, such as instance traffic, storage data, or authentication requests. The type of traffic these NICs carry affects how you must configure the ports on the physical switch.

First, you must decide which physical NICs oFn your Compute node you want to carry which types of traffic. Then, when the NIC is cabled to a physical switch port, you must configure the switch port to allow trunked or general traffic.

For example, the following diagram depicts a Compute node with two NICs, eth0 and eth1. Each NIC is cabled to a Gigabit Ethernet port on a physical switch, with eth0 carrying instance traffic, and eth1 providing connectivity for OpenStack services:

Figure 7.1. Sample network layout



OPENSTACK_377160_1115

NOTE

This diagram does not include any additional redundant NICs required for fault tolerance.

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

7.2. CONFIGURING A CISCO CATALYST SWITCH

7.2.1. About trunk ports

With OpenStack Networking you can connect instances to the VLANs that already exist on your physical network. The term *trunk* is used to describe a port that allows multiple VLANs to traverse through the same port. Using these ports, VLANs can span across multiple switches, including virtual switches. For example, traffic tagged as VLAN110 in the physical network reaches the Compute node, where the 8021q module directs the tagged traffic to the appropriate VLAN on the vSwitch.

7.2.2. Configuring trunk ports for a Cisco Catalyst switch

- If using a Cisco Catalyst switch running Cisco IOS, you might use the following configuration syntax to allow traffic for VLANs 110 and 111 to pass through to your instances.
  This configuration assumes that your physical node has an ethernet cable connected to interface GigabitEthernet1/0/12 on the physical switch.

  IMPORTANT

  These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

  ```
  interface GigabitEthernet1/0/12
    description Trunk to Compute Node
    spanning-tree portfast trunk
    switchport trunk encapsulation dot1q
    switchport mode trunk
    switchport trunk native vlan 2
    switchport trunk allowed vlan 2,110,111
  ```

  Use the following list to understand these parameters:

Field	Description
interface GigabitEthernet1/0/12	The switch port that the NIC of the X node connects to. Ensure that you replace the **GigabitEthernet1/0/12** value with the correct port value for your environment. Use the show interface command to view a list of ports.
description Trunk to Compute Node	A unique and descriptive value that you can use to identify this interface.
spanning-tree portfast trunk	If your environment uses STP, set this value to instruct Port Fast that this port is used to trunk traffic.
switchport trunk encapsulation dot1q	Enables the 802.1q trunking standard (rather than ISL). This value varies depending on the configuration that your switch supports.
switchport mode trunk	Configures this port as a trunk port, rather than an access port, meaning that it allows VLAN traffic to pass through to the virtual switches.

Field	Description
switchport trunk native vlan 2	Set a native VLAN to instruct the switch where to send untagged (non-VLAN) traffic.
switchport trunk allowed vlan 2,110,111	Defines which VLANs are allowed through the trunk.

7.2.3. About access ports

Not all NICs on your Compute node carry instance traffic, and so you do not need to configure all NICs to allow multiple VLANs to pass through. Access ports require only one VLAN, and might fulfill other operational requirements, such as transporting management traffic or Block Storage data. These ports are commonly known as access ports and usually require a simpler configuration than trunk ports.

7.2.4. Configuring access ports for a Cisco Catalyst switch

- Using the example from the Figure 7.1, “Sample network layout” diagram, GigabitEthernet1/0/13 (on a Cisco Catalyst switch) is configured as an access port for **eth1**.
  In this configuration,your physical node has an ethernet cable connected to interface GigabitEthernet1/0/12 on the physical switch.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

```
interface GigabitEthernet1/0/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 spanning-tree portfast
```

These settings are described below:

Field	Description
interface GigabitEthernet1/0/13	The switch port that the NIC of the X node connects to. Ensure that you replace the **GigabitEthernet1/0/12** value with the correct port value for your environment. Use the show interface command to view a list of ports.
description Access port for Compute Node	A unique and descriptive value that you can use to identify this interface.
switchport mode access	Configures this port as an access port, rather than a trunk port.

Field	Description
switchport access vlan 200	Configures the port to allow traffic on VLAN 200. You must configure your Compute node with an IP address from this VLAN.
spanning-tree portfast	If using STP, set this value to instruct STP not to attempt to initialize this as a trunk, allowing for quicker port handshakes during initial connections (such as server reboot).

7.2.5. About LACP port aggregation

You can use Link Aggregation Control Protocol (LACP) to bundle multiple physical NICs together to form a single logical channel. Also known as 802.3ad (or bonding mode 4 in Linux), LACP creates a dynamic bond for load-balancing and fault tolerance. You must configure LACP at both physical ends: on the physical NICs, and on the physical switch ports.

Additional resources

- Network Interface Bonding in the *Installing and managing Red Hat OpenStack Platform with director* guide.

7.2.6. Configuring LACP on the physical NIC

You can configure Link Aggregation Control Protocol (LACP) on a physical NIC.

Procedure

1. Edit the */home/stack/network-environment.yaml* file:

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. Configure the Open vSwitch bridge to use LACP:

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

7.2.7. Configuring LACP for a Cisco Catalyst switch

In this example, the Compute node has two NICs using VLAN 100:

Procedure

1. Physically connect both NICs on the Compute node to the switch (for example, ports 12 and 13).

2. Create the LACP port channel:

```
interface port-channel1
  switchport access vlan 100
  switchport mode access
  spanning-tree guard root
```

3. Configure switch ports 12 (Gi1/0/12) and 13 (Gi1/0/13):

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config) interface GigabitEthernet1/0/12
   switchport access vlan 100
   switchport mode access
   speed 1000
   duplex full
   channel-group 10 mode active
   channel-protocol lacp

interface GigabitEthernet1/0/13
  switchport access vlan 100
  switchport mode access
  speed 1000
  duplex full
  channel-group 10 mode active
  channel-protocol lacp
```

4. Review your new port channel. The resulting output lists the new port-channel **Po1**, with member ports **Gi1/0/12** and **Gi1/0/13**:

```
sw01# show etherchannel summary
<snip>

Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
1      Po1(SD)         LACP      Gi1/0/12(D)  Gi1/0/13(D)
```

NOTE

Remember to apply your changes by copying the running-config to the startup-config: **copy running-config startup-config**.

7.2.8. About MTU settings

You must adjust your MTU size for certain types of network traffic. For example, jumbo frames (9000 bytes) are required for certain NFS or iSCSI traffic.

NOTE

You must change MTU settings from end-to-end on all hops that the traffic is expected to pass through, including any virtual switches.

Additional resources

- Configuring maximum transmission unit (MTU) settings

7.2.9. Configuring MTU settings for a Cisco Catalyst switch

Complete the steps in this example procedure to enable jumbo frames on your Cisco Catalyst 3750 switch.

1. Review the current MTU settings:

```
sw01# show system mtu

System MTU size is 1600 bytes
System Jumbo MTU size is 1600 bytes
System Alternate MTU size is 1600 bytes
Routing MTU size is 1600 bytes
```

2. MTU settings are changed switch-wide on 3750 switches, and not for individual interfaces. Run the following commands to configure the switch to use jumbo frames of 9000 bytes. You might prefer to configure the MTU settings for individual interfaces, if your switch supports this feature.

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config)# system mtu jumbo 9000
Changes to the system jumbo MTU will not take effect until the next reload is done
```

NOTE

Remember to save your changes by copying the running-config to the startup-config: **copy running-config startup-config**.

3. Reload the switch to apply the change.

IMPORTANT

Reloading the switch causes a network outage for any devices that are dependent on the switch. Therefore, reload the switch only during a scheduled maintenance period.

```
sw01# reload
Proceed with reload? [confirm]
```

4. After the switch reloads, confirm the new jumbo MTU size.
   The exact output may differ depending on your switch model. For example, **System MTU** might apply to non-Gigabit interfaces, and **Jumbo MTU** might describe all Gigabit interfaces.

```
sw01# show system mtu

System MTU size is 1600 bytes
System Jumbo MTU size is 9000 bytes
System Alternate MTU size is 1600 bytes
Routing MTU size is 1600 bytes
```

7.2.10. About LLDP discovery

The **ironic-python-agent** service listens for LLDP packets from connected switches. The collected information can include the switch name, port details, and available VLANs. Similar to Cisco Discovery Protocol (CDP), LLDP assists with the discovery of physical hardware during the director introspection process.

7.2.11. Configuring LLDP for a Cisco Catalyst switch

Procedure

1. Run the **lldp run** command to enable LLDP globally on your Cisco Catalyst switch:

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config)# lldp run
```

2. View any neighboring LLDP-compatible devices:

```
sw01# show lldp neighbor
Capability codes:
  (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
  (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID          Local Intf     Hold-time  Capability      Port ID
DEP42037061562G3     Gi1/0/11       180        B,T          422037061562G3:P1

Total entries displayed: 1
```

NOTE

Remember to save your changes by copying the running-config to the startup-config: **copy running-config startup-config**.

7.3. CONFIGURING A CISCO NEXUS SWITCH

7.3.1. About trunk ports

With OpenStack Networking you can connect instances to the VLANs that already exist on your physical network. The term *trunk* is used to describe a port that allows multiple VLANs to traverse through the same port. Using these ports, VLANs can span across multiple switches, including virtual switches. For example, traffic tagged as VLAN110 in the physical network reaches the Compute node, where the 8021q module directs the tagged traffic to the appropriate VLAN on the vSwitch.

7.3.2. Configuring trunk ports for a Cisco Nexus switch

- If using a Cisco Nexus you might use the following configuration syntax to allow traffic for VLANs 110 and 111 to pass through to your instances.
  This configuration assumes that your physical node has an ethernet cable connected to interface **Ethernet1/12** on the physical switch.

  IMPORTANT

  These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

```
interface Ethernet1/12
  description Trunk to Compute Node
  switchport mode trunk
  switchport trunk allowed vlan 2,110,111
  switchport trunk native vlan 2
end
```

7.3.3. About access ports

Not all NICs on your Compute node carry instance traffic, and so you do not need to configure all NICs to allow multiple VLANs to pass through. Access ports require only one VLAN, and might fulfill other operational requirements, such as transporting management traffic or Block Storage data. These ports are commonly known as access ports and usually require a simpler configuration than trunk ports.

7.3.4. Configuring access ports for a Cisco Nexus switch

Procedure

- Using the example from the Figure 7.1, “Sample network layout” diagram, Ethernet1/13 (on a Cisco Nexus switch) is configured as an access port for **eth1**. This configuration assumes that your physical node has an ethernet cable connected to interface **Ethernet1/13** on the physical switch.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

```
interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
```

7.3.5. About LACP port aggregation

You can use Link Aggregation Control Protocol (LACP) to bundle multiple physical NICs together to form a single logical channel. Also known as 802.3ad (or bonding mode 4 in Linux), LACP creates a dynamic bond for load-balancing and fault tolerance. You must configure LACP at both physical ends: on the physical NICs, and on the physical switch ports.

Additional resources

- Network Interface Bonding in the *Installing and managing Red Hat OpenStack Platform with director* guide.

7.3.6. Configuring LACP on the physical NIC

You can configure Link Aggregation Control Protocol (LACP) on a physical NIC.

Procedure

1. Edit the */home/stack/network-environment.yaml* file:

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. Configure the Open vSwitch bridge to use LACP:

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

7.3.7. Configuring LACP for a Cisco Nexus switch

In this example, the Compute node has two NICs using VLAN 100:

Procedure

1. Physically connect the Compute node NICs to the switch (for example, ports 12 and 13).
2. Confirm that LACP is enabled:

   ```
   (config)# show feature | include lacp
   lacp                  1         enabled
   ```

3. Configure ports 1/12 and 1/13 as access ports, and as members of a channel group.
   Depending on your deployment, you can deploy trunk interfaces rather than access interfaces.

   For example, for Cisco UCI the NICs are virtual interfaces, so you might prefer to configure access ports exclusively. Often these interfaces contain VLAN tagging configurations.

   ```
   interface Ethernet1/13
    description Access port for Compute Node
    switchport mode access
    switchport access vlan 200
    channel-group 10 mode active

   interface Ethernet1/13
    description Access port for Compute Node
    switchport mode access
    switchport access vlan 200
    channel-group 10 mode active
   ```

NOTE

When you use PXE to provision nodes on Cisco switches, you might need to set the options **no lacp graceful-convergence** and **no lacp suspend-individual** to bring up the ports and boot the server. For more information, see your Cisco switch documentation.

7.3.8. About MTU settings

You must adjust your MTU size for certain types of network traffic. For example, jumbo frames (9000 bytes) are required for certain NFS or iSCSI traffic.

NOTE

You must change MTU settings from end-to-end on all hops that the traffic is expected to pass through, including any virtual switches.

Additional resources

- Configuring maximum transmission unit (MTU) settings

7.3.9. Configuring MTU settings for a Cisco Nexus 7000 switch

Apply MTU settings to a single interface on 7000-series switches.

Procedure

- Run the following commands to configure interface 1/12 to use jumbo frames of 9000 bytes:

```
interface ethernet 1/12
  mtu 9216
  exit
```

7.3.10. About LLDP discovery

The **ironic-python-agent** service listens for LLDP packets from connected switches. The collected information can include the switch name, port details, and available VLANs. Similar to Cisco Discovery Protocol (CDP), LLDP assists with the discovery of physical hardware during the director introspection process.

7.3.11. Configuring LLDP for a Cisco Nexus 7000 switch

Procedure

- You can enable LLDP for individual interfaces on Cisco Nexus 7000-series switches:

```
interface ethernet 1/12
  lldp transmit
  lldp receive
  no lacp suspend-individual
  no lacp graceful-convergence

interface ethernet 1/13
  lldp transmit
  lldp receive
  no lacp suspend-individual
  no lacp graceful-convergence
```

NOTE

Remember to save your changes by copying the running-config to the startup-config: **copy running-config startup-config**.

7.4. CONFIGURING A CUMULUS LINUX SWITCH

7.4.1. About trunk ports

With OpenStack Networking you can connect instances to the VLANs that already exist on your physical network. The term *trunk* is used to describe a port that allows multiple VLANs to traverse through the same port. Using these ports, VLANs can span across multiple switches, including virtual switches. For example, traffic tagged as VLAN110 in the physical network reaches the Compute node, where the 8021q module directs the tagged traffic to the appropriate VLAN on the vSwitch.

7.4.2. Configuring trunk ports for a Cumulus Linux switch

This configuration assumes that your physical node has transceivers connected to switch ports swp1 and swp2 on the physical switch.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

Procedure

- Use the following configuration syntax to allow traffic for VLANs 100 and 200 to pass through to your instances.

```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200
```

7.4.3. About access ports

Not all NICs on your Compute node carry instance traffic, and so you do not need to configure all NICs to allow multiple VLANs to pass through. Access ports require only one VLAN, and might fulfill other operational requirements, such as transporting management traffic or Block Storage data. These ports are commonly known as access ports and usually require a simpler configuration than trunk ports.

7.4.4. Configuring access ports for a Cumulus Linux switch

This configuration assumes that your physical node has an ethernet cable connected to the interface on the physical switch. Cumulus Linux switches use **eth** for management interfaces and **swp** for access/trunk ports.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

Procedure

- Using the example from the Figure 7.1, “Sample network layout” diagram, **swp1** (on a Cumulus Linux switch) is configured as an access port.

```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200


auto swp1
iface swp1
  bridge-access 100
```

```
auto swp2
iface swp2
  bridge-access 200
```

7.4.5. About LACP port aggregation

You can use Link Aggregation Control Protocol (LACP) to bundle multiple physical NICs together to form a single logical channel. Also known as 802.3ad (or bonding mode 4 in Linux), LACP creates a dynamic bond for load-balancing and fault tolerance. You must configure LACP at both physical ends: on the physical NICs, and on the physical switch ports.

Additional resources

- Network Interface Bonding in the *Installing and managing Red Hat OpenStack Platform with director* guide.

7.4.6. About MTU settings

You must adjust your MTU size for certain types of network traffic. For example, jumbo frames (9000 bytes) are required for certain NFS or iSCSI traffic.

NOTE

You must change MTU settings from end-to-end on all hops that the traffic is expected to pass through, including any virtual switches.

Additional resources

- Configuring maximum transmission unit (MTU) settings

7.4.7. Configuring MTU settings for a Cumulus Linux switch

Procedure

- This example enables jumbo frames on your Cumulus Linux switch.

```
auto swp1
iface swp1
  mtu 9000
```

NOTE

Remember to apply your changes by reloading the updated configuration: **sudo ifreload -a**

7.4.8. About LLDP discovery

The **ironic-python-agent** service listens for LLDP packets from connected switches. The collected information can include the switch name, port details, and available VLANs. Similar to Cisco Discovery Protocol (CDP), LLDP assists with the discovery of physical hardware during the director introspection process.

7.4.9. Configuring LLDP for a Cumulus Linux switch

By default, the LLDP service lldpd runs as a daemon and starts when the switch boots.

Procedure

- To view all LLDP neighbors on all ports/interfaces, run the following command:

```
cumulus@switch$ netshow lldp
Local Port  Speed  Mode        Remote Port   Remote Host Summary
----------  ---    ---------    -----  -----  ----------- --------
eth0        10G    Mgmt        ====   swp6   mgmt-sw     IP: 10.0.1.11/24
swp51       10G    Interface/L3 ====   swp1   spine01     IP: 10.0.0.11/32
swp52       10G    Interface/L  ====   swp1   spine02     IP: 10.0.0.11/32
```

7.5. CONFIGURING A EXTREME EXOS SWITCH

7.5.1. About trunk ports

With OpenStack Networking you can connect instances to the VLANs that already exist on your physical network. The term *trunk* is used to describe a port that allows multiple VLANs to traverse through the same port. Using these ports, VLANs can span across multiple switches, including virtual switches. For example, traffic tagged as VLAN110 in the physical network reaches the Compute node, where the 8021q module directs the tagged traffic to the appropriate VLAN on the vSwitch.

7.5.2. Configuring trunk ports on an Extreme Networks EXOS switch

If using an X-670 series switch, refer to the following example to allow traffic for VLANs 110 and 111 to pass through to your instances.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

Procedure

- This configuration assumes that your physical node has an ethernet cable connected to interface 24 on the physical switch. In this example, DATA and MNGT are the VLAN names.

```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

7.5.3. About access ports

Not all NICs on your Compute node carry instance traffic, and so you do not need to configure all NICs to allow multiple VLANs to pass through. Access ports require only one VLAN, and might fulfill other operational requirements, such as transporting management traffic or Block Storage data. These ports are commonly known as access ports and usually require a simpler configuration than trunk ports.

7.5.4. Configuring access ports for an Extreme Networks EXOS switch

This configuration assumes that your physical node has an ethernet cable connected to interface **10** on the physical switch.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

Procedure

- In this configuration example, on a Extreme Networks X-670 series switch, **10** is used as an access port for **eth1**.

```
create vlan VLANNAME tag NUMBER
configure vlan Default delete ports PORTSTRING
configure vlan VLANNAME add ports PORTSTRING untagged
```

For example:

```
#create vlan DATA tag 110
#configure vlan Default delete ports 10
#configure vlan DATA add ports 10 untagged
```

7.5.5. About LACP port aggregation

You can use Link Aggregation Control Protocol (LACP) to bundle multiple physical NICs together to form a single logical channel. Also known as 802.3ad (or bonding mode 4 in Linux), LACP creates a dynamic bond for load-balancing and fault tolerance. You must configure LACP at both physical ends: on the physical NICs, and on the physical switch ports.

Additional resources

- Network Interface Bonding in the *Installing and managing Red Hat OpenStack Platform with director* guide.

7.5.6. Configuring LACP on the physical NIC

You can configure Link Aggregation Control Protocol (LACP) on a physical NIC.

Procedure

1. Edit the */home/stack/network-environment.yaml* file:

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
```

```
    members:
      - type: interface
        name: nic3
        mtu: 9000
        primary: true
      - type: interface
        name: nic4
        mtu: 9000
```

2. Configure the Open vSwitch bridge to use LACP:

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

7.5.7. Configuring LACP on an Extreme Networks EXOS switch

Procedure

- In this example, the Compute node has two NICs using VLAN 100:

```
enable sharing MASTERPORT grouping ALL_LAG_PORTS lacp
configure vlan VLANNAME add ports PORTSTRING tagged
```

  For example:

```
#enable sharing 11 grouping 11,12 lacp
#configure vlan DATA add port 11 untagged
```

NOTE

You might need to adjust the timeout period in the LACP negotiation script. For more information, see https://gtacknowledge.extremenetworks.com/articles/How_To/LACP-configured-ports-interfere-with-PXE-DHCP-on-servers

7.5.8. About MTU settings

You must adjust your MTU size for certain types of network traffic. For example, jumbo frames (9000 bytes) are required for certain NFS or iSCSI traffic.

NOTE

You must change MTU settings from end-to-end on all hops that the traffic is expected to pass through, including any virtual switches.

Additional resources

- Configuring maximum transmission unit (MTU) settings

7.5.9. Configuring MTU settings on an Extreme Networks EXOS switch

Procedure

- Run the commands in this example to enable jumbo frames on an Extreme Networks EXOS switch and configure support for forwarding IP packets with 9000 bytes:

```
enable jumbo-frame ports PORTSTRING
configure ip-mtu 9000 vlan VLANNAME
```

Example

```
# enable jumbo-frame ports 11
# configure ip-mtu 9000 vlan DATA
```

7.5.10. About LLDP discovery

The **ironic-python-agent** service listens for LLDP packets from connected switches. The collected information can include the switch name, port details, and available VLANs. Similar to Cisco Discovery Protocol (CDP), LLDP assists with the discovery of physical hardware during the director introspection process.

7.5.11. Configuring LLDP settings on an Extreme Networks EXOS switch

Procedure

- In this example, LLDP is enabled on an Extreme Networks EXOS switch. **11** represents the port string:

```
enable lldp ports 11
```

7.6. CONFIGURING A JUNIPER EX SERIES SWITCH

7.6.1. About trunk ports

With OpenStack Networking you can connect instances to the VLANs that already exist on your physical network. The term *trunk* is used to describe a port that allows multiple VLANs to traverse through the same port. Using these ports, VLANs can span across multiple switches, including virtual switches. For example, traffic tagged as VLAN110 in the physical network reaches the Compute node, where the 8021q module directs the tagged traffic to the appropriate VLAN on the vSwitch.

7.6.2. Configuring trunk ports for a Juniper EX Series switch

Procedure

- If using a Juniper EX series switch running Juniper JunOS, use the following configuration syntax to allow traffic for VLANs 110 and 111 to pass through to your instances.

This configuration assumes that your physical node has an ethernet cable connected to interface ge-1/0/12 on the physical switch.

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

```
 ge-1/0/12 {
          description Trunk to Compute Node;
              unit 0 {
                  family ethernet-switching {
                      port-mode trunk;
                      vlan {
                          members [110 111];
                           }
                      native-vlan-id 2;
                  }
              }
}
```

7.6.3. About access ports

Not all NICs on your Compute node carry instance traffic, and so you do not need to configure all NICs to allow multiple VLANs to pass through. Access ports require only one VLAN, and might fulfill other operational requirements, such as transporting management traffic or Block Storage data. These ports are commonly known as access ports and usually require a simpler configuration than trunk ports.

7.6.4. Configuring access ports for a Juniper EX Series switch

This example on, a Juniper EX series switch, shows **ge-1/0/13** as an access port for **eth1**.

+

IMPORTANT

These values are examples. You must change the values in this example to match those in your environment. Copying and pasting these values into your switch configuration without adjustment can result in an unexpected outage.

Procedure

This configuration assumes that your physical node has an ethernet cable connected to interface **ge-1/0/13** on the physical switch.

+

```
 ge-1/0/13 {
          description Access port for Compute Node
              unit 0 {
                  family ethernet-switching {
                      port-mode access;
```

```
           vlan {
             members 200;
              }
           native-vlan-id 2;
         }
       }
}
```

7.6.5. About LACP port aggregation

You can use Link Aggregation Control Protocol (LACP) to bundle multiple physical NICs together to form a single logical channel. Also known as 802.3ad (or bonding mode 4 in Linux), LACP creates a dynamic bond for load-balancing and fault tolerance. You must configure LACP at both physical ends: on the physical NICs, and on the physical switch ports.

Additional resources

- Network Interface Bonding in the *Installing and managing Red Hat OpenStack Platform with director* guide.

7.6.6. Configuring LACP on the physical NIC

You can configure Link Aggregation Control Protocol (LACP) on a physical NIC.

Procedure

1. Edit the */home/stack/network-environment.yaml* file:

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. Configure the Open vSwitch bridge to use LACP:

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

Additional resources

- Network Interface Bonding in the *Customizing your Red Hat OpenStack Platform deployment* guide.

7.6.7. Configuring LACP for a Juniper EX Series switch

In this example, the Compute node has two NICs using VLAN 100.

Procedure

1. Physically connect the Compute node’s two NICs to the switch (for example, ports 12 and 13).

2. Create the port aggregate:

```
chassis {
    aggregated-devices {
        ethernet {
            device-count 1;
        }
    }
}
```

3. Configure switch ports 12 (ge-1/0/12) and 13 (ge-1/0/13) to join the port aggregate **ae1**:

```
interfaces {
    ge-1/0/12 {
        gigether-options {
            802.3ad ae1;
        }
    }
    ge-1/0/13 {
        gigether-options {
            802.3ad ae1;
            }
        }
}
```

> **NOTE**
>
> For Red Hat OpenStack Platform director deployments, in order to PXE boot from the bond, you must configure one of the bond members as lacp force-up toensure that only one bond member comes up during introspection and first boot. The bond member that you configure with lacp force-up must be the same bond member that has the MAC address in *instackenv.json* (the MAC address known to ironic must be the same MAC address configured with force-up).

4. Enable LACP on port aggregate **ae1**:

```
interfaces {
    ae1 {
        aggregated-ether-options {
            lacp {
                active;
            }
        }
    }
}
```

5. Add aggregate **ae1** to VLAN 100:

```
interfaces {
   ae1 {
      vlan-tagging;
      native-vlan-id 2;
      unit 100 {
         vlan-id 100;
      }
   }
}
```

6. Review your new port channel. The resulting output lists the new port aggregate **ae1** with member ports **ge-1/0/12** and **ge-1/0/13**:

```
> show lacp statistics interfaces ae1

Aggregated interface: ae1
LACP Statistics: LACP Rx LACP Tx Unknown Rx Illegal Rx
ge-1/0/12 0 0 0 0
ge-1/0/13 0 0 0 0
```

NOTE

Remember to apply your changes by running the **commit** command.

7.6.8. About MTU settings

You must adjust your MTU size for certain types of network traffic. For example, jumbo frames (9000 bytes) are required for certain NFS or iSCSI traffic.

NOTE

You must change MTU settings from end-to-end on all hops that the traffic is expected to pass through, including any virtual switches.

Additional resources

- Configuring maximum transmission unit (MTU) settings

7.6.9. Configuring MTU settings for a Juniper EX Series switch

This example enables jumbo frames on your Juniper EX4200 switch.

NOTE

The MTU value is calculated differently depending on whether you are using Juniper or Cisco devices. For example, **9216** on Juniper would equal to **9202** for Cisco. The extra bytes are used for L2 headers, where Cisco adds this automatically to the MTU value specified, but the usable MTU will be 14 bytes smaller than specified when using Juniper. So in order to support an MTU of **9000** on the VLANs, the MTU of **9014** would have to be configured on Juniper.

Procedure

1. For Juniper EX series switches, MTU settings are set for individual interfaces. These commands configure jumbo frames on the **ge-1/0/14** and **ge-1/0/15** ports:

   ```
   set interfaces ge-1/0/14 mtu 9216
   set interfaces ge-1/0/15 mtu 9216
   ```

   NOTE

   Remember to save your changes by running the **commit** command.

2. If using a LACP aggregate, you will need to set the MTU size there, and not on the member NICs. For example, this setting configures the MTU size for the ae1 aggregate:

   ```
    set interfaces ae1 mtu 9216
   ```

7.6.10. About LLDP discovery

The **ironic-python-agent** service listens for LLDP packets from connected switches. The collected information can include the switch name, port details, and available VLANs. Similar to Cisco Discovery Protocol (CDP), LLDP assists with the discovery of physical hardware during the director introspection process.

7.6.11. Configuring LLDP for a Juniper EX Series switch

You can enable LLDP globally for all interfaces, or just for individual ones.

Procedure

- Use the following too enable LLDP globally on your Juniper EX 4200 switch:

  ```
  lldp {
   interface all{
    enable;
   }
   }
  }
  ```

- Use the following to enable LLDP for the single interface **ge-1/0/14**:

  ```
  lldp {
   interface ge-1/0/14{
    enable;
   }
   }
  }
  ```

  NOTE

  Remember to apply your changes by running the **commit** command.

CHAPTER 8. CONFIGURING MAXIMUM TRANSMISSION UNIT (MTU) SETTINGS

8.1. MTU OVERVIEW

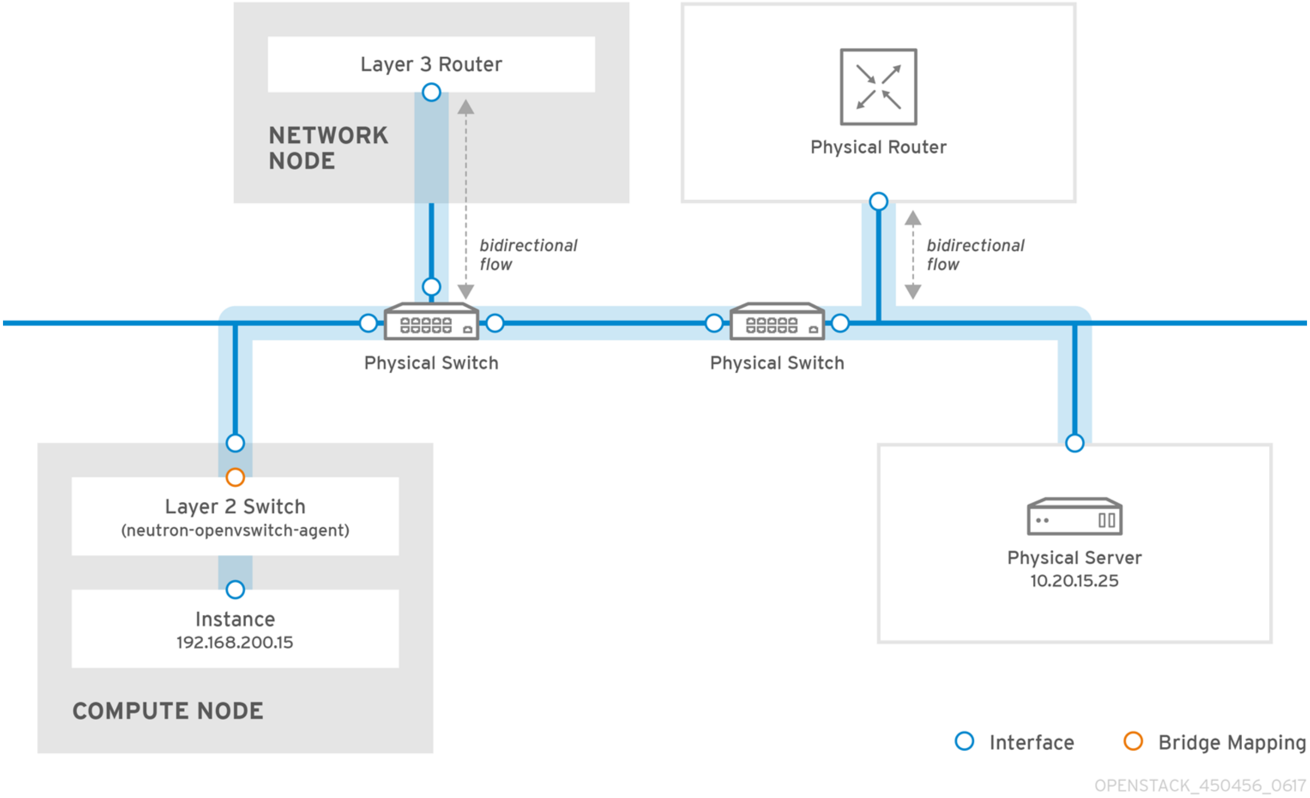
OpenStack Networking can calculate the largest possible maximum transmission unit (MTU) size that you can apply safely to instances. The MTU value specifies the maximum amount of data that a single network packet can transfer; this number is variable depending on the most appropriate size for the application. For example, NFS shares might require a different MTU size to that of a VoIP application.

NOTE

You can use the **openstack network show <network_name>** command to view the largest possible MTU values that OpenStack Networking calculates. **net-mtu** is a neutron API extension that is not present in some implementations. The MTU value that you require can be advertised to DHCPv4 clients for automatic configuration, if supported by the instance, as well as to IPv6 clients through Router Advertisement (RA) packets. To send Router Advertisements, the network must be attached to a router.

You must configure MTU settings consistently from end-to-end. This means that the MTU setting must be the same at every point the packet passes through, including the VM, the virtual network infrastructure, the physical network, and the destination server.

For example, the circles in the following diagram indicate the various points where an MTU value must be adjusted for traffic between an instance and a physical server. You must change the MTU value for very interface that handles network traffic to accommodate packets of a particular MTU size. This is necessary if traffic travels from the instance *192.168.200.15* through to the physical server *10.20.15.25*:



Inconsistent MTU values can result in several network issues, the most common being random packet loss that results in connection drops and slow network performance. Such issues are problematic to troubleshoot because you must identify and examine every possible network point to ensure it has the correct MTU value.

8.2. CONFIGURING MTU SETTINGS IN DIRECTOR

This example demonstrates how to set the MTU using the NIC config templates. You must set the MTU on the bridge, bond (if applicable), interface(s), and VLAN(s):

```
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              mtu: 9000    # <--- Set MTU
              members:
                -
                  type: ovs_bond
                  name: bond1
                  mtu: 9000    # <--- Set MTU
                  ovs_options: {get_param: BondInterfaceOvsOptions}
                  members:
                    -
                      type: interface
                      name: ens15f0
                      mtu: 9000    # <--- Set MTU
                      primary: true
                    -
                      type: interface
                      name: enp131s0f0
                      mtu: 9000    # <--- Set MTU
                -
                  type: vlan
                  device: bond1
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  mtu: 9000    # <--- Set MTU
                  addresses:
                  -
                    ip_netmask: {get_param: InternalApiIpSubnet}
                -
                  type: vlan
                  device: bond1
                  mtu: 9000    # <--- Set MTU
                  vlan_id: {get_param: TenantNetworkVlanID}
                  addresses:
                  -
                    ip_netmask: {get_param: TenantIpSubnet}
```

8.3. REVIEWING THE RESULTING MTU CALCULATION

You can view the calculated MTU value, which is the largest possible MTU value that instances can use. Use this calculated MTU value to configure all interfaces involved in the path of network traffic.

```
# openstack network show <network>
```

CHAPTER 9. USING QUALITY OF SERVICE (QOS) POLICIES TO MANAGE DATA TRAFFIC

You can offer varying service levels for VM instances by using quality of service (QoS) policies to apply rate limits to egress and ingress traffic on Red Hat OpenStack Platform (RHOSP) networks.

You can apply QoS policies to individual ports, or apply QoS policies to a project network, where ports with no specific policy attached inherit the policy.

NOTE

Internal network owned ports, such as DHCP and internal router ports, are excluded from network policy application.

You can apply, modify, or remove QoS policies dynamically. However, for guaranteed minimum bandwidth QoS policies, you can only apply modifications when there are no instances that use any of the ports the policy is assigned to.

9.1. QOS RULES

You can configure the following rule types to define a quality of service (QoS) policy in the Red Hat OpenStack Platform (RHOSP) Networking service (neutron):

Minimum bandwidth (`minimum_bandwidth`)

Provides minimum bandwidth constraints on certain types of traffic. If implemented, best efforts are made to provide no less than the specified bandwidth to each port on which the rule is applied.

Bandwidth limit (`bandwidth_limit`)

Provides bandwidth limitations on networks, ports, floating IPs (FIPs), and router gateway IPs. If implemented, any traffic that exceeds the specified rate is dropped.

DSCP marking (`dscp_marking`)

Marks network traffic with a Differentiated Services Code Point (DSCP) value.

QoS policies can be enforced in various contexts, including virtual machine instance placements, floating IP assignments, and gateway IP assignments.

Depending on the enforcement context and on the mechanism driver you use, a QoS rule affects egress traffic (upload from instance), ingress traffic (download to instance), or both.

NOTE

Starting with Red Hat OpenStack Platform (RHOSP) 17.1, in ML2/OVN deployments, you can enable minimum bandwidth and bandwidth limit egress policies for hardware offloaded ports. You cannot enable ingress policies for hardware offloaded ports. For more information, see Section 9.2, “Configuring the Networking service for QoS policies” .

Table 9.1. Supported traffic direction by driver (all QoS rule types)

Rule [8]	**Supported traffic direction by mechanism driver**		
	ML2/OVS	**ML2/SR-IOV**	**ML2/OVN**

Minimum bandwidth	Egress only [4][5]	Egress only	Currently, no support [6]
Bandwidth limit	Egress [1][2] and ingress	Egress only [3]	Egress and ingress
DSCP marking	Egress only	N/A	Egress only [7]

[1] The OVS egress bandwidth limit is performed in the TAP interface and is traffic policing, not traffic shaping.

[2] In RHOSP 16.2.2 and later, the OVS egress bandwidth limit is supported in hardware offloaded ports by applying the QoS policy in the network interface using **ip link** commands.

[3] The mechanism drivers ignore the **max-burst-kbits** parameter because they do not support it.

[4] Rule applies only to non-tunnelled networks: flat and VLAN.

[5] The OVS egress minimum bandwidth is supported in hardware offloaded ports by applying the QoS policy in the network interface using **ip link** commands.

[6] https://bugzilla.redhat.com/show_bug.cgi?id=2060310

[7] ML2/OVN does not support DSCP marking on tunneled protocols.

[8] RHOSP does not support QoS for trunk ports.

Table 9.2. Supported traffic direction by driver for placement reporting and scheduling (minimum bandwidth only)

Enforcement type	**Supported traffic by direction mechanism driver**		
	ML2/OVS	**ML2/SR-IOV**	**ML2/OVN**
Placement	Egress and ingress	Egress and ingress	Currently, no support

Table 9.3. Supported traffic direction by driver for enforcement types (bandwidth limit only)

Enforcement type	**Supported traffic direction by mechanism driver**	
	ML2/OVS	**ML2/OVN**
Floating IP	Egress and ingress	Egress and ingress
Gateway IP	Egress and ingress	Egress and ingress [1]

[1] Technology preview in RHOSP 17.1. See BZ 2088291.

Additional resources

- Creating and applying a bandwidth limit QoS policy and rule
- Creating and applying a guaranteed minimum bandwidth QoS policy and rule
- Creating and applying a DSCP marking QoS policy and rule for egress traffic

9.2. CONFIGURING THE NETWORKING SERVICE FOR QOS POLICIES

The quality of service feature in the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) is provided through the **qos** service plug-in. With the ML2/OVS and ML2/OVN mechanism drivers, **qos** is loaded by default. However, this is not true for ML2/SR-IOV.

When using the **qos** service plug-in with the ML2/OVS and ML2/SR-IOV mechanism drivers, you must also load the **qos** extension on their respective agents.

The following list summarizes the tasks that you must perform to configure the Networking service for QoS. The task details follow this list:

- For all types of QoS policies:
  - Add the **qos** service plug-in.
  - Add **qos** extension for the agents (OVS and SR-IOV only).
- In ML2/OVN deployments, you can enable minimum bandwidth and bandwidth limit egress policies for hardware offloaded ports. You cannot enable ingress policies for hardware offloaded ports.
- Additional tasks for scheduling VM instances using minimum bandwidth policies only:
  - Specify the hypervisor name if it differs from the name that the Compute service (nova) uses.
  - Configure the resource provider ingress and egress bandwidths for the relevant agents on each Compute node.
  - (Optional) Mark **vnic_types** as not supported.
- Additional task for DSCP marking policies on systems that use ML/OVS with tunneling only:
  - Set **dscp_inherit** to **true**.

Prerequisites

- Access to the undercloud host and credentials for the **stack** user.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Confirm that the **qos** service plug-in is not already loaded.

```
$ openstack network qos policy list
```

If the **qos** service plug-in is not loaded, then you receive a **ResourceNotFound** error. If you do not receive the error, then the plug-in is loaded and you do not need to perform the steps in this topic.

4. Create a YAML custom environment file.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

5. Your environment file must contain the keywords **parameter_defaults**. On a new line below **parameter_defaults** add **qos** to the **NeutronServicePlugins** parameter:

   ```
   parameter_defaults:
      NeutronServicePlugins: "qos"
   ```

6. If you use ML2/OVS and ML2/SR-IOV mechanism drivers, then you must also load the **qos** extension on the agent, by using either the **NeutronAgentExtensions** or the **NeutronSriovAgentExtensions** variable, respectively:

   - ML2/OVS

     ```
     parameter_defaults:
       NeutronServicePlugins: "qos"
       NeutronAgentExtensions: "qos"
     ```

   - ML2/SR-IOV

     ```
     parameter_defaults:
       NeutronServicePlugins: "qos"
       NeutronSriovAgentExtensions: "qos"
     ```

7. In ML2/OVN deployments, you can enable egress minimum and maximum bandwidth policies for hardware offloaded ports. To do this, set the **OvnHardwareOffloadedQos** parameter to **true**:

   ```
   parameter_defaults:
     NeutronServicePlugins: "qos"
     OvnHardwareOffloadedQos: true
   ```

8. If you want to schedule VM instances by using minimum bandwidth QoS policies, then you must also do the following:

   a. Add **placement** to the list of plug-ins and ensure the list also includes **qos**:

      ```
      parameter_defaults:
         NeutronServicePlugins: "qos,placement"
      ```

   b. If the hypervisor name matches the canonical hypervisor name used by the Compute service (nova), skip to step 7.iii.

If the hypervisor name does not match the canonical hypervisor name used by the Compute service, specify the alternative hypervisor name, using **resource_provider_default_hypervisor**:

- ML2/OVS

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
    Neutron::agents::ml2::ovs::resource_provider_default_hypervisor: %
{hiera('fqdn_canonical')}
```

- ML2/SR-IOV

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
    Neutron::agents::ml2::sriov::resource_provider_default_hypervisor: %
{hiera('fqdn_canonical')}
```

> IMPORTANT
>
> Another method for setting the alternative hypervisor name is to use **resource_provider_hypervisor**:
>
> - ML2/OVS
>
> ```
> parameter_defaults:
>   ExtraConfig:
>
> Neutron::agents::ml2::ovs::resource_provider_hypervisors:"ens5:%
> {hiera('fqdn_canonical')},ens6:%{hiera('fqdn_canonical')}"
> ```
>
> - ML2/SR-IOV
>
> ```
> parameter_defaults:
>   ExtraConfig:
>      Neutron::agents::ml2::sriov::resource_provider_hypervisors:
>      "ens5:%{hiera('fqdn_canonical')},ens6:%
> {hiera('fqdn_canonical')}"
> ```

c. Configure the resource provider ingress and egress bandwidths for the relevant agents on each Compute node that needs to provide a minimum bandwidth.
You can configure egress, ingress, or both, using the following formats:

- Configure only egress bandwidth, in kbps:

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<egress_kbps>:,<bridge1>:
<egress_kbps>:,...,<bridgeN>:<egress_kbps>:
```

- Configure only ingress bandwidth, in kbps:

```
NeutronOvsResourceProviderBandwidths: <bridge0>::<ingress_kbps>,<bridge1>::
<ingress_kbps>,...,<bridgeN>::<ingress_kbps>
```

- Configure both egress and ingress bandwidth, in kbps:

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<egress_kbps>:
<ingress_kbps>,<bridge1>:<egress_kbps>:<ingress_kbps>,...,<bridgeN>:
<egress_kbps>:<ingress_kbps>
```

Example - OVS agent

To configure the resource provider ingress and egress bandwidths for the OVS agent, add the following configuration to your network environment file:

```
parameter_defaults:
  ...
  NeutronBridgeMappings: physnet0:br-physnet0
  NeutronOvsResourceProviderBandwidths: br-physnet0:10000000:10000000
```

Example - SRIOV agent

To configure the resource provider ingress and egress bandwidths for the SRIOV agent, add the following configuration to your network environment file:

```
parameter_defaults:
  ...
  NeutronML2PhysicalNetworkMtus: physnet0:1500,physnet1:1500
  NeutronSriovResourceProviderBandwidths:
ens5:40000000:40000000,ens6:40000000:40000000
```

d. Optional: To mark **vnic_types** as not supported when multiple ML2 mechanism drivers support them by default and multiple agents are being tracked in the Placement service, also add the following configuration to your environment file:

Example - OVS agent

```
parameter_defaults:
  ...
  NeutronOvsVnicTypeBlacklist: direct
```

Example - SRIOV agent

```
parameter_defaults:
  ...
  NeutronSriovVnicTypeBlacklist: direct
```

9. If you want to create DSCP marking policies and use ML2/OVS with a tunneling protocol (VXLAN or GRE), then under **NeutronAgentExtensions**, add the following lines:

```
parameter_defaults:
  ...
  ControllerExtraConfig:
```

```
neutron::config::server_config:
  agent/dscp_inherit:
    value: true
```

When **dscp_inherit** is **true**, the Networking service copies the DSCP value of the inner header to the outer header.

10. Run the deployment command and include the core heat templates, other environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <other_environment_files> \
-e /home/stack/templates/my-neutron-environment.yaml
```

Verification

- Confirm that the **qos** service plug-in is loaded:

```
$ openstack network qos policy list
```

If the **qos** service plug-in is loaded, then you do not receive a **ResourceNotFound** error.

Additional resources

- Extension drivers for the RHOSP Networking service
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Section 9.3.1, “Using Networking service back-end enforcement to enforce minimum bandwidth”
- Section 9.3.2, “Scheduling instances by using minimum bandwidth QoS policies”
- Section 9.4, “Limiting network traffic by using QoS policies”
- Section 9.5, “Prioritizing network traffic by using DSCP marking QoS policies”

9.3. CONTROLLING MINIMUM BANDWIDTH BY USING QOS POLICIES

For the Red Hat OpenStack Platform (RHOSP) Networking service (neutron), a guaranteed minimum bandwidth QoS rule can be enforced in two distinct contexts: Networking service back-end enforcement and resource allocation scheduling enforcement.

The network back end, ML2/OVS or ML2/SR-IOV, attempts to guarantee that each port on which the rule is applied has no less than the specified network bandwidth.

When you use resource allocation scheduling bandwidth enforcement, the Compute service (nova) only places VM instances on hosts that support the minimum bandwidth.

You can apply QoS minumum bandwidth rules using Networking service back-end enforcement, resource allocation scheduling enforcement, or both.

The following table identifies the Modular Layer 2 (ML2) mechanism drivers that support minimum bandwidth QoS policies.

Table 9.4. ML2 mechanism drivers that support minimum bandwidth QoS

ML2 mechanism driver	Agent	VNIC types
ML2/SR-IOV	sriovnicswitch	direct
ML2/OVS	openvswitch	normal

Additional resources

- Section 9.3.1, “Using Networking service back-end enforcement to enforce minimum bandwidth”
- Section 9.3.2, “Scheduling instances by using minimum bandwidth QoS policies”

9.3.1. Using Networking service back-end enforcement to enforce minimum bandwidth

You can guarantee a minimum bandwidth for network traffic for ports by applying Red Hat OpenStack Platform (RHOSP) quality of service (QoS) policies to the ports. These ports must be backed by a flat or VLAN physical network.

> **NOTE**
>
> Currently, the Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN) does not support minimum bandwidth QoS rules.

Prerequisites

- The RHOSP Networking service (neutron) must have the **qos** service plug-in loaded. (This is the default.)
- Do not mix ports with and without bandwidth guarantees on the same physical interface, because this might cause denial of necessary resources (starvation) to the ports without a guarantee.

  TIP

  Create host aggregates to separate ports with bandwidth guarantees from those ports without bandwidth guarantees.

Procedure

1. Source your credentials file.

   Example

Example

```
$ source ~/overcloudrc
```

2. Confirm that the **qos** service plug-in is loaded in the Networking service:

```
$ openstack network qos policy list
```

If the **qos** service plug-in is not loaded, then you receive a **ResourceNotFound** error, and you must load the **qos** services plug-in before you can continue. For more information, see Section 9.2, “Configuring the Networking service for QoS policies” .

3. Identify the ID of the project you want to create the QoS policy for:

```
$ openstack project list
```

Sample output

```
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+----------------------------------+----------+
```

4. Using the project ID from the previous step, create a QoS policy for the project.

Example

In this example, a QoS policy named **guaranteed_min_bw** is created for the **admin** project:

```
$ openstack network qos policy create --share \
 --project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. Configure the rules for the policy.

Example

In this example, QoS rules for ingress and egress with a minimum bandwidth of **40000000** kbps are created for the policy named **guaranteed_min_bw**:

```
$ openstack network qos rule create \
 --type minimum-bandwidth --min-kbps 40000000 \
 --ingress guaranteed_min_bw

$ openstack network qos rule create \
 --type minimum-bandwidth --min-kbps 40000000 \
 --egress guaranteed_min_bw
```

6. Configure a port to apply the policy to.

Example

In this example, the **guaranteed_min_bw** policy is applied to port ID, **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12**:

```
$ openstack port set --qos-policy guaranteed_min_bw \
 56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

Verification

- ML2/SR-IOV
  Using root access, log in to the Compute node, and show the details of the virtual functions that are held in the physical function.

  #### Example

  ```
  # ip -details link show enp4s0f1
  ```

  #### Sample output

  ```
  50: enp4s0f1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 9000 qdisc mq
  master mx-bond state UP mode DEFAULT group default qlen 1000
      link/ether 98:03:9b:9d:73:74 brd ff:ff:ff:ff:ff:ff permaddr 98:03:9b:9d:73:75 promiscuity 0
  minmtu 68 maxmtu 9978
      bond_slave state BACKUP mii_status UP link_failure_count 0 perm_hwaddr
  98:03:9b:9d:73:75 queue_id 0 addrgenmode eui64 numtxqueues 320 numrxqueues 40
  gso_max_size 65536 gso_max_segs 65535 portname p1 switchid 74739d00039b0398
      vf 0     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 1     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 2     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 3     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 4     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 5     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 6     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 7     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
      vf 8     link/ether fa:16:3e:2a:d2:7f brd ff:ff:ff:ff:ff:ff, tx rate 999 (Mbps), max_tx_rate
  999Mbps, spoof checking off, link-state disable, trust off, query_rss off
      vf 9     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
  trust off, query_rss off
  ```

- ML2/OVS
  Using root access, log in to the compute node, show the **tc** rules and classes on the physical bridge interface.

  #### Example

  ```
  #  tc class show dev mx-bond
  ```

Sample output

```
class htb 1:11 parent 1:fffe prio 0 rate 4Gbit ceil 34359Mbit burst 9000b cburst 8589b
class htb 1:1 parent 1:fffe prio 0 rate 72Kbit ceil 34359Mbit burst 9063b cburst 8589b
class htb 1:fffe root rate 34359Mbit ceil 34359Mbit burst 8589b cburst 8589b
```

Additional resources

- network qos policy create in the *Command line interface reference*
- network qos rule create in the *Command line interface reference*
- port set in the *Command line interface reference*

9.3.2. Scheduling instances by using minimum bandwidth QoS policies

You can apply a minimum bandwidth QoS policy to a port to guarantee that the host on which its Red Hat OpenStack Platform (RHOSP) VM instance is spawned has a minimum network bandwidth.

Prerequisites

- The RHOSP Networking service (neutron) must have the **qos** and **placement** service plug-ins loaded. The **qos** service plug-in is loaded by default.
- The Networking service must support the following API extensions:
  - **agent-resources-synced**
  - **port-resource-request**
  - **qos-bw-minimum-ingress**
- You must use the ML2/OVS or ML2/SR-IOV mechanism drivers.
- You can only modify a minimum bandwidth QoS policy when there are no instances using any of the ports the policy is assigned to. The Networking service cannot update the Placement API usage information if a port is bound.
- The Placement service must support microversion 1.29.
- The Compute service (nova) must support microversion 2.72.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Confirm that the **qos** service plug-in is loaded in the Networking service:

   ```
   $ openstack network qos policy list
   ```

If the **qos** service plug-in is not loaded, then you receive a **ResourceNotFound** error, and you must load the **qos** services plug-in before you can continue. For more information, see Section 9.2, "Configuring the Networking service for QoS policies" .

3. Identify the ID of the project you want to create the QoS policy for:

```
$ openstack project list
```

Sample output

```
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+----------------------------------+----------+
```

4. Using the project ID from the previous step, create a QoS policy for the project.

Example

In this example, a QoS policy named **guaranteed_min_bw** is created for the **admin** project:

```
$ openstack network qos policy create --share \
 --project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. Configure the rules for the policy.

Example

In this example, QoS rules for ingress and egress with a minimum bandwidth of **40000000** kbps are created for the policy named **guaranteed_min_bw**:

```
$ openstack network qos rule create \
 --type minimum-bandwidth --min-kbps 40000000 \
 --ingress guaranteed_min_bw
$ openstack network qos rule create \
 --type minimum-bandwidth --min-kbps 40000000 \
 --egress guaranteed_min_bw
```

6. Configure a port to apply the policy to.

Example

In this example, the **guaranteed_min_bw** policy is applied to port ID, **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12**:

```
$ openstack port set --qos-policy guaranteed_min_bw \
 56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

Verification

1. Log in to the undercloud host as the stack user.

2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. List all the available resource providers:

```
$ openstack --os-placement-api-version 1.17 resource provider list
```

Sample output

```
+--------------------------------------+----------------------------------------------------+------------+----
--------------------------------+--------------------------------------+
| uuid                                 | name                                               | generation |
root_provider_uuid                   | parent_provider_uuid                 |
+--------------------------------------+----------------------------------------------------+------------+----
--------------------------------+--------------------------------------+
| 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | dell-r730-014.localdomain                          |
28 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | None                                 |
| 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | dell-r730-063.localdomain                          |
18 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | None                                 |
| e2f5082a-c965-55db-acb3-8daf9857c721 | dell-r730-063.localdomain:NIC Switch agent
|          0 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | 6b15ddce-13cf-4c85-a58f-
baec5b57ab52 |
| d2fb0ef4-2f45-53a8-88be-113b3e64ba1b | dell-r730-014.localdomain:NIC Switch agent
|          0 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | 31d3d88b-bc3a-41cd-9dc0-
fda54028a882 |
| f1ca35e2-47ad-53a0-9058-390ade93b73e | dell-r730-063.localdomain:NIC Switch
agent:enp6s0f1 |         13 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | e2f5082a-c965-55db-
acb3-8daf9857c721 |
| e518d381-d590-5767-8f34-c20def34b252 | dell-r730-014.localdomain:NIC Switch
agent:enp6s0f1 |         19 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | d2fb0ef4-2f45-53a8-
88be-113b3e64ba1b |
+--------------------------------------+----------------------------------------------------+------------+----
--------------------------------+--------------------------------------+
```

4. Check the bandwidth a specific resource provides.

```
(undercloud)$ openstack --os-placement-api-version 1.17 \
 resource provider inventory list <rp_uuid>
```

Example

In this example, the bandwidth provided by interface **enp6s0f1** on the host **dell-r730-014** is checked, using the resource provider UUID, **e518d381-d590-5767-8f34-c20def34b252**:

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 \
 resource provider inventory list e518d381-d590-5767-8f34-c20def34b252
```

Sample output

```
+----------------------------+------------------+----------+------------+----------+-----------+----------+
| resource_class             | allocation_ratio | min_unit |   max_unit | reserved | step_size |
total |
+----------------------------+------------------+----------+------------+----------+-----------+----------+
| NET_BW_EGR_KILOBIT_PER_SEC |          1.0 |        1 | 2147483647 |        0 |         1 |
10000000 |
| NET_BW_IGR_KILOBIT_PER_SEC |          1.0 |        1 | 2147483647 |        0 |         1 |
10000000 |
+----------------------------+------------------+----------+------------+----------+-----------+----------+
```

5. To check claims against the resource provider when instances are running, run the following command:

```
(undercloud)$ openstack --os-placement-api-version 1.17 \
 resource provider show --allocations  <rp_uuid>
```

Example

In this example, claims against the resource provider are checked on the host, **dell-r730-014**, using the resource provider UUID, **e518d381-d590-5767-8f34-c20def34b252**:

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 resource provider
show --allocations  e518d381-d590-5767-8f34-c20def34b252 -f value -c allocations
```

Sample output

```
{3cbb9e07-90a8-4154-8acd-b6ec2f894a83: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 8848b88b-4464-443f-bf33-5d4e49fd6204: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 9a29e946-698b-4731-bc28-89368073be1a: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, a6c83b86-9139-4e98-9341-dc76065136cc: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}, da60e33f-156e-47be-a632-870172ec5483: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, eb582a0e-8274-4f21-9890-9a0d55114663: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}}
```

Additional resources

- network qos policy create in the *Command line interface reference*
- network qos rule create in the *Command line interface reference*
- port set in the *Command line interface reference*

9.4. LIMITING NETWORK TRAFFIC BY USING QOS POLICIES

You can create a Red Hat OpenStack Platform (RHOSP) Networking service (neutron) quality of service (QoS) policy that limits the bandwidth on your RHOSP networks, ports, or floating IPs, and drops any traffic that exceeds the specified rate.

Prerequisites

- The Networking service must have the **qos** service plug-in loaded. (The plug-in is loaded by default.)

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

2. Confirm that the **qos** service plug-in is loaded in the Networking service:

   ```
   $ openstack network qos policy list
   ```

   If the **qos** service plug-in is not loaded, then you receive a **ResourceNotFound** error, and you must load the **qos** services plug-in before you can continue. For more information, see Section 9.2, "Configuring the Networking service for QoS policies" .

3. Identify the ID of the project you want to create the QoS policy for:

   ```
   $ openstack project list
   ```

   #### Sample output

   ```
   +----------------------------------+----------+
   | ID                               | Name     |
   +----------------------------------+----------+
   | 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
   | 519e6344f82e4c079c8e2eabb690023b | services |
   | 80bf5732752a41128e612fe615c886c6 | demo     |
   | 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
   +----------------------------------+----------+
   ```

4. Using the project ID from the previous step, create a QoS policy for the project.

   #### Example

   In this example, a QoS policy named **bw-limiter** is created for the **admin** project:

   ```
   $ openstack network qos policy create --share --project
   98a2f53c20ce4d50a40dac4a38016c69 bw-limiter
   ```

5. Configure the rules for the policy.

   #### NOTE

   You can add more than one rule to a policy, as long as the type or direction of each rule is different. For example, You can specify two bandwidth-limit rules, one with egress and one with ingress direction.

   #### Example

   In this example, QoS ingress and egress rules are created for the policy named **bw-limiter** with a bandwidth limit of **50000** kbps and a maximum burst size of **50000** kbps:

   ```
   $ openstack network qos rule create --type bandwidth-limit \
   ```

```
  --max-kbps 50000 --max-burst-kbits 50000 --ingress bw-limiter

$ openstack network qos rule create --type bandwidth-limit \
  --max-kbps 50000 --max-burst-kbits 50000 --egress bw-limiter
```

6. You can create a port with a policy attached to it, or attach a policy to a pre-existing port.

Example - create a port with a policy attached

In this example, the policy **bw-limiter** is associated with port **port2**:

```
$ openstack port create --qos-policy bw-limiter --network private port2
```

Sample output

```
+-----------------------+--------------------------------------------------+
| Field                 | Value                                            |
+-----------------------+--------------------------------------------------+
| admin_state_up        | UP                                               |
| allowed_address_pairs |                                                  |
| binding_host_id       |                                                  |
| binding_profile       |                                                  |
| binding_vif_details   |                                                  |
| binding_vif_type      | unbound                                          |
| binding_vnic_type     | normal                                           |
| created_at            | 2022-07-04T19:20:24Z                             |
| data_plane_status     | None                                             |
| description           |                                                  |
| device_id             |                                                  |
| device_owner          |                                                  |
| dns_assignment        | None                                             |
| dns_name              | None                                             |
| extra_dhcp_opts       |                                                  |
| fixed_ips             | ip_address='192.0.2.210', subnet_id='292f8c-...' |
| id                    | f51562ee-da8d-42de-9578-f6f5cb248226             |
| ip_address            | None                                             |
| mac_address           | fa:16:3e:d9:f2:ba                                |
| name                  | port2                                            |
| network_id            | 55dc2f70-0f92-4002-b343-ca34277b0234             |
| option_name           | None                                             |
| option_value          | None                                             |
| port_security_enabled | False                                            |
| project_id            | 98a2f53c20ce4d50a40dac4a38016c69                 |
| qos_policy_id         | 8491547e-add1-4c6c-a50e-42121237256c             |
| revision_number       | 6                                                |
| security_group_ids    | 0531cc1a-19d1-4cc7-ada5-49f8b08245be             |
| status                | DOWN                                             |
| subnet_id             | None                                             |
| tags                  | []                                               |
| trunk_details         | None                                             |
| updated_at            | 2022-07-04T19:23:00Z                             |
+-----------------------+--------------------------------------------------+
```

Example - attach a policy to a pre-existing port

In this example, the policy **bw-limiter** is associated with **port1**:

```
$ openstack port set --qos-policy bw-limiter port1
```

Verification

- Confirm that the bandwith limit policy is applied to the port.
  - Obtain the policy ID.

    Example

    In this example, the QoS policy, **bw-limiter** is queried:

    ```
    $ openstack network qos policy show bw-limiter
    ```

    Sample output

    ```
    +-------------------+-------------------------------------------------------------------+
    | Field             | Value                                                             |
    +-------------------+-------------------------------------------------------------------+
    | description       |                                                                   |
    | id                | 8491547e-add1-4c6c-a50e-42121237256c                              |
    | is_default        | False                                                             |
    | name              | bw-limiter                                                        |
    | project_id        | 98a2f53c20ce4d50a40dac4a38016c69                                  |
    | revision_number   | 4                                                                 |
    | rules             | [{u'max_kbps': 50000, u'direction': u'egress',                    |
    |                   |   u'type': u'bandwidth_limit',                                    |
    |                   |   u'id': u'0db48906-a762-4d32-8694-3f65214c34a6',                 |
    |                   |   u'max_burst_kbps': 50000,                                       |
    |                   |   u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'},     |
    |                   | [{u'max_kbps': 50000, u'direction': u'ingress',                   |
    |                   |   u'type': u'bandwidth_limit',                                    |
    |                   |   u'id': u'faabef24-e23a-4fdf-8e92-f8cb66998834',                 |
    |                   |   u'max_burst_kbps': 50000,                                       |
    |                   |   u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'}]     |
    | shared            | False                                                             |
    +-------------------+-------------------------------------------------------------------+
    ```

  - Query the port, and confirm that its policy ID matches the one obtained in the previous step.

    Example

    In this example, **port1** is queried:

    ```
    $ openstack port show port1
    ```

    Sample output

    ```
    +-------------------------+--------------------------------------------------------------------+
    | Field                   | Value                                                              |
    +-------------------------+--------------------------------------------------------------------+
    | admin_state_up          | UP                                                                 |
    ```

```
| allowed_address_pairs   | ip_address='192.0.2.128', mac_address='fa:16:3e:e1:eb:73'
|
| binding_host_id         | compute-2.redhat.local                                       |
| binding_profile         |                                                              |
| binding_vif_details     | port_filter='True'                                           |
| binding_vif_type        | ovs                                                          |
| binding_vnic_type       | normal                                                       |
| created_at              | 2022-07-04T19:07:56                                          |
| data_plane_status       | None                                                         |
| description             |                                                              |
| device_id               | 53abd2c4-955d-4b44-b6ad-f106e3f15df0                         |
| device_owner            | compute:nova                                                 |
| dns_assignment          | fqdn='host-192-0-2-213.openstacklocal.', hostname='my-host3',
|
|                         | ip_address='192.0.2.213'                                     |
| dns_domain              | None                                                         |
| dns_name                |                                                              |
| extra_dhcp_opts         |                                                              |
| fixed_ips               | ip_address='192.0.2..213', subnet_id='641d1db2-3b40-437b-b87b-
63   |
|                         | 079a7063ca'                                                  |
|                         | ip_address='2001:db8:0:f868:f816:3eff:fee1:eb73', subnet_id='c7ed0 |
|                         | 70a-d2ee-4380-baab-6978932a7dcc'                             |
| id                      | 56x9aiw1-2v74-144x-c2q8-ed8w423a6s12                         |
| location                | cloud='', project.domain_id=, project.domain_name=, project.id='7c |
|                         | b99d752fdb4944a2208ec9ee019226', project.name=,
region_name='regio |
|                         | nOne', zone=                                                 |
| mac_address             | fa:16:3e:e1:eb:73                                            |
| name                    | port2                                                        |
| network_id              | 55dc2f70-0f92-4002-b343-ca34277b0234                         |
| port_security_enabled   | True                                                         |
| project_id              | 98a2f53c20ce4d50a40dac4a38016c69                             |
| propagate_uplink_status | None                                                         |
| qos_policy_id           | 8491547e-add1-4c6c-a50e-42121237256c                         |
| resource_request        | None                                                         |
| revision_number         | 6                                                            |
| security_group_ids      | 4cdeb836-b5fd-441e-bd01-498d758704fd                         |
| status                  | ACTIVE                                                       |
| tags                    |                                                              |
| trunk_details           | None                                                         |
| updated_at              | 2022-07-04T19:11:41Z                                         |
+-------------------------+--------------------------------------------------------------+
```

Additional resources

- network qos rule create in the *Command line interface reference*
- network qos rule set in the *Command line interface reference*
- network qos rule delete in the *Command line interface reference*
- network qos rule list in the *Command line interface reference*

9.5. PRIORITIZING NETWORK TRAFFIC BY USING DSCP MARKING QOS POLICIES

You can use differentiated services code point (DSCP) to implement quality of service (QoS) policies on your Red Hat OpenStack Platform (RHOSP) network by embedding relevant values in the IP headers. The RHOSP Networking service (neutron) QoS policies can use DSCP marking to manage only egress traffic on neutron ports and networks.

Prerequisites

- The Networking service must have the **qos** service plug-in loaded. (This is the default.)
- You must use the ML2/OVS or ML2/OVN mechanism drivers.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Confirm that the **qos** service plug-in is loaded in the Networking service:

   ```
   $ openstack network qos policy list
   ```

   If the **qos** service plug-in is not loaded, then you receive a **ResourceNotFound** error, and you must configure the Networking service before you can continue. For more information, see Section 9.2, “Configuring the Networking service for QoS policies” .

3. Identify the ID of the project you want to create the QoS policy for:

   ```
   $ openstack project list
   ```

   Sample output

   ```
   +----------------------------------+----------+
   | ID                               | Name     |
   +----------------------------------+----------+
   | 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
   | 519e6344f82e4c079c8e2eabb690023b | services |
   | 80bf5732752a41128e612fe615c886c6 | demo     |
   | 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
   +----------------------------------+----------+
   ```

4. Using the project ID from the previous step, create a QoS policy for the project.

   Example

   In this example, a QoS policy named **qos-web-servers** is created for the **admin** project:

   ```
   openstack network qos policy create --project 98a2f53c20ce4d50a40dac4a38016c69 qos-web-servers
   ```

5. Create a DSCP rule and apply it to a policy.

   ### Example

   In this example, a DSCP rule is created using DSCP mark **18** and is applied to the **qos-web-servers** policy:

   ```
   openstack network qos rule create --type dscp-marking --dscp-mark 18 qos-web-servers
   ```

   ### Sample output

   ```
   Created a new dscp_marking_rule:
   +-----------+--------------------------------------+
   | Field     | Value                                |
   +-----------+--------------------------------------+
   | dscp_mark | 18                                   |
   | id        | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
   +-----------+--------------------------------------+
   ```

6. You can change the DSCP value assigned to a rule.

   ### Example

   In this example, the DSCP mark value is changed to 22 for the rule, **d7f976ec-7fab-4e60-af70-f59bf88198e6**, in the **qos-web-servers** policy:

   ```
   $ openstack network qos rule set --dscp-mark 22 qos-web-servers d7f976ec-7fab-4e60-af70-
   f59bf88198e6
   ```

7. You can delete a DSCP rule.

   ### Example

   In this example, the DSCP rule, **d7f976ec-7fab-4e60-af70-f59bf88198e6**, in the **qos-web-servers** policy is deleted:

   ```
   $ openstack network qos rule delete qos-web-servers d7f976ec-7fab-4e60-af70-
   f59bf88198e6
   ```

Verification

- Confirm that the DSCP rule is applied to the QoS policy.

  ### Example

  In this example, the DSCP rule, **d7f976ec-7fab-4e60-af70-f59bf88198e6** is applied to the QoS policy, **qos-web-servers**:

  ```
  $ openstack network qos rule list qos-web-servers
  ```

  ### Sample output

  ```
  +-----------+--------------------------------------+
  | dscp_mark | id                                   |
  ```

```
+-----------+--------------------------------------+
|        18 | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----------+--------------------------------------+
```

Additional resources

- network qos rule create in the *Command line interface reference*
- network qos rule set in the *Command line interface reference*
- network qos rule delete in the *Command line interface reference*
- network qos rule list in the *Command line interface reference*

9.6. APPLYING QOS POLICIES TO PROJECTS BY USING NETWORKING SERVICE RBAC

With the Red Hat OpenStack Platform (RHOSP) Networking service (neutron), you can add a role-based access control (RBAC) for quality of service (QoS) policies. As a result, you can apply QoS policies to individual projects.

Prerequisities

- You must have one or more QoS policies available.

Procedure

- Create an RHOSP Networking service RBAC policy associated with a specific QoS policy, and assign it to a specific project:

  ```
  $ openstack network rbac create --type qos_policy --target-project <project_name |
  project_ID> --action access_as_shared <QoS_policy_name | QoS_policy_ID>
  ```

  ### Example

  For example, you might have a QoS policy that allows for lower-priority network traffic, named **bw-limiter**. Using a RHOSP Networking service RBAC policy, you can apply the QoS policy to a specific project:

  ```
  $ openstack network rbac create --type qos_policy --target-project
  80bf5732752a41128e612fe615c886c6 --action access_as_shared bw-limiter
  ```

Additional resources

- network rbac create in the *Command line interface reference*
- Section 9.3.1, “Using Networking service back-end enforcement to enforce minimum bandwidth”
- Section 9.3.2, “Scheduling instances by using minimum bandwidth QoS policies”
- Section 9.4, “Limiting network traffic by using QoS policies”
- Section 9.5, “Prioritizing network traffic by using DSCP marking QoS policies”

CHAPTER 10. CONFIGURING BRIDGE MAPPINGS

In Red Hat OpenStack Platform (RHOSP), a bridge mapping associates a physical network name (an interface label) to a bridge created with the Modular Layer 2 plug-in mechanism drivers Open vSwitch (OVS) or Open Virtual Network (OVN). The RHOSP Networking service (neutron) uses bridge mappings to allow provider network traffic to reach the physical network.

The topics included in this section are:

- Section 10.1, “Overview of bridge mappings”
- Section 10.2, “Traffic flow”
- Section 10.3, “Configuring bridge mappings”
- Section 10.4, “Maintaining bridge mappings for OVS”
- Section 10.4.1, “Cleaning up OVS patch ports manually”
- Section 10.4.2, “Cleaning up OVS patch ports automatically”

10.1. OVERVIEW OF BRIDGE MAPPINGS

In the Red Hat OpenStack Platform (RHOSP) Networking service (neutron), you use bridge mappings to allow provider network traffic to reach the physical network. Traffic leaves the provider network from the **qg-xxx** interface of the router and arrives at the intermediate bridge (**br-int**).

The next part of the data path varies depending on which mechanism driver your deployment uses:

- ML2/OVS: a patch port between **br-int** and **br-ex** allows the traffic to pass through the bridge of the provider network and out to the physical network.
- ML2/OVN: the Networking service creates a patch port on a hypervisor only when there is a VM bound to the hypervisor and the VM requires the port.

You configure the bridge mapping on the network node on which the router is scheduled. Router traffic can egress using the correct physical network, as represented by the provider network.

NOTE

The Networking service supports only one bridge for each physical network. Do not map more than one physical network to the same bridge.

10.2. TRAFFIC FLOW

Each external network is represented by an internal VLAN ID, which is tagged to the router **qg-xxx** port. When a packet reaches **phy-br-ex**, the **br-ex** port strips the VLAN tag and moves the packet to the physical interface and then to the external network.

The return packet from the external network arrives on **br-ex** and moves to **br-int** using **phy-br-ex <-> int-br-ex**. When the packet is going through **br-ex** to **br-int**, the packet’s external VLAN ID is replaced by an internal VLAN tag in **br-int**, and this allows **qg-xxx** to accept the packet.

In the case of egress packets, the packet’s internal VLAN tag is replaced with an external VLAN tag in **br-ex** (or in the external bridge that is defined in the **NeutronNetworkVLANRanges** parameter).

10.3. CONFIGURING BRIDGE MAPPINGS

To modify the bridge mappings that the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) uses to connect provider network traffic with the physical network, you modify the necessary heat parameters and redeploy your overcloud.

Prerequisites

- You must be able to access the underclod host as the **stack** user.
- You must configure bridge mappings on the network node on which the router is scheduled.
- You must also configure bridge mappings for your Compute nodes.

Procedure

1. Log in to the undercloud host as the stack user.
2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a custom YAML environment file.

   #### Example

   ```
   $ vi /home/stack/templates/my_bridge_mappings.yaml
   ```

4. Your environment file must contain the keywords **parameter_defaults**. Add the **NeutronBridgeMappings** heat parameter with values that are appropriate for your site after the **parameter_defaults** keyword.

   #### Example

   In this example, the **NeutronBridgeMappings** parameter associates the physical names, **datacentre** and **tenant**, the bridges **br-ex** and **br-tenant**, respectively.

   ```
   parameter_defaults:
     NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
   ```

   > NOTE
   >
   > When the **NeutronBridgeMappings** parameter is not used, the default maps the external bridge on hosts (br-ex) to a physical name (datacentre).

5. If you are using a flat network, add its name using the **NeutronFlatNetworks** parameter.

   #### Example

   In this example, the parameter associates physical name **datacentre** with bridge **br-ex**, and physical name **tenant** with bridge br-tenant."

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
  NeutronFlatNetworks: "my_flat_network"
```

NOTE

When the **NeutronFlatNetworks** parameter is not used, the default is **datacentre**.

6. If you are using a VLAN network, specify the network name along with the range of VLANs it accesses by using the **NeutronNetworkVLANRanges** parameter.

   ### Example

   In this example, the **NeutronNetworkVLANRanges** parameter specifies the VLAN range of **1 - 1000** for the **tenant** network:

   ```
   parameter_defaults:
    NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
    NeutronNetworkVLANRanges: "tenant:1:1000"
   ```

7. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   ```
   $ openstack overcloud deploy --templates \
     -e <your_environment_files> \
     -e /home/stack/templates/my_bridge_mappings.yaml
   ```

8. Perform the following steps:

   a. Using the network VLAN ranges, create the provider networks that represent the corresponding external networks. (You use the physical name when creating neutron provider networks or floating IP networks.)

   b. Connect the external networks to your project networks with router interfaces.

Additional resources

- Updating the format of your network configuration files in the *Installing and managing Red Hat OpenStack Platform with director* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

10.4. MAINTAINING BRIDGE MAPPINGS FOR OVS

After removing any OVS bridge mappings, you must perform a subsequent cleanup to ensure that the bridge configuration is cleared of any associated patch port entries. You can perform this operation in the following ways:

- Manual port cleanup - requires careful removal of the superfluous patch ports. No outages of network connectivity are required.

- Automated port cleanup - performs an automated cleanup, but requires an outage, and requires that the necessary bridge mappings be re-added. Choose this option during scheduled maintenance windows when network connectivity outages can be tolerated.

> NOTE
>
> When OVN bridge mappings are removed, the OVN controller automatically cleans up any associated patch ports.

10.4.1. Cleaning up OVS patch ports manually

After removing any OVS bridge mappings, you must also remove the associated patch ports.

Prerequisites

- The patch ports that you are cleaning up must be Open Virtual Switch (OVS) ports.
- A system outage is **not** required to perform a manual patch port cleanup.
- You can identify the patch ports to cleanup by their naming convention:
  - In **br-$external_bridge** patch ports are named **phy-<external bridge name>** (for example, phy-br-ex2).
  - In **br-int** patch ports are named **int-<external bridge name>** (for example, **int-br-ex2**).

Procedure

1. Use **ovs-vsctl** to remove the OVS patch ports associated with the removed bridge mapping entry:

```
# ovs-vsctl del-port br-ex2 datacentre
# ovs-vsctl del-port br-tenant tenant
```

2. Restart **neutron-openvswitch-agent**:

```
# service neutron-openvswitch-agent restart
```

10.4.2. Cleaning up OVS patch ports automatically

After removing any OVS bridge mappings, you must also remove the associated patch ports.

> NOTE
>
> When OVN bridge mappings are removed, the OVN controller automatically cleans up any associated patch ports.

Prerequisites

- The patch ports that you are cleaning up must be Open Virtual Switch (OVS) ports.
- Cleaning up patch ports automatically with the **neutron-ovs-cleanup** command causes a network connectivity outage, and should be performed only during a scheduled maintenance window.

- Use the flag **--ovs_all_ports** to remove all patch ports from **br-int**, cleaning up tunnel ends from **br-tun**, and patch ports from bridge to bridge.
- The **neutron-ovs-cleanup** command unplugs all patch ports (instances, qdhcp/qrouter, among others) from all OVS bridges.

Procedure

1. Run the **neutron-ovs-cleanup** command with the **--ovs_all_ports** flag.

   IMPORTANT

   Performing this step will result in a total networking outage.

   ```
   # /usr/bin/neutron-ovs-cleanup
   --config-file /etc/neutron/plugins/ml2/openvswitch_agent.ini
   --log-file /var/log/neutron/ovs-cleanup.log --ovs_all_ports
   ```

2. Restore connectivity by redeploying the overcloud.
   When you rerun the **openstack overcloud deploy** command, your bridge mapping values are reapplied.

   NOTE

   After a restart, the OVS agent does not interfere with any connections that are not present in bridge_mappings. So, if you have **br-int** connected to **br-ex2**, and **br-ex2** has some flows on it, removing **br-int** from the bridge_mappings configuration does not disconnect the two bridges when you restart the OVS agent or the node.

Additional resources

- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

CHAPTER 11. VLAN-AWARE INSTANCES

11.1. VLAN TRUNKS AND VLAN TRANSPARENT NETWORKS

VM instances can send and receive VLAN-tagged traffic over a single virtual NIC. This is particularly useful for NFV applications (VNFs) that expect VLAN-tagged traffic, allowing a single virtual NIC to serve multiple customers or services.

In ML2/OVN deployments you can support VLAN-aware instances using VLAN transparent networks. As an alternative in ML2/OVN or ML2/OVS deployments, you can support VLAN-aware instances using trunks.

In a VLAN transparent network, you set up VLAN tagging in the VM instances. The VLAN tags are transferred over the network and consumed by the instances on the same VLAN, and ignored by other instances and devices. In a VLAN transparent network, the VLANs are managed in the instance. You do not need to set up the VLAN in the OpenStack Networking Service (neutron).

VLAN trunks support VLAN-aware instances by combining VLANs into a single trunked port. For example, a project data network can use VLANs or tunneling (VXLAN, GRE, or Geneve) segmentation, while the instances see the traffic tagged with VLAN IDs. Network packets are tagged immediately before they are injected to the instance and do not need to be tagged throughout the entire network.

The following table compares certain features of VLAN transparent networks and VLAN trunks.

	Transparent	Trunk
Mechanism driver support	ML2/OVN	ML2/OVN, ML2/OVS
VLAN setup managed by	VM instance	OpenStack Networking Service (neutron)
IP assignment	Configured in VM instance	Assigned by DHCP
VLAN ID	Flexible. You can set the VLAN ID in the instance	Fixed. Instances must use the VLAN ID configured in the trunk

11.2. ENABLING VLAN TRANSPARENCY IN ML2/OVN DEPLOYMENTS

Enable VLAN transparency if you need to send VLAN tagged traffic between virtual machine (VM) instances. In a VLAN transparent network you can configure the VLANS directly in the VMs without configuring them in neutron.

Prerequisites

- Deployment of Red Hat OpenStack Platform 16.1 or higher, with ML2/OVN as the mechanism driver.
- Provider network of type VLAN or Geneve. Do not use VLAN transparency in deployments with flat type provider networks.

- Ensure that the external switch supports 802.1q VLAN stacking using ethertype 0x8100 on both VLANs. OVN VLAN transparency does not support 802.1ad QinQ with outer provider VLAN ethertype set to 0x88A8 or 0x9100.
- You must have RHOSP administrator privileges.

Procedure

1. Log in to the undercloud host as the stack user.
2. Source the stackrc undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. In an environment file on the undercloud node, set the **EnableVLANTransparency** parameter to **true**. For example, add the following lines to **ovn-extras.yaml**.

   ```
   parameter_defaults:
     EnableVLANTransparency: true
   ```

4. Include the environment file in the **openstack overcloud deploy** command with any other environment files that are relevant to your environment and deploy the overcloud:

   ```
   $ openstack overcloud deploy \
   --templates \
   …
   -e <other_overcloud_environment_files> \

   -e ovn-extras.yaml \
   …
   ```

   Replace **<other_overcloud_environment_files>** with the list of environment files that are part of your existing deployment.

5. Create the network using the **--transparent-vlan** argument.

   #### Example

   ```
   $ openstack network create network-name --transparent-vlan
   ```

6. Set up a VLAN interface on each participating VM.
   Set the interface MTU to 4 bytes less than the MTU of the underlay network to accommodate the extra tagging required by VLAN transparency. For example, if the underlay network MTU is 1500, set the interface MTU to 1496.

   The following example command adds a VLAN interface on **eth0** with an MTU of 1496. The VLAN is 50 and the interface name is **vlan50**:

   #### Example

   ```
   $ ip link add link eth0 name vlan50 type vlan id 50 mtu 1496
   $ ip link set vlan50 up
   $ ip addr add 192.128.111.3/24 dev vlan50
   ```

7. Choose one of these alternatives for the IP address you created on the VLAN interface inside the VM in step 4:

    - Set an allowed address pair on the VM port.

      ### Example

      The following example sets an allowed address pair on port, **fv82gwk3-qq2e-yu93-go31-56w7sf476mm0**, by using **192.128.111.3** and optionally adding a MAC address, **00:40:96:a8:45:c4**:

      ```
      $ openstack port set --allowed-address \
      ip-address=192.128.111.3,mac-address=00:40:96:a8:45:c4 \
      fv82gwk3-qq2e-yu93-go31-56w7sf476mm0
      ```

    - Disable port security on the port.
      Disabling port security provides a practical alternative when it is not possible to list all of the possible combinations in allowed address pairs.

      ### Example

      The following example disables port security on port **fv82gwk3-qq2e-yu93-go31-56w7sf476mm0**:

      ```
      $ openstack port set --no-security-group \
      --disable-port-security \
      fv82gwk3-qq2e-yu93-go31-56w7sf476mm0
      ```

Verification

1. Ping between two VMs on the VLAN using the vlan50 IP address.

2. Use **tcpdump** on **eth0** to see if the packets arrive with the VLAN tag intact.

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide

- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

- port set in the *Command line interface reference*

11.3. REVIEWING THE TRUNK PLUG-IN

During a Red Hat openStack deployment, the trunk plug-in is enabled by default. You can review the configuration on the controller nodes:

- On the controller node, confirm that the trunk plug-in is enabled in the /var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf file:

  ```
  service_plugins=router,qos,trunk
  ```

11.4. CREATING A TRUNK CONNECTION

To implement trunks for VLAN-tagged traffic, create a parent port and attach the new port to an existing neutron network. When you attach the new port, OpenStack Networking adds a trunk connection to the parent port you created. Next, create subports. These subports connect VLANs to instances, which allow connectivity to the trunk. Within the instance operating system, you must also create a sub-interface that tags traffic for the VLAN associated with the subport.

1. Identify the network that contains the instances that require access to the trunked VLANs. In this example, this is the *public* network:

```
openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 82845092-4701-4004-add7-838837837621 | private | 434c7982-cd96-4c41-a8c9-
b93adbdcb197 |
| 8d8bc6d6-5b28-4e00-b99e-157516ff0050 | public  | 3fd811b4-c104-44b5-8ff8-
7a86af5e332c |
+--------------------------------------+---------+--------------------------------------+
```

2. Create the parent trunk port, and attach it to the network that the instance connects to. In this example, create a neutron port named parent-trunk-port on the *public* network. This trunk is the *parent* port, as you can use it to create *subports*.

```
openstack port create --network public parent-trunk-port
+-----------------------+-----------------------------------------------------------------------------+
| Field                 | Value                                                                       |
+-----------------------+-----------------------------------------------------------------------------+
| admin_state_up        | UP                                                                          |
| allowed_address_pairs |                                                                             |
| binding_host_id       |                                                                             |
| binding_profile       |                                                                             |
| binding_vif_details   |                                                                             |
| binding_vif_type      | unbound                                                                     |
| binding_vnic_type     | normal                                                                      |
| created_at            | 2016-10-20T02:02:33Z                                                        |
| description           |                                                                             |
| device_id             |                                                                             |
| device_owner          |                                                                             |
| extra_dhcp_opts       |                                                                             |
| fixed_ips             | ip_address='172.24.4.230', subnet_id='dc608964-9af3-4fed-9f06-
6d3844fb9b9b' |
| headers               |                                                                             |
| id                    | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39                                        |
| mac_address           | fa:16:3e:33:c4:75                                                           |
| name                  | parent-trunk-port                                                           |
| network_id            | 871a6bd8-4193-45d7-a300-dcb2420e7cc3                                        |
| project_id            | 745d33000ac74d30a77539f8920555e7                                            |
| project_id            | 745d33000ac74d30a77539f8920555e7                                            |
| revision_number       | 4                                                                           |
| security_groups       | 59e2af18-93c6-4201-861b-19a8a8b79b23                                        |
| status                | DOWN                                                                        |
| updated_at            | 2016-10-20T02:02:33Z                                                        |
+-----------------------+-----------------------------------------------------------------------------+
```

3. Create a trunk using the port that you created in step 2. In this example the trunk is named **parent-trunk**.

```
openstack network trunk create --parent-port parent-trunk-port parent-trunk
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| created_at      | 2016-10-20T02:05:17Z                 |
| description     |                                      |
| id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name            | parent-trunk                         |
| port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                                    |
| status          | DOWN                                 |
| sub_ports       |                                      |
| tenant_id       | 745d33000ac74d30a77539f8920555e7     |
| updated_at      | 2016-10-20T02:05:17Z                 |
+-----------------+--------------------------------------+
```

4. View the trunk connection:

```
openstack network trunk list
+--------------------------------------+--------------+--------------------------------------+-------------+
| ID                                   | Name         | Parent Port                          | Description |
+--------------------------------------+--------------+--------------------------------------+-------------+
| 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 | parent-trunk | 20b6fdf8-0d43-475a-a0f1-
ec8f757a4a39 |             |
+--------------------------------------+--------------+--------------------------------------+-------------+
```

5. View the details of the trunk connection:

```
openstack network trunk show parent-trunk
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| created_at      | 2016-10-20T02:05:17Z                 |
| description     |                                      |
| id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name            | parent-trunk                         |
| port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                                    |
| status          | DOWN                                 |
| sub_ports       |                                      |
| tenant_id       | 745d33000ac74d30a77539f8920555e7     |
| updated_at      | 2016-10-20T02:05:17Z                 |
+-----------------+--------------------------------------+
```

11.5. ADDING SUBPORTS TO THE TRUNK

1. Create a neutron port.
   This port is a subport connection to the trunk. You must also specify the MAC address that you assigned to the parent port:

```
openstack port create --network private --mac-address fa:16:3e:33:c4:75 subport-trunk-port
+-----------------------+--------------------------------------------------------------------------+
| Field                 | Value                                                                    |
+-----------------------+--------------------------------------------------------------------------+
| admin_state_up        | UP                                                                       |
| allowed_address_pairs |                                                                          |
| binding_host_id       |                                                                          |
| binding_profile       |                                                                          |
| binding_vif_details   |                                                                          |
| binding_vif_type      | unbound                                                                  |
| binding_vnic_type     | normal                                                                   |
| created_at            | 2016-10-20T02:08:14Z                                                     |
| description           |                                                                          |
| device_id             |                                                                          |
| device_owner          |                                                                          |
| extra_dhcp_opts       |                                                                          |
| fixed_ips             | ip_address='10.0.0.11', subnet_id='1a299780-56df-4c0b-a4c0-
c5a612cef2e8' |
| headers               |                                                                          |
| id                    | 479d742e-dd00-4c24-8dd6-b7297fab3ee9                                     |
| mac_address           | fa:16:3e:33:c4:75                                                        |
| name                  | subport-trunk-port                                                       |
| network_id            | 3fe6b758-8613-4b17-901e-9ba30a7c4b51                                     |
| project_id            | 745d33000ac74d30a77539f8920555e7                                         |
| project_id            | 745d33000ac74d30a77539f8920555e7                                         |
| revision_number       | 4                                                                        |
| security_groups       | 59e2af18-93c6-4201-861b-19a8a8b79b23                                     |
| status                | DOWN                                                                     |
| updated_at            | 2016-10-20T02:08:15Z                                                     |
+-----------------------+--------------------------------------------------------------------------+
```

NOTE

If you receive the error **HttpException: Conflict**, confirm that you are creating the subport on a different network to the one that has the parent trunk port. This example uses the public network for the parent trunk port, and private for the subport.

2. Associate the port with the trunk (**parent-trunk**), and specify the VLAN ID (**55**):

```
openstack network trunk set --subport port=subport-trunk-port,segmentation-
type=vlan,segmentation-id=55 parent-trunk
```

11.6. CONFIGURING AN INSTANCE TO USE A TRUNK

You must configure the VM instance operating system to use the MAC address that the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) assigned to the subport. You can also configure the subport to use a specific MAC address during the subport creation step.

Prerequisites

- If you are performing live migrations of your Compute nodes, ensure that the RHOSP Networking service RPC response timeout is appropriately set for your RHOSP deployment.

The RPC response timeout value can vary between sites and is dependent on the system speed. The general recommendation is to set the value to at least 120 seconds per/100 trunk ports.

The best practice is to measure the trunk port bind process time for your RHOSP deployment, and then set the RHOSP Networking service RPC response timeout appropriately. Try to keep the RPC response timeout value low, but also provide enough time for the RHOSP Networking service to receive an RPC response. For more information, see Section 11.7, "Configuring Networking service RPC timeout".

Procedure

1. Review the configuration of your network trunk, using the **network trunk** command.

   #### Example

   ```
   $ openstack network trunk list
   ```

   #### Sample output

   ```
   +---------------------+--------------+---------------------+-------------+
   | ID                  | Name         | Parent Port         | Description |
   +---------------------+--------------+---------------------+-------------+
   | 0e4263e2-5761-4cf6- | parent-trunk | 20b6fdf8-0d43-475a- |             |
   | ab6d-b22884a0fa88   |              | a0f1-ec8f757a4a39   |             |
   +---------------------+--------------+---------------------+-------------+
   ```

   #### Example

   ```
   $ openstack network trunk show parent-trunk
   ```

   #### Sample output

   ```
   +-----------------+----------------------------------------------------+
   | Field           | Value                                              |
   +-----------------+----------------------------------------------------+
   | admin_state_up  | UP                                                 |
   | created_at      | 2021-10-20T02:05:17Z                               |
   | description     |                                                    |
   | id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88               |
   | name            | parent-trunk                                       |
   | port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39               |
   | revision_number | 2                                                  |
   | status          | DOWN                                               |
   | sub_ports       | port_id='479d742e-dd00-4c24-8dd6-b7297fab3ee9', segm |
   |                 | entation_id='55', segmentation_type='vlan'         |
   | tenant_id       | 745d33000ac74d30a77539f8920555e7                   |
   | updated_at      | 2021-08-20T02:10:06Z                               |
   +-----------------+----------------------------------------------------+
   ```

2. Create an instance that uses the parent **port-id** as its vNIC.

   #### Example

```
openstack server create --image cirros --flavor m1.tiny --security-group default --key-name
sshaccess --nic port-id=20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 testInstance
```

Sample output

```
+--------------------------------------+---------------------------------+
| Property                             | Value                           |
+--------------------------------------+---------------------------------+
| OS-DCF:diskConfig                    | MANUAL                          |
| OS-EXT-AZ:availability_zone          |                                 |
| OS-EXT-SRV-ATTR:host                 | -                               |
| OS-EXT-SRV-ATTR:hostname             | testinstance                    |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                               |
| OS-EXT-SRV-ATTR:instance_name        |                                 |
| OS-EXT-SRV-ATTR:kernel_id            |                                 |
| OS-EXT-SRV-ATTR:launch_index         | 0                               |
| OS-EXT-SRV-ATTR:ramdisk_id           |                                 |
| OS-EXT-SRV-ATTR:reservation_id       | r-juqco0el                      |
| OS-EXT-SRV-ATTR:root_device_name     | -                               |
| OS-EXT-SRV-ATTR:user_data            | -                               |
| OS-EXT-STS:power_state               | 0                               |
| OS-EXT-STS:task_state                | scheduling                      |
| OS-EXT-STS:vm_state                  | building                        |
| OS-SRV-USG:launched_at               | -                               |
| OS-SRV-USG:terminated_at             | -                               |
| accessIPv4                           |                                 |
| accessIPv6                           |                                 |
| adminPass                            | uMyL8PnZRBwQ                    |
| config_drive                         |                                 |
| created                              | 2021-08-20T03:02:51Z            |
| description                          | -                               |
| flavor                               | m1.tiny (1)                     |
| hostId                               |                                 |
| host_status                          |                                 |
| id                                   | 88b7aede-1305-4d91-a180-67e7eac |
|                                      | 8b70d                           |
| image                                | cirros (568372f7-15df-4e61-a05f |
|                                      | -10954f79a3c4)                  |
| key_name                             | sshaccess                       |
| locked                               | False                           |
| metadata                             | {}                              |
| name                                 | testInstance                    |
| os-extended-volumes:volumes_attached | []                              |
| progress                             | 0                               |
| security_groups                      | default                         |
| status                               | BUILD                           |
| tags                                 | []                              |
| tenant_id                            | 745d33000ac74d30a77539f8920555e |
|                                      | 7                               |
| updated                              | 2021-08-20T03:02:51Z            |
| user_id                              | 8c4aea738d774967b4ef388eb41fef5 |
|                                      | e                               |
+--------------------------------------+---------------------------------+
```

Additional resources

- Configuring Networking service RPC timeout

11.7. CONFIGURING NETWORKING SERVICE RPC TIMEOUT

There can be situations when you must modify the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) RPC response timeout. For example, live migrations for Compute nodes that use trunk ports can fail if the timeout value is too low.

The RPC response timeout value can vary between sites and is dependent on the system speed. The general recommendation is to set the value to at least 120 seconds per/100 trunk ports.

If your site uses trunk ports, the best practice is to measure the trunk port bind process time for your RHOSP deployment, and then set the RHOSP Networking service RPC response timeout appropriately. Try to keep the RPC response timeout value low, but also provide enough time for the RHOSP Networking service to receive an RPC response.

By using a manual hieradata override, **rpc_response_timeout**, you can set the RPC response timeout value for the RHOSP Networking service.

Procedure

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   #### Example

   ```
   $ vi /home/stack/templates/my-modules-environment.yaml
   ```

   #### TIP

   The RHOSP Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

2. In the YAML environment file under **ExtraConfig**, set the appropriate value (in seconds) for **rpc_response_timeout**. (The default value is 60 seconds.)

   #### Example

   ```
   parameter_defaults:
     ExtraConfig:
       neutron::rpc_response_timeout: 120
   ```

   #### NOTE

   The RHOSP Orchestration service (heat) updates all RHOSP nodes with the value you set in the custom environment file, however this value only impacts the RHOSP Networking components.

3. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-modules-
environment.yaml
```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

11.8. UNDERSTANDING TRUNK STATES

- **ACTIVE**: The trunk is working as expected and there are no current requests.
- **DOWN**: The virtual and physical resources for the trunk are not in sync. This can be a temporary state during negotiation.
- **BUILD**: There has been a request and the resources are being provisioned. After successful completion the trunk returns to **ACTIVE**.
- **DEGRADED**: The provisioning request did not complete, so the trunk has only been partially provisioned. It is recommended to remove the subports and try again.
- **ERROR**: The provisioning request was unsuccessful. Remove the resource that caused the error to return the trunk to a healthier state. Do not add more subports while in the **ERROR** state, as this can cause more issues.

CHAPTER 12. CONFIGURING RBAC POLICIES

12.1. OVERVIEW OF RBAC POLICIES

Role-based access control (RBAC) policies in OpenStack Networking allow granular control over shared *neutron* networks. OpenStack Networking uses a RBAC table to control sharing of *neutron* networks among projects, allowing an administrator to control which projects are granted permission to attach instances to a network.

As a result, cloud administrators can remove the ability for some projects to create networks and can instead allow them to attach to pre-existing networks that correspond to their project.

12.2. CREATING RBAC POLICIES

This example procedure demonstrates how to use a role-based access control (RBAC) policy to grant a project access to a shared network.

1. View the list of available networks:

```
# openstack network list
+--------------------------------------+-------------+-------------------------------------------------------+
| id                                   | name        | subnets                                               |
+--------------------------------------+-------------+-------------------------------------------------------+
| fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 | web-servers | 20512ffe-ad56-4bb4-b064-
2cb18fecc923 192.168.200.0/24 |
| bcc16b34-e33e-445b-9fde-dd491817a48a | private     | 7fe4a05a-4b81-4a59-8c47-
82c965b0e050 10.0.0.0/24      |
| 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 | public      | 2318dc3b-cff0-43fc-9489-
7d4cf48aaab9 172.24.4.224/28  |
+--------------------------------------+-------------+-------------------------------------------------------+
```

2. View the list of projects:

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+----------------------------------+----------+
```

3. Create a RBAC entry for the **web-servers** network that grants access to the *auditors* project (**4b0b98f8c6c040f38ba4f7146e8680f5**):

```
# openstack network rbac create --type network --target-project
4b0b98f8c6c040f38ba4f7146e8680f5 --action access_as_shared web-servers
Created a new rbac_policy:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| action         | access_as_shared                     |
| id             | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id      | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type    | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| project_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+----------------+--------------------------------------+
```

As a result, users in the *auditors* project can connect instances to the **web-servers** network.

12.3. REVIEWING RBAC POLICIES

1. Run the **openstack network rbac list** command to retrieve the ID of your existing role-based access control (RBAC) policies:

```
# openstack network rbac list
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. Run the **openstack network rbac-show** command to view the details of a specific RBAC entry:

```
# openstack network rbac show 314004d0-2261-4d5e-bda7-0181fcf40709
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| action         | access_as_shared                     |
| id             | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id      | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type    | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| project_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+----------------+--------------------------------------+
```

12.4. DELETING RBAC POLICIES

1. Run the **openstack network rbac list** command to retrieve the ID of your existing role-based access control (RBAC) policies:

```
# openstack network rbac list
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. Run the **openstack network rbac delete** command to delete the RBAC, using the ID of the RBAC that you want to delete:

```
# openstack network rbac delete 314004d0-2261-4d5e-bda7-0181fcf40709
Deleted rbac_policy: 314004d0-2261-4d5e-bda7-0181fcf40709
```

12.5. GRANTING RBAC POLICY ACCESS FOR EXTERNAL NETWORKS

You can grant role-based access control (RBAC) policy access to external networks (networks with gateway interfaces attached) using the **--action access_as_external** parameter.

Complete the steps in the following example procedure to create a RBAC for the web-servers network and grant access to the engineering project (c717f263785d4679b16a122516247deb):

- Create a new RBAC policy using the **--action access_as_external** option:

```
# openstack network rbac create --type network --target-project
c717f263785d4679b16a122516247deb --action access_as_external web-servers
 Created a new rbac_policy:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| action         | access_as_external                   |
| id             | ddef112a-c092-4ac1-8914-c714a3d3ba08 |
| object_id      | 6e437ff0-d20f-4483-b627-c3749399bdca |
| object_type    | network                              |
| target_project | c717f263785d4679b16a122516247deb     |
| project_id     | c717f263785d4679b16a122516247deb     |
+----------------+--------------------------------------+
```

As a result, users in the engineering project are able to view the network or connect instances to it:

```
$ openstack network list
+--------------------------------------+-------------+------------------------------------------------------+
| id                                   | name        | subnets                                              |
+--------------------------------------+-------------+------------------------------------------------------+
| 6e437ff0-d20f-4483-b627-c3749399bdca | web-servers | fa273245-1eff-4830-b40c-
57eaeac9b904 192.168.10.0/24 |
+--------------------------------------+-------------+------------------------------------------------------+
```

CHAPTER 13. CONFIGURING DISTRIBUTED VIRTUAL ROUTING (DVR)

13.1. UNDERSTANDING DISTRIBUTED VIRTUAL ROUTING (DVR)

When you deploy Red Hat OpenStack Platform you can choose between a centralized routing model or DVR.

Each model has advantages and disadvantages. Use this document to carefully plan whether centralized routing or DVR better suits your needs.

New default RHOSP deployments use DVR and the Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN).

DVR is disabled by default in ML2/OVS deployments.

13.1.1. Overview of Layer 3 routing

The Red Hat OpenStack Platform Networking service (neutron) provides routing services for project networks. Without a router, VM instances in a project network can communicate with other instances over a shared L2 broadcast domain. Creating a router and assigning it to a project network allows the instances in that network to communicate with other project networks or upstream (if an external gateway is defined for the router).

13.1.2. Routing flows

Routing services in Red Hat OpenStack Platform (RHOSP) can be categorized into three main flows:

- **East-West routing** - routing of traffic between different networks in the same project. This traffic does not leave the RHOSP deployment. This definition applies to both IPv4 and IPv6 subnets.

- **North-South routing with floating IPs** - Floating IP addressing is a one-to-one network address translation (NAT) that can be modified and that floats between VM instances. While floating IPs are modeled as a one-to-one association between the floating IP and a Networking service (neutron) port, they are implemented by association with a Networking service router that performs the NAT translation. The floating IPs themselves are taken from the uplink network that provides the router with external connectivity. As a result, instances can communicate with external resources (such as endpoints on the internet) or the other way around. Floating IPs are an IPv4 concept and do not apply to IPv6. It is assumed that the IPv6 addressing used by projects uses Global Unicast Addresses (GUAs) with no overlap across the projects, and therefore can be routed without NAT.

- **North-South routing without floating IPs** (also known as *SNAT*) - The Networking service offers a default port address translation (PAT) service for instances that do not have allocated floating IPs. With this service, instances can communicate with external endpoints through the router, but not the other way around. For example, an instance can browse a website on the internet, but a web browser outside cannot browse a website hosted within the instance. SNAT is applied for IPv4 traffic only. In addition, Networking service networks that are assigned GUAs prefixes do not require NAT on the Networking service router external gateway port to access the outside world.

13.1.3. Centralized routing

Originally, the Networking service (neutron) was designed with a centralized routing model where a project’s virtual routers, managed by the neutron L3 agent, are all deployed in a dedicated node or cluster of nodes (referred to as the Network node, or Controller node). This means that each time a routing function is required (east/west, floating IPs or SNAT), traffic would traverse through a dedicated node in the topology. This introduced multiple challenges and resulted in sub-optimal traffic flows. For example:

- Traffic between instances flows through a Controller node - when two instances need to communicate with each other using L3, traffic has to hit the Controller node. Even if the instances are scheduled on the same Compute node, traffic still has to leave the Compute node, flow through the Controller, and route back to the Compute node. This negatively impacts performance.

- Instances with floating IPs receive and send packets through the Controller node - the external network gateway interface is available only at the Controller node, so whether the traffic is originating from an instance, or destined to an instance from the external network, it has to flow through the Controller node. Consequently, in large environments the Controller node is subject to heavy traffic load. This would affect performance and scalability, and also requires careful planning to accommodate enough bandwidth in the external network gateway interface. The same requirement applies for SNAT traffic.

To better scale the L3 agent, the Networking service can use the L3 HA feature, which distributes the virtual routers across multiple nodes. In the event that a Controller node is lost, the HA router will failover to a standby on another node and there will be packet loss until the HA router failover completes.

13.2. DVR OVERVIEW

Distributed Virtual Routing (DVR) offers an alternative routing design. DVR isolates the failure domain of the Controller node and optimizes network traffic by deploying the L3 agent and schedule routers on every Compute node. DVR has these characteristics:

- East-West traffic is routed directly on the Compute nodes in a distributed fashion.

- North-South traffic with floating IP is distributed and routed on the Compute nodes. This requires the external network to be connected to every Compute node.

- North-South traffic without floating IP is not distributed and still requires a dedicated Controller node.

- The L3 agent on the Controller node uses the **dvr_snat** mode so that the node serves only SNAT traffic.

- The neutron metadata agent is distributed and deployed on all Compute nodes. The metadata proxy service is hosted on all the distributed routers.

13.3. DVR KNOWN ISSUES AND CAVEATS

- Support for DVR is limited to the ML2 core plug-in and the Open vSwitch (OVS) mechanism driver or ML2/OVN mechanism driver. Other back ends are not supported.

- On ML2/OVS DVR deployments, network traffic for the Red Hat OpenStack Platform Load-balancing service (octavia) goes through the Controller and network nodes, instead of the compute nodes.

- With an ML2/OVS mechanism driver network back end and DVR, it is possible to create VIPs. However, the IP address assigned to a bound port using **allowed_address_pairs**, should match the virtual port IP address (/32).
  If you use a CIDR format IP address for the bound port **allowed_address_pairs** instead, port forwarding is not configured in the back end, and traffic fails for any IP in the CIDR expecting to reach the bound IP port.

- SNAT (source network address translation) traffic is not distributed, even when DVR is enabled. SNAT does work, but all ingress/egress traffic must traverse through the centralized Controller node.

- In ML2/OVS deployments, IPv6 traffic is not distributed, even when DVR is enabled. All ingress/egress traffic goes through the centralized Controller node. If you use IPv6 routing extensively with ML2/OVS, do not use DVR.
  Note that in ML2/OVN deployments, all east/west traffic is always distributed, and north/south traffic is distributed when DVR is configured.

- In ML2/OVS deployments, DVR is not supported in conjunction with L3 HA. If you use DVR with Red Hat OpenStack Platform 17.1 director, L3 HA is disabled. This means that routers are still scheduled on the Network nodes (and load-shared between the L3 agents), but if one agent fails, all routers hosted by this agent fail as well. This affects only SNAT traffic. The **allow_automatic_l3agent_failover** feature is recommended in such cases, so that if one network node fails, the routers are rescheduled to a different node.

- For ML2/OVS environments, the DHCP server is not distributed and is deployed on a Controller node. The ML2/OVS neutron DCHP agent, which manages the DHCP server, is deployed in a highly available configuration on the Controller nodes, regardless of the routing design (centralized or DVR).

- Compute nodes require an interface on the external network attached to an external bridge. They use this interface to attach to a VLAN or flat network for an external router gateway, to host floating IPs, and to perform SNAT for VMs that use floating IPs.

- In ML2/OVS deployments, each Compute node requires one additional IP address. This is due to the implementation of the external gateway port and the floating IP network namespace.

- VLAN, GRE, and VXLAN are all supported for project data separation. When you use GRE or VXLAN, you must enable the L2 Population feature. The Red Hat OpenStack Platform director enforces L2 Population during installation.

13.4. SUPPORTED ROUTING ARCHITECTURES

Red Hat OpenStack Platform (RHOSP) supports both centralized, high-availability (HA) routing and distributed virtual routing (DVR) in the RHOSP versions listed:

- RHOSP centralized HA routing support began in RHOSP 8.

- RHOSP distributed routing support began in RHOSP 12.

13.5. MIGRATING CENTRALIZED ROUTERS TO DISTRIBUTED ROUTING

This section contains information about upgrading to distributed routing for Red Hat OpenStack Platform deployments that use L3 HA centralized routing.

Procedure

1. Upgrade your deployment and validate that it is working correctly.

2. Run the director stack update to configure DVR.

3. Confirm that routing functions correctly through the existing routers.

4. You cannot transition an L3 HA router to *distributed* directly. Instead, for each router, disable the L3 HA option, and then enable the distributed option:

a. Disable the router:

Example

```
$ openstack router set --disable router1
```

b. Clear high availability:

Example

```
$ openstack router set --no-ha router1
```

c. Configure the router to use DVR:

Example

```
$ openstack router set --distributed router1
```

d. Enable the router:

Example

```
$ openstack router set --enable router1
```

e. Confirm that distributed routing functions correctly.

Additional resources

- Deploying DVR with ML2 OVS

13.6. DEPLOYING ML2/OVN OPENSTACK WITH DISTRIBUTED VIRTUAL ROUTING (DVR) DISABLED

New Red Hat OpenStack Platform (RHOSP) deployments default to the neutron Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN) and DVR.

In a DVR topology, compute nodes with floating IP addresses route traffic between virtual machine instances and the network that provides the router with external connectivity (north-south traffic). Traffic between instances (east-west traffic) is also distributed.

You can optionally deploy with DVR disabled. This disables north-south DVR, requiring north-south traffic to traverse a controller or networker node. East-west routing is always distributed in an an ML2/OVN deployment, even when DVR is disabled.

Prerequisites

- RHOSP 17.1 distribution ready for customization and deployment.

Procedure

1. Create a custom environment file, and add the following configuration:

```
parameter_defaults:
  NeutronEnableDVR: false
```

2. To apply this configuration, deploy the overcloud, adding your custom environment file to the stack along with your other environment files. For example:

```
(undercloud) $ openstack overcloud deploy --templates \
  -e [your environment files]
  -e /home/stack/templates/<custom-environment-file>.yaml
```

13.6.1. Additional resources

- Understanding distributed virtual routing (DVR) in the *Configuring Red Hat OpenStack Platform networking* guide.

CHAPTER 14. PROJECT NETWORKING WITH IPV6

14.1. IPV6 SUBNET OPTIONS

When you create IPv6 subnets in a Red Hat OpenStack Platform (RHOSP) project network you can specify address mode and Router Advertisement mode to obtain a particular result as described in the following table.

NOTE

RHOSP does not support IPv6 prefix delegation from an external entity in ML2/OVN deployments. You must obtain the Global Unicast Address prefix from your external prefix delegation router and set it by using the **subnet-range** argument during creation of a IPv6 subnet.

For example:

```
openstack subnet create
--subnet-range 2002:c000:200::64
--no-dhcp
--gateway 2002:c000:2fe::
--dns-nameserver 2002:c000:2fe::
--network provider
provider-subnet-2002:c000:200::
```

RA Mode	Address Mode	Result

RA Mode	Address Mode	Result
ipv6_ra_mode=not set	ipv6-address-mode=slaac	The instance receives an IPv6 address from the external router (not managed by OpenStack Networking) using Stateless Address Autoconfiguration (SLAAC). **NOTE** OpenStack Networking supports only EUI-64 IPv6 address assignment for SLAAC. This allows for simplified IPv6 networking, as hosts self-assign addresses based on the base 64-bits plus the MAC address. You cannot create subnets with a different netmask and *address_assign_type* of SLAAC.

RA Mode	Address Mode	Result
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	The instance receives an IPv6 address and optional information from OpenStack Networking (dnsmasq) using **DHCPv6 stateful**.
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateless	The instance receives an IPv6 address from the external router using SLAAC, and optional information from OpenStack Networking (dnsmasq) using **DHCPv6 stateless**.
ipv6_ra_mode=slaac	ipv6-address-mode=not-set	The instance uses SLAAC to receive an IPv6 address from OpenStack Networking (**radvd**).
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=not-set	The instance receives an IPv6 address and optional information from an external DHCPv6 server using **DHCPv6 stateful**.
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=not-set	The instance receives an IPv6 address from OpenStack Networking (**radvd**) using SLAAC, and optional information from an external DHCPv6 server using **DHCPv6 stateless**.
ipv6_ra_mode=slaac	ipv6-address-mode=slaac	The instance receives an IPv6 address from OpenStack Networking (**radvd**) using **SLAAC**.
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=dhcpv6-stateful	The instance receives an IPv6 address from OpenStack Networking (**dnsmasq**) using **DHCPv6 stateful**, and optional information from OpenStack Networking (**dnsmasq**) using **DHCPv6 stateful**.
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	The instance receives an IPv6 address from OpenStack Networking (**radvd**) using **SLAAC**, and optional information from OpenStack Networking (**dnsmasq**) using **DHCPv6 stateless**.

14.2. CREATE AN IPV6 SUBNET USING STATEFUL DHCPV6

You can create an IPv6 subnet in a Red Hat OpenStack (RHOSP) project network.

For example, you can create an IPv6 subnet using Stateful DHCPv6 in network named database-servers in a project named QA.

Procedure

1. Retrieve the project ID of the Project where you want to create the IPv6 subnet. These values are unique between OpenStack deployments, so your values differ from the values in this example.

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 25837c567ed5458fbb441d39862e1399 |    QA    |
| f59f631a77264a8eb0defc898cb836af |  admin   |
| 4e2e1951e70643b5af7ed52f3ff36539 |   demo   |
| 8561dff8310e4cd8be4b6fd03dc8acf5 | services |
+----------------------------------+----------+
```

2. Retrieve a list of all networks present in OpenStack Networking (neutron), and note the name of the network where you want to host the IPv6 subnet:

```
# openstack network list
+--------------------------------------+------------------+-------------------------------------------------------------+
| id                                   | name             | subnets                                                     |
+--------------------------------------+------------------+-------------------------------------------------------------+
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-db41-4538-af40-48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-6019-4e79-a21c-1942a460b920 172.24.4.224/28        |
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers |                                                             |
+--------------------------------------+------------------+-------------------------------------------------------------+
```

3. Include the project ID, network name, and ipv6 address mode in the **openstack subnet create** command:

```
# openstack subnet create --ip-version 6 --ipv6-address-mode dhcpv6-stateful --project
25837c567ed5458fbb441d39862e1399 --network database-servers --subnet-range
fdf8:f53b:82e4::53/125 subnet_name

Created a new subnet:
+-------------------+--------------------------------------------------------------+
| Field             | Value                                                        |
+-------------------+--------------------------------------------------------------+
| allocation_pools  | {"start": "fdf8:f53b:82e4::52", "end": "fdf8:f53b:82e4::56"} |
| cidr              | fdf8:f53b:82e4::53/125                                       |
| dns_nameservers   |                                                              |
| enable_dhcp       | True                                 |
| gateway_ip        | fdf8:f53b:82e4::51                   |
| host_routes       |                                      |
| id                | cdfc3398-997b-46eb-9db1-ebbd88f7de05 |
| ip_version        | 6                                    |
| ipv6_address_mode | dhcpv6-stateful                      |
| ipv6_ra_mode      |                                      |
| name              |                                      |
| network_id        | 6aff6826-4278-4a35-b74d-b0ca0cbba340 |
| tenant_id         | 25837c567ed5458fbb441d39862e1399     |
+-------------------+--------------------------------------+
```

Validation steps

1. Validate this configuration by reviewing the network list. Note that the entry for *database-servers* now reflects the newly created IPv6 subnet:

```
# openstack network list
+--------------------------------------+------------------+--------------------------------------------------
---------+
| id                                   | name             | subnets                                         |
+--------------------------------------+------------------+--------------------------------------------------
---------+
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers | cdfc3398-997b-46eb-9db1-
ebbd88f7de05 fdf8:f53b:82e4::50/125 |
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-db41-4538-af40-
48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-6019-4e79-a21c-
1942a460b920 172.24.4.224/28        |
+--------------------------------------+------------------+--------------------------------------------------
---------+
```

Result

As a result of this configuration, instances that the QA project creates can receive a DHCP IPv6 address when added to the database-servers subnet:

```
# openstack server list
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
| ID                                   | Name       | Status | Task State | Power State | Networks
|
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
| fad04b7a-75b5-4f96-aed9-b40654b56e03 | corp-vm-01 | ACTIVE | -          | Running     |
database-servers=fdf8:f53b:82e4::52 |
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
```

Additional resources

To find the Router Advertisement mode and address mode combinations to achieve a particular result in an IPv6 subnet, see IPv6 subnet options in the Configuring Red Hat OpenStack Platform networking .

CHAPTER 15. MANAGING PROJECT QUOTAS

15.1. CONFIGURING PROJECT QUOTAS

OpenStack Networking (neutron) supports the use of quotas to constrain the number of resources created by tenants/projects.

Procedure

- You can set project quotas for various network components in the /var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf file.
  For example, to limit the number of routers that a project can create, change the **quota_router** value:

  ```
  quota_router = 10
  ```

  In this example, each project is limited to a maximum of 10 routers.

For a listing of the quota settings, see sections that immediately follow.

15.2. L3 QUOTA OPTIONS

Here are quota options available for layer 3 (L3) networking:

- **quota_floatingip** - The number of floating IPs available to a project.
- **quota_network** - The number of networks available to a project.
- **quota_port** - The number of ports available to a project.
- **quota_router** - The number of routers available to a project.
- **quota_subnet** - The number of subnets available to a project.
- **quota_vip** - The number of virtual IP addresses available to a project.

15.3. FIREWALL QUOTA OPTIONS

Here are quota options available for managing firewalls for projects:

- **quota_firewall** - The number of firewalls available to a project.
- **quota_firewall_policy** - The number of firewall policies available to a project.
- **quota_firewall_rule** - The number of firewall rules available to a project.

15.4. SECURITY GROUP QUOTA OPTIONS

The Networking service quota engine manages security groups and security group rules, and it is not possible to set all quotas to zero before creating the default security group (and the two default security group rules that accepts all egress traffic for IPv4 and IPv6). When you create a new project, the Networking service does not create the default security group until a network or a port is created, or until you list the security group or the security group rules.

Here are quota options available for managing the number of security groups that projects can create:

- **quota_security_group** - The number of security groups available to a project.
- **quota_security_group_rule** - The number of security group rules available to a project.

15.5. MANAGEMENT QUOTA OPTIONS

Here are additional options available to administrators for managing quotas for projects:

- **default_quota*** - The default number of resources available to a project.
- **quota_health_monitor*** - The number of health monitors available to a project.
  Health monitors do not consume resources, however the quota option is available because OpenStack Networking considers health monitors as resource consumers.
- **quota_member** - The number of pool members available to a project.
  Pool members do not consume resources, however the quota option is available because OpenStack Networking considers pool members as resource consumers.
- **quota_pool** - The number of pools available to a project.

CHAPTER 16. DEPLOYING ROUTED PROVIDER NETWORKS

16.1. ADVANTAGES OF ROUTED PROVIDER NETWORKS

In Red Hat OpenStack Platform (RHOSP), administrators can create routed provider networks. Routed provider networks are typically used in edge deployments, and rely on multiple layer 2 network segments instead of traditional networks that have only one segment.

Routed provider networks simplify the cloud for end users because they see only one network. For administrators, routed provider networks deliver scalabilty and fault tolerance. For example, if a major error occurs, only one segment is impacted instead of the entire network failing.

Before routed provider networks, administrators typically had to choose from one of the following architectures:

- A single, large layer 2 network
- Multiple, smaller layer 2 networks

Single, large layer 2 networks become complex when scaling and reduce fault tolerance (increase failure domains).

Multiple, smaller layer 2 networks scale better and shrink failure domains, but can introduce complexity for end users.

Starting with RHOSP 16.2 and later, you can deploy routed provider networks using the Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN). (Routed provider network support for the ML2/Open vSwitch (OVS) and SR-IOV mechanism drivers was introduced in RHOSP 16.1.1.)

Additional resources

- Section 16.2, “Fundamentals of routed provider networks”

16.2. FUNDAMENTALS OF ROUTED PROVIDER NETWORKS

A routed provider network is different from other types of networks because of the one-to-one association between a network subnet and a segment. In the past, the Red Hat OpenStack (RHOSP) Networking service has not supported routed provider networks, because the Networking service required that all subnets must either belong to the same segment or to no segment.

With routed provider networks, the IP addresses available to virtual machine (VM) instances depend on the segment of the network available on the particular compute node. The Networking service port can be associated with only one network segment.

Similar to conventional networking, layer 2 (switching) handles transit of traffic between ports on the same network segment and layer 3 (routing) handles transit of traffic between segments.

The Networking service does not provide layer 3 services between segments. Instead, it relies on physical network infrastructure to route subnets. Thus, both the Networking service and physical network infrastructure must contain configuration for routed provider networks, similar to conventional provider networks.

You can configure the Compute scheduler to filter Compute nodes that have affinity with routed network segments, so that the scheduler places instances only on Compute nodes that are in the required routed provider network segment.

If you require a DHCP-metadata service, you must define an availability zone for each edge site or network segment, to ensure that the local DHCP agent is deployed.

Additional resources

- Section 16.1, “Advantages of routed provider networks”

16.3. LIMITATIONS OF ROUTED PROVIDER NETWORKS

The known constraints of routed provider networks in Red Hat OpenStack Platform include:

- North-south routing with central SNAT or a floating IP is not supported.
- When using SR-IOV or PCI pass-through, physical network (physnet) names must be the same in central and remote sites or segments. You cannot reuse segment IDs.

16.4. PREPARING FOR A ROUTED PROVIDER NETWORK

To create a routed provider network in Red Hat OpenStack Platform (RHOSP), you must first gather the network information that is required to create it. You must configure the overcloud to create a custom role that deploys a RHOSP Networking service (neutron) metadata agent for the Compute nodes that contain the network segments. For environments that use the ML2/OVS mechanism driver, in addition to the metadata agent, you must also include the **NeutronDhcpAgent** service on the Compute nodes. On the Controllers that are running the Compute scheduler services, you must enable scheduling support for routed provider networks.

Prerequisites

- You must be a RHOSP user with the **admin** role.

Procedure

1. Gather the VLAN IDs from the **tripleo-heat-templates/network_data.yaml** file for the network you want to create the routed provider network on, and assign unique physical network names for each segment that you will create on the routed provider network. This enables reuse of the same segmentation details between subnets.
   Create a reference table to visualize the relationships between the VLAN IDs, segments, and physical network names:

   Table 16.1. Example - routed provider network segment definitions

Routed provider network	VLAN ID	Segment	Physical network
multisegment1	128	segment1	provider1
multisegment1	129	segment2	provider2

2. Plan the routing between segments.

Each subnet on a segment must contain the gateway address of the router interface on that particular subnet. You need the subnet address in both IPv4 and IPv6 formats.

Table 16.2. Example - routing plan for routed provider network segments

Routed provider network	Segment	Subnet address	Gateway address
multisegment1	segment1 (IPv4)	203.0.113.0/24	203.0.113.1
multisegment1	segment1 (IPv6)	fd00:203:0:113::/64	fd00:203:0:113::1
multisegment1	segment2 (IPv4)	198.51.100.0/24	198.51.100.1
multisegment1	segment2 (IPv6)	fd00:198:51:100::/64	fd00:198:51:100::1

3. Routed provider networks require that Compute nodes reside on different segments. Check the **templates/overcloud-baremetal-deployed.yaml** file to ensure that every Compute host in a routed provider network has direct connectivity to one of its segments.
   For more information, see Provisioning bare metal nodes for the overcloud in the *Installing and managing Red Hat OpenStack Platform with director* guide.

4. Ensure that the **NeutronMetadataAgent** service is included in **templates/roles_data-custom.yaml** for the Compute nodes containing the segments:

   ```
   ...
   - name: Compute
     ...
     ServicesDefault:
       - OS::TripleO::Services::NeutronMetadataAgent
   ...
   ```

   For more information, see Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide.

5. When using the ML2/OVS mechanism driver, in addition to the **NeutronMetadataAgent** service, also ensure that the **NeutronDhcpAgent** service is included in **templates/roles_data-custom.yaml** for the Compute nodes containing the segments:

   ```
   ...
   - name: Compute
     ...
     ServicesDefault:
       - OS::TripleO::Services::NeutronDhcpAgent
       - OS::TripleO::Services::NeutronMetadataAgent
   ...
   ```

   ### TIP

   Unlike conventional provider networks, a DHCP agent cannot support more than one segment within a network. Deploy DHCP agents on the Compute nodes containing the segments rather than on the network nodes to reduce the node count.

6. Create an routed provider network environment file, for example, **rpn_env.yaml**.

7. Configure DHCP to enable metadata support on isolated networks:

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
```

8. Ensure that the **segments** service plug-in is loaded into the Networking service:

```
$ openstack extension list --network --max-width 80 | grep -E "Segment"
```

   If the **segments** plug-in is missing, add it to the **NeutronServicePlugins** parameter:

   Example

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
  NeutronServicePlugins: 'router,qos,segments,trunk,placement'
```

   IMPORTANT

   When you add new values to the **NeutronServicePlugins** parameter, RHOSP director overwrites any previously declared values with the ones that you are adding. Therefore, when you are adding **segments**, you must also include any previously declared Networking service plug-ins.

9. To verify the network with the Placement service before scheduling an instance on a host, enable scheduling support for routed provider networks on the Controllers that are running the Compute scheduler services.

   Example

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
  NeutronServicePlugins: 'router,qos,segments,trunk,placement'
  NovaSchedulerQueryPlacementForRoutedNetworkAggregates: true
```

10. Add your routed provider network environment file to the stack with your other environment files and deploy the overcloud:

```
$ openstack overcloud deploy --templates \
 -e <your_environment_files> \
 -e /home/stack/templates/rpn_env.yaml
```

Next steps

- Creating a routed provider network

Additional resources

- Provisioning bare metal nodes for the overcloud in the *Installing and managing Red Hat OpenStack Platform with director* guide

- Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide

16.5. CREATING A ROUTED PROVIDER NETWORK

Routed provider networks simplify the Red Hat OpenStack Platform (RHOSP) cloud for end users because they see only one network. For administrators, routed provider networks deliver scalabilty and fault tolerance.

When you perform this procedure, you create an routed provider network with two network segments. Each segment contains one IPv4 subnet and one IPv6 subnet.

Prerequisites

- Complete the steps in Section 16.4, "Preparing for a routed provider network" .

- You must be a RHOSP user with the **admin** role.

Procedure

1. Create a VLAN provider network that includes a default segment.
   In this example, the VLAN provider network is named **multisegment1** and uses a physical network called **provider1** and a VLAN whose ID is **128**:

   #### Example

   ```
   $ openstack network create --share --provider-physical-network provider1 \
     --provider-network-type vlan --provider-segment 128 multisegment1
   ```

   #### Sample output

   ```
   +---------------------------+--------------------------------------+
   | Field                     | Value                                |
   +---------------------------+--------------------------------------+
   | admin_state_up            | UP                                   |
   | id                        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
   | ipv4_address_scope        | None                                 |
   | ipv6_address_scope        | None                                 |
   | l2_adjacency              | True                                 |
   | mtu                       | 1500                                 |
   | name                      | multisegment1                        |
   | port_security_enabled     | True                                 |
   | provider:network_type     | vlan                                 |
   | provider:physical_network | provider1                            |
   | provider:segmentation_id  | 128                                  |
   | revision_number           | 1                                    |
   | router:external           | Internal                             |
   | shared                    | True                                 |
   | status                    | ACTIVE                               |
   | subnets                   |                                      |
   | tags                      | []                                   |
   +---------------------------+--------------------------------------+
   ```

2. Rename the default network segment to **segment1**.

a. Obtain the segment ID:

```
$ openstack network segment list --network multisegment1
```

Sample output

```
+--------------------------------------+----------+--------------------------------------+--------------
+---------+
| ID                                   | Name     | Network                              | Network Type |
Segment |
+--------------------------------------+----------+--------------------------------------+--------------
+---------+
| 43e16869-ad31-48e4-87ce-acf756709e18 | None     | 6ab19caa-dda9-4b3d-abc4-
5b8f435b98d9 | vlan         | 128     |
+--------------------------------------+----------+--------------------------------------+--------------
+---------+
```

b. Using the segment ID, rename the network segment to **segment1**:

```
$ openstack network segment set --name segment1 43e16869-ad31-48e4-87ce-
acf756709e18
```

3. Create a second segment on the provider network.
In this example, the network segment uses a physical network called **provider2** and a VLAN whose ID is **129**:

Example

```
$ openstack network segment create --physical-network provider2 \
  --network-type vlan --segment 129 --network multisegment1 segment2
```

Sample output

```
+------------------+--------------------------------------+
| Field            | Value                                |
+------------------+--------------------------------------+
| description      | None                                 |
| headers          |                                      |
| id               | 053b7925-9a89-4489-9992-e164c8cc8763 |
| name             | segment2                             |
| network_id       | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| network_type     | vlan                                 |
| physical_network | provider2                            |
| revision_number  | 1                                    |
| segmentation_id  | 129                                  |
| tags             | []                                   |
+------------------+--------------------------------------+
```

4. Verify that the network contains the **segment1** and **segment2** segments:

```
$ openstack network segment list --network multisegment1
```

Sample output

```
+--------------------------------------+----------+--------------------------------------+--------------+-----
----+
| ID                                   | Name     | Network                              | Network Type | Segment |
+--------------------------------------+----------+--------------------------------------+--------------+-----
----+
| 053b7925-9a89-4489-9992-e164c8cc8763 | segment2 | 6ab19caa-dda9-4b3d-abc4-
5b8f435b98d9 | vlan         | 129     |
| 43e16869-ad31-48e4-87ce-acf756709e18 | segment1 | 6ab19caa-dda9-4b3d-abc4-
5b8f435b98d9 | vlan         | 128     |
+--------------------------------------+----------+--------------------------------------+--------------+-----
----+
```

5. Create one IPv4 subnet and one IPv6 subnet on the **segment1** segment.
   In this example, the IPv4 subnet uses **203.0.113.0/24**:

Example

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 4 --subnet-range 203.0.113.0/24 \
  multisegment1-segment1-v4
```

Sample output

```
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.2-203.0.113.254            |
| cidr              | 203.0.113.0/24                       |
| enable_dhcp       | True                                 |
| gateway_ip        | 203.0.113.1                          |
| id                | c428797a-6f8e-4cb1-b394-c404318a2762 |
| ip_version        | 4                                    |
| name              | multisegment1-segment1-v4            |
| network_id        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number   | 1                                    |
| segment_id        | 43e16869-ad31-48e4-87ce-acf756709e18 |
| tags              | []                                   |
+-------------------+--------------------------------------+
```

In this example, the IPv6 subnet uses **fd00:203:0:113::/64**:

Example

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 6 --subnet-range fd00:203:0:113::/64 \
  --ipv6-address-mode slaac multisegment1-segment1-v6
```

Sample output

```
+-------------------+------------------------------------------------------+
| Field             | Value                                                |
+-------------------+-------------------------------------------------------+
| allocation_pools  | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr              | fd00:203:0:113::/64                                   |
| enable_dhcp       | True                                                  |
| gateway_ip        | fd00:203:0:113::1                                     |
| id                | e41cb069-9902-4c01-9e1c-268c8252256a                  |
| ip_version        | 6                                                     |
| ipv6_address_mode | slaac                                                 |
| ipv6_ra_mode      | None                                                  |
| name              | multisegment1-segment1-v6                             |
| network_id        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9                  |
| revision_number   | 1                                                     |
| segment_id        | 43e16869-ad31-48e4-87ce-acf756709e18                  |
| tags              | []                                                    |
+-------------------+-------------------------------------------------------+
```

NOTE

By default, IPv6 subnets on provider networks rely on physical network infrastructure for stateless address autoconfiguration (SLAAC) and router advertisement.

6. Create one IPv4 subnet and one IPv6 subnet on the **segment2** segment.
   In this example, the IPv4 subnet uses **198.51.100.0/24**:

   #### Example

   ```
   $ openstack subnet create \
     --network multisegment1 --network-segment segment2 \
     --ip-version 4 --subnet-range 198.51.100.0/24 \
     multisegment1-segment2-v4
   ```

   #### Sample output

   ```
   +-------------------+--------------------------------------+
   | Field             | Value                                |
   +-------------------+--------------------------------------+
   | allocation_pools  | 198.51.100.2-198.51.100.254          |
   | cidr              | 198.51.100.0/24                      |
   | enable_dhcp       | True                                 |
   | gateway_ip        | 198.51.100.1                         |
   | id                | 242755c2-f5fd-4e7d-bd7a-342ca95e50b2 |
   | ip_version        | 4                                    |
   | name              | multisegment1-segment2-v4            |
   | network_id        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
   | revision_number   | 1                                    |
   | segment_id        | 053b7925-9a89-4489-9992-e164c8cc8763 |
   | tags              | []                                   |
   +-------------------+--------------------------------------+
   ```

   In this example, the IPv6 subnet uses **fd00:198:51:100::/64**:

   #### Example

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 6 --subnet-range fd00:198:51:100::/64 \
  --ipv6-address-mode slaac multisegment1-segment2-v6
```

Sample output

```
+-------------------+------------------------------------------------------+
| Field             | Value                                                |
+-------------------+------------------------------------------------------+
| allocation_pools  | fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff |
| cidr              | fd00:198:51:100::/64                                 |
| enable_dhcp       | True                                                 |
| gateway_ip        | fd00:198:51:100::1                                   |
| id                | b884c40e-9cfe-4d1b-a085-0a15488e9441                 |
| ip_version        | 6                                                    |
| ipv6_address_mode | slaac                                                |
| ipv6_ra_mode      | None                                                 |
| name              | multisegment1-segment2-v6                            |
| network_id        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9                 |
| revision_number   | 1                                                    |
| segment_id        | 053b7925-9a89-4489-9992-e164c8cc8763                 |
| tags              | []                                                   |
+-------------------+------------------------------------------------------+
```

Verification

1. Verify that each IPv4 subnet associates with at least one DHCP agent:

```
$ openstack network agent list --agent-type dhcp --network multisegment1
```

Sample output

```
+--------------------------------------+------------+-------------+-------------------+-------+-------+------
--------------+
| ID                                   | Agent Type | Host        | Availability Zone | Alive | State | Binary
|
+--------------------------------------+------------+-------------+-------------------+-------+-------+------
--------------+
| c904ed10-922c-4c1a-84fd-d928abaf8f55 | DHCP agent | compute0001 | nova              | :-)
| UP    | neutron-dhcp-agent |
| e0b22cc0-d2a6-4f1c-b17c-27558e20b454 | DHCP agent | compute0101 | nova              | :-)
| UP    | neutron-dhcp-agent |
+--------------------------------------+------------+-------------+-------------------+-------+-------+------
--------------+
```

2. Verify that inventories were created for each segment IPv4 subnet in the Compute service placement API.
   Run this command for all segment IDs:

```
$ SEGMENT_ID=053b7925-9a89-4489-9992-e164c8cc8763
$ openstack resource provider inventory list $SEGMENT_ID
```

Sample output

In this sample output, only one of the segments is shown:

```
+----------------+------------------+----------+----------+-----------+----------+-------+
| resource_class | allocation_ratio | max_unit | reserved | step_size | min_unit | total |
+----------------+------------------+----------+----------+-----------+----------+-------+
| IPV4_ADDRESS   |              1.0 |        1 |        2 |         1 |        1 |    30 |
+----------------+------------------+----------+----------+-----------+----------+-------+
```

3. Verify that host aggregates were created for each segment in the Compute service:

```
$ openstack aggregate list
```

Sample output

In this example, only one of the segments is shown:

```
+----+---------------------------------------------------------+-------------------+
| Id | Name                                                    | Availability Zone |
+----+---------------------------------------------------------+-------------------+
| 10 | Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763 | None              |
+----+---------------------------------------------------------+-------------------+
```

4. Launch one or more instances. Each instance obtains IP addresses according to the segment it uses on the particular compute node.

NOTE

If a fixed IP is specified by the user in the port create request, that particular IP is allocated immediately to the port. However, creating a port and passing it to an instance yields a different behavior than conventional networks. If the fixed IP is not specified on the port create request, the Networking service defers assignment of IP addresses to the port until the particular compute node becomes apparent. For example, when you run this command:

```
$ openstack port create --network multisegment1 port1
```

Sample output

```
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | UP                                   |
| binding_vnic_type     | normal                               |
| id                    | 6181fb47-7a74-4add-9b6b-f9837c1c90c4 |
| ip_allocation         | deferred                             |
| mac_address           | fa:16:3e:34:de:9b                    |
| name                  | port1                                |
| network_id            | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| port_security_enabled | True                                 |
| revision_number       | 1                                    |
| security_groups       | e4fcef0d-e2c5-40c3-a385-9c33ac9289c5 |
| status                | DOWN                                 |
| tags                  | []                                   |
+-----------------------+--------------------------------------+
```

Additional resources

- Section 16.4, “Preparing for a routed provider network”
- network create in the *Command line interface reference*
- network segment create in the *Command line interface reference*
- subnet create in the *Command line interface reference*
- port create in the *Command line interface reference*

16.6. MIGRATING A NON-ROUTED NETWORK TO A ROUTED PROVIDER NETWORK

You can migrate a non-routed network to a routed provider network by associating the subnet of the network with the ID of the network segment.

Prerequisites

- The non-routed network you are migrating must contain *only* one segment and *only* one subnet.

IMPORTANT

In non-routed provider networks that contain multiple subnets or network segments it is not possible to safely migrate to an routed provider network. In non-routed networks, addresses from the subnet allocation pools are assigned to ports without consideration of the network segment to which the port is bound.

Procedure

1. For the network that is being migrated, obtain the ID of the current network segment.

   Example

   ```
   $ openstack network segment list --network my_network
   ```

   Sample output

   ```
   +--------------------------------------+------+--------------------------------------+--------------+---------
   +
   | ID                                   | Name | Network                              | Network Type | Segment |
   +--------------------------------------+------+--------------------------------------+--------------+---------
   +
   | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 | None | 45e84575-2918-471c-95c0-
   018b961a2984 | flat         | None    |
   +--------------------------------------+------+--------------------------------------+--------------+---------
   +
   ```

2. For the network that is being migrated, obtain the ID of the current subnet.

   Example

   ```
   $ openstack network segment list --network my_network
   ```

   Sample output

   ```
   +--------------------------------------+-----------+--------------------------------------+---------------+
   | ID                                   | Name      | Network                              | Subnet        |
   +--------------------------------------+-----------+--------------------------------------+---------------+
   | 71d931d2-0328-46ae-93bc-126caf794307 | my_subnet | 45e84575-2918-471c-95c0-
   018b961a2984 | 172.24.4.0/24 |
   +--------------------------------------+-----------+--------------------------------------+---------------+
   ```

3. Verify that the current **segment_id** of the subnet has a value of **None**.

   Example

   ```
   $ openstack subnet show my_subnet --c segment_id
   ```

   Sample output

   ```
   +------------+-------+
   | Field      | Value |
   +------------+-------+
   ```

```
| segment_id | None  |
+------------+-------+
```

4. Change the value of the subnet **segment_id** to the network segment ID.
   Here is an example:

```
$ openstack subnet set --network-segment 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7
my_subnet
```

Verification

- Verify that the subnet is now associated with the desired network segment.

  Example

```
$ openstack subnet show my_subnet --c segment_id
```

  Sample output

```
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| segment_id | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 |
+------------+--------------------------------------+
```

Additional resources

- subnet show in the *Command line interface reference*
- subnet set in the *Command line interface reference*

CHAPTER 17. CREATING CUSTOM VIRTUAL ROUTERS WITH ROUTER FLAVORS

IMPORTANT

The content in this section 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, see Technology Preview.

You can use router flavors to deploy custom virtual routers in your Red Hat OpenStack Platform (RHOSP) ML2/OVN environments. After the RHOSP administrator enables the router flavor feature and creates the router flavor, users can create custom routers by using the router flavor.

Within a RHOSP deployment you can combine virtual custom routers that are based on router flavors with routers of the default OVN type.

Using router flavors does not affect the operation of the default OVN router. When router flavors are used, the default OVN router is treated as the default router flavor, with no impact on its configuration or operation.

To set up router flavors and create custom routers, perform the following general steps:

1. The administrator loads the necessary RHOSP Networking service (neutron) plug-in and specifies the service provider.
   See Section 17.1, "Enabling router flavors and creating service providers" .

2. The administrator creates the router flavor.
   See Section 17.2, "Creating a router flavor" .

3. The user creates a custom router by using one of the router flavors.
   See Section 17.3, "Creating a custom virtual router" .

17.1. ENABLING ROUTER FLAVORS AND CREATING SERVICE PROVIDERS

Before a Red Hat OpenStack Platform (RHOSP) administrator can create a router flavor, the administrator must first load the necessary RHOSP Networking service (neutron) plug-in and specify the service provider.

The administrator must deploy the service provider code in a module in the Networking service directories. Red Hat recommends the **neutron.services.ovn_l3.service_providers.user_defined** module.

You can find a sample service provider named **UserDefined** in the **neutron.services.ovn_l3.service_providers.user_defined** module.

NOTE

The following procedure involves direct editing of **.conf** files on the Controller nodes. Red Hat is developing heat template methods and OpenStack commands to replace this direct editing method.

Prerequisites

- Your Networking service mechanism driver must be ML2/OVN.
- You have a router flavor service provider created for your deployment.
- You have access to the RHOSP Controller nodes and permission to update configuration files.

Procedure

1. On one of the Controller nodes, open the file, **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf**.

2. In the **service_plugins** list, change **ovn-routers** to **ovn-router-flavors-ha**:

   ```
   [DEFAULT]
   service_plugins = qos,ovn-router-flavors-ha,trunk,segments,port_forwarding,log
   ```

3. Create a **service_providers** section and add a service provider definition for each router flavor that you plan to use.

   #### Example

   In this example, a service provider, **user_defined_1**, is added:

   ```
   ...
   [service_providers]
   service_provider =
   L3_ROUTER_NAT:user_defined_1:neutron.services.ovn_l3.service_providers.user_defined.Us
   erDefined1
   ```

   A router flavor service provider definition has the following elements:

   Service provider constant

   L3_ROUTER_NAT

   Name

   Name of the service provider, which is a descriptive string between two colon characters.
   For example, **:user_defined_1:**. The name must be unique within the environment.

   Path

   Red Hat recommends using this path:
   neutron.services.ovn_l3.service_providers.user_defined

   Class

   A python class name for the service provider.
   Each provider has its own class. For example, **UserDefined1**.

   > **NOTE**
   >
   > Retain this class name and its path. You need it later when you create the router flavor.

4. Restart the Networking service (neutron):

```
$ sudo podman restart neutron_api
```

5. Perform steps 1 - 4 on the remaining RHOSP Controller nodes.

Verification

- Verify that the Networking service has loaded your user defined service provider:

  ```
  $ openstack network service provider list
  ```

  If the procedure was successful the new service appears in the list.

  #### Sample output

  ```
  +-------------------------+-------+---------+
  | Service Type  | Name           | Default |
  +---------------+----------------+---------+
  | L3_ROUTER_NAT | user_defined_1  | False   |
  | L3_ROUTER_NAT | ovn            | True    |
  +---------------+----------------+---------+
  ```

17.2. CREATING A ROUTER FLAVOR

The Red Hat OpenStack Platform (RHOSP) administrator can create router flavors that users must specify when they create custom virtual routers in their RHOSP ML2/OVN environment. After the administrator has loaded the Networking service (neutron) **ovn-router-flavors-ha** plug-in and specified the service provider, the remaining steps for creating a router flavor are:

1. Create a service profile for the router flavor.
2. Create the router flavor.
3. Add the service profile to the router flavor.

Prerequisites

- Your Networking service mechanism driver must be ML2/OVN.
- You must be a RHOSP user with the **admin** role.
- The Networking service has the **ovn-router-flavors-ha** plug-in loaded.
- The router flavor service provider has been created and you know the name and path of its class.
  For more information, see Section 17.1, “Enabling router flavors and creating service providers” .

Procedure

1. Source your overcloud credentials file that assigns you the **admin** role.
2. Using the service provider class and its path, create a service profile for the router flavor.
   Retain the profile ID, as you need it in a later step.

   #### Example

In this example, the driver class name is **UserDefined1**, and its path is, **neutron.services.ovn_l3.service_providers.user_defined**:

```
$ openstack network flavor profile create \
--description "User-defined router flavor profile" \
--enable --driver \
neutron.services.ovn_l3.service_providers.user_defined.UserDefined1
```

Sample output

```
+-------------+---------------------------------------------------------------------+
| Field       | Value                                                               |
+-------------+---------------------------------------------------------------------+
| description | User-defined router flavor profile                                  |
| driver      | neutron.services.ovn_l3.service_providers.user_defined.UserDefined1 |
| enabled     | True                                                                |
| id          | a717c92c-63f7-47e8-9efb-6ad0d61c4875                                |
| meta_info   |                                                                     |
| project_id  | None                                                                |
+-------------+---------------------------------------------------------------------+
```

3. Create the router flavor:

```
$ openstack network flavor create \
--service-type L3_ROUTER_NAT \
--description "User-defined flavor for routers" \
user-defined-router-flavor
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| description         | User-defined flavor for routers      |
| enabled             | True                                 |
| id                  | e47c1c5c-629b-4c48-b49a-78abe6ac7696 |
| name                | user-defined-router-flavor           |
| service_profile_ids | []                                   |
| service_type        | L3_ROUTER_NAT                        |
+---------------------+--------------------------------------+
```

4. Add the service profile to the router flavor, using the profile ID from an earlier step.

Example

```
$ openstack network flavor add profile user-defined-router-flavor \
a717c92c-63f7-47e8-9efb-6ad0d61c4875
```

Additional resources

- Section 17.1, “Enabling router flavors and creating service providers”

17.3. CREATING A CUSTOM VIRTUAL ROUTER

You can create a custom virtual router in your Red Hat OpenStack Platform (RHOSP) environment by using a router flavor provided by your RHOSP administrator.

Prerequisites

- The RHOSP administrator has created a router flavor.
- Your Networking service (neutron) mechanism driver must be ML2/OVN.

Procedure

1. Source your credentials file.
2. Get the ID for the router flavor to use to create your custom router:

   ```
   $ openstack network flavor list -c ID -c Name
   ```

   Sample output

   ```
   +--------------------------------------+-----------------------------+
   | ID                                   | Name                        |
   +--------------------------------------+-----------------------------+
   | 4b37f895-e78e-49df-a96b-1916550f9116 | user-defined-router-flavor  |
   +--------------------------------------+-----------------------------+
   ```

3. Using the router flavor ID, create a custom router:

   Example

   In this example, a custom router, **user-defined-router** is created using the flavor ID for **user-defined-router-flavor**:

   ```
   $ openstack router create \
   --flavor-id 4b37f895-e78e-49df-a96b-1916550f9116 user-defined-router
   ```

   If you do not use the **--flavor-id** argument, the **openstack router create** command creates a default OVN router.

4. List your deployment's routers to verify the router creation:

   ```
   $ openstack router list -c ID -c Name -c Status -c HA
   ```

   Sample output

   ```
   +--------------------------------------+---------------------+--------+------+
   | ID                                   | Name                | Status | HA   |
   +--------------------------------------+---------------------+--------+------+
   | 9f5fec56-1829-4bad-abe5-7b4221649c8e | router1             | ACTIVE | True |
   | e9f25566-ff73-4a76-aeb4-969c819f9c47 | user-defined-router | ACTIVE | True |
   +--------------------------------------+---------------------+--------+------+
   ```

Additional resources

- Section 17.1, “Enabling router flavors and creating service providers”
- Section 17.2, “Creating a router flavor”

CHAPTER 18. CONFIGURING ALLOWED ADDRESS PAIRS

18.1. OVERVIEW OF ALLOWED ADDRESS PAIRS

In Red Hat OpenStack Platform (RHOSP) networking environments, an *allowed address pair* is when you identify a specific MAC address, IP address, or both to allow network traffic to pass through a port regardless of the subnet. When you define allowed address pairs, you are able to use protocols like Virtual Router Redundancy Protocol (VRRP) that float an IP address between two VM instances to enable fast data plane failover. A port whose IP address is a member of an allowed address pair of another port is referred to as a virtual port (vport).

IMPORTANT

In RHOSP networking environments, when creating a VM instance, do not bind the instance to a virtual port (vport). Instead, use a port whose IP address is not a member of another port’s allowed address pair.

Binding a vport to an instance prevents the instance from spawning and produces an error message similar to the following:

```
WARNING nova.virt.libvirt.driver [req-XXXX - - - default default] [instance:
XXXXXXXXX] Timeout waiting for [('network-vif-plugged', 'XXXXXXXXXX')] for
instance with vm_state building and task_state spawning.: eventlet.timeout.Timeout:
300 seconds
```

You define allowed address pairs using the Red Hat OpenStack Platform command-line client **openstack port** command.

IMPORTANT

Be aware that you should not use the default security group with a wider IP address range in an allowed address pair. Doing so can allow a single port to bypass security groups for all other ports within the same network.

For example, this command impacts all ports in the network and bypasses all security groups:

```
# openstack port set --allowed-address mac-address=3e:37:09:4b,ip-
address=0.0.0.0/0 9e67d44eab334f07bf82fa1b17d824b6
```

NOTE

With an ML2/OVN mechanism driver network back end, it is possible to create VIPs. However, the IP address assigned to a bound port using **allowed_address_pairs**, should match the virtual port IP address (/32).

If you use a CIDR format IP address for the bound port **allowed_address_pairs** instead, port forwarding is not configured in the back end, and traffic fails for any IP in the CIDR expecting to reach the bound IP port.

Additional resources

- port command in the *Command line interface reference*
- Section 18.2, “Creating a port and allowing one address pair”
- Section 18.3, “Adding allowed address pairs”

18.2. CREATING A PORT AND ALLOWING ONE ADDRESS PAIR

Creating a port with an allowed address pair enables network traffic to flow through the port regardless of the subnet.

IMPORTANT

Do not use the default security group with a wider IP address range in an allowed address pair. Doing so can allow a single port to bypass security groups for all other ports within the same network.

Procedure

- Use the following command to create a port and allow one address pair:

```
$ openstack port create --network <network> --allowed-address mac-address=
<mac_address>,ip-address=<ip_cidr> <port_name>
```

Additional resources

- port command in the *Command line interface reference*

18.3. ADDING ALLOWED ADDRESS PAIRS

You can add an allowed address pair to a port to enable network traffic to flow through the port regardless of the subnet.

IMPORTANT

Do not use the default security group with a wider IP address range in an allowed address pair. Doing so can allow a single port to bypass security groups for all other ports within the same network.

Procedure

- Use the following command to add allowed address pairs:

```
$ openstack port set --allowed-address mac-address=<mac_address>,ip-address=<ip_cidr>
<port>
```

NOTE

You cannot set an allowed-address pair that matches the **mac_address** and **ip_address** of a port. This is because such a setting has no effect since traffic matching the **mac_address** and **ip_address** is already allowed to pass through the port.

Additional resources

- port command in the *Command line interface reference*

CHAPTER 19. CONFIGURING SECURITY GROUPS

Security groups are sets of IP filter rules that control network and protocol access to and from instances, such as ICMP to allow you to ping an instance, and SSH to allow you to connect to an instance. The security group rules are applied to all instances within a project.

All projects have a default security group called **default**, which is used when you do not specify a security group for your instances. By default, the default security group allows all outgoing traffic and denies all incoming traffic from any source other than instances in the same security group. You can either add rules to the default security group or create a new security group for your project. You can apply one or more security groups to an instance during instance creation. To apply a security group to a running instance, apply the security group to a port attached to the instance.

When you create a security group, you can choose stateful or stateless in ML2/OVN deployments.

NOTE

Stateless security groups are not supported in ML2/OVS deployments.

Security groups are stateful by default and in most cases stateful security groups provide better control with less administrative overhead.

A stateless security group can provide significant performance benefits, because it bypasses connection tracking in the underlying firewall. But stateless security groups require more security group rules than stateful security groups. Stateless security groups also offer less granularity in some cases.

Stateless security group advantages

- Stateless security groups can be faster than stateful security groups
- Stateless security groups are the only viable security group option in applications that offload OpenFlow actions to hardware.

Stateless security group disadvantages

- Stateless security group rules do not automatically allow returning traffic. For example, if you create a rule to allow outgoing TCP traffic from a port that is in a stateless security group, you must also create a rule that allows incoming replies. Stateful security groups automatically allow the incoming replies.
- Control over those incoming replies may not be as granular as the control provided by stateful security groups.

In general, use the default stateful security group type unless your application is highly sensitive to performance or uses hardware offloading of OpenFlow actions.

NOTE

You cannot apply a role-based access control (RBAC)-shared security group directly to an instance during instance creation. To apply an RBAC-shared security group to an instance you must first create the port, apply the shared security group to that port, and then assign that port to the instance. See Adding a security group to a port .

19.1. CREATING A SECURITY GROUP

You can create a new security group to apply to instances and ports within a project.

Procedure

1. Optional: To ensure the security group you need does not already exist, review the available security groups and their rules:

   ```
   $ openstack security group list
   $ openstack security group rule list <sec_group>
   ```

   - Replace **<sec_group>** with the name or ID of the security group that you retrieved from the list of available security groups.

2. . Create your security group:

   ```
   $ openstack security group create [--stateless] mySecGroup
   ```

   - Optional: Include the **--stateless** option to create a stateless security group. Security groups are stateful by default.

     NOTE

     Only ML2/OVN deployments support stateless security groups.

3. Add rules to your security group:

   ```
   $ openstack security group rule create --protocol <protocol> \
   [--dst-port <port-range>] \
   [--remote-ip <ip-address> | --remote-group <group>] \
   [--ingress | --egress] mySecGroup
   ```

   - Replace **<protocol>** with the name of the protocol you want to allow to communicate with your instances.
   - Optional: Replace **<port-range>** with the destination port or port range to open for the protocol. Required for IP protocols TCP, UDP, and SCTP. Set to **-1** to allow all ports for the specified protocol. Separate port range values with a colon.
   - Optional: You can allow access only from specified IP addresses by using **--remote-ip** to specify the remote IP address block, or **--remote-group** to specify that the rule only applies to packets from interfaces that are a member of the remote group. If using **--remote-ip**, replace **<ip-address>** with the remote IP address block. You can use CIDR notation. If using **--remote-group**, replace **<group>** with the name or ID of the existing security group. If neither option is specified, then access is allowed to all addresses, as the remote IP access range defaults (IPv4 default: **0.0.0.0/0**; IPv6 default: **::/0**).
   - Specify the direction of network traffic the protocol rule applies to, either incoming (**ingress**) or outgoing (**egress**). If not specified, defaults to **ingress**.

     NOTE

     If you created a stateless security group, and you created a rule to allow outgoing TCP traffic from a port that is in the stateless security group, you must also create a rule that allows incoming replies.

4. Repeat step 3 until you have created rules for all the protocols that you want to allow to access your instances. The following example creates a rule to allow SSH connections to instances in the security group **mySecGroup**:

```
$ openstack security group rule create --protocol tcp \
 --dst-port 22 mySecGroup
```

19.2. UPDATING SECURITY GROUP RULES

You can update the rules of any security group that you have access to.

Procedure

1. Retrieve the name or ID of the security group that you want to update the rules for:

```
$ openstack security group list
```

2. Determine the rules that you need to apply to the security group.

3. Add rules to your security group:

```
$ openstack security group rule create --protocol <protocol> \
[--dst-port <port-range>] \
[--remote-ip <ip-address> | --remote-group <group>] \
[--ingress | --egress] <group_name>
```

- Replace **<protocol>** with the name of the protocol you want to allow to communicate with your instances.

- Optional: Replace **<port-range>** with the destination port or port range to open for the protocol. Required for IP protocols TCP, UDP, and SCTP. Set to **-1** to allow all ports for the specified protocol.Separate port range values with a colon.

- Optional: You can allow access only from specified IP addresses by using **--remote-ip** to specify the remote IP address block, or **--remote-group** to specify that the rule only applies to packets from interfaces that are a member of the remote group. If using **--remote-ip**, replace **<ip-address>** with the remote IP address block. You can use CIDR notation. If using **--remote-group**, replace **<group>** with the name or ID of the existing security group. If neither option is specified, then access is allowed to all addresses, as the remote IP access range defaults (IPv4 default: **0.0.0.0/0**; IPv6 default: **::/0**).

- Specify the direction of network traffic the protocol rule applies to, either incoming (**ingress**) or outgoing (**egress**). If not specified, defaults to **ingress**.

- Replace **<group_name>** with the name or ID of the security group that you want to apply the rule to.

4. Repeat step 3 until you have created rules for all the protocols that you want to allow to access your instances. The following example creates a rule to allow SSH connections to instances in the security group **mySecGroup**:

```
$ openstack security group rule create --protocol tcp \
 --dst-port 22 mySecGroup
```

19.3. DELETING SECURITY GROUP RULES

You can delete rules from a security group.

Procedure

1. Identify the security group that the rules are applied to:

   ```
   $ openstack security group list
   ```

2. Retrieve IDs of the rules associated with the security group:

   ```
   $ openstack security group show <sec-group>
   ```

3. Delete the rule or rules:

   ```
   $ openstack security group rule delete <rule> [<rule> ...]
   ```

   Replace **<rule>** with the ID of the rule to delete. You can delete more than one rule at a time by specifying a space-delimited list of the IDs of the rules to delete.

19.4. DELETING A SECURITY GROUP

You can delete security groups that are not associated with any ports.

Procedure

1. Retrieve the name or ID of the security group that you want to delete:

   ```
   $ openstack security group list
   ```

2. Retrieve a list of the available ports:

   ```
   $ openstack port list
   ```

3. Check each port for an associated security group:

   ```
   $ openstack port show <port-uuid> -c security_group_ids
   ```

   If the security group you want to delete is associated with any of the ports, then you must first remove the security group from the port. For more information, see Removing a security group from a port.

4. Delete the security group:

   ```
   $ openstack security group delete <group> [<group> ...]
   ```

   Replace **<group>** with the ID of the group that you want to delete. You can delete more than one group at a time by specifying a space-delimited list of the IDs of the groups to delete.

19.5. CONFIGURING SHARED SECURITY GROUPS

When you want one or more Red Hat OpenStack Platform (RHOSP) projects to be able to share data, you can use the RHOSP Networking service (neutron) RBAC policy feature to share a security group. You create security groups and Networking service role-based access control (RBAC) policies using the OpenStack Client.

You can apply a security group directly to an instance during instance creation, or to a port on the running instance.

NOTE

You cannot apply a role-based access control (RBAC)-shared security group directly to an instance during instance creation. To apply an RBAC-shared security group to an instance you must first create the port, apply the shared security group to that port, and then assign that port to the instance. See Adding a security group to a port .

Prerequisites

- You have at least two RHOSP projects that you want to share.
- In one of the projects, the *current project*, you have created a security group that you want to share with another project, the *target project*.
In this example, the **ping_ssh** security group is created:

  Example

  ```
  $ openstack security group create ping_ssh
  ```

Procedure

1. Log in to the overcloud for the current project that contains the security group.
2. Obtain the name or ID of the target project.

   ```
   $ openstack project list
   ```

3. Obtain the name or ID of the security group that you want to share between RHOSP projects.

   ```
   $ openstack security group list
   ```

4. Using the identifiers from the previous steps, create an RBAC policy using the **openstack network rbac create** command.
In this example, the ID of the target project is **32016615de5d43bb88de99e7f2e26a1e**. The ID of the security group is **5ba835b7-22b0-4be6-bdbe-e0722d1b5f24**:

   Example

   ```
   $ openstack network rbac create --target-project \
   32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
   --type security_group 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24
   ```

   --target-project

   specifies the project that requires access to the security group.

TIP

You can share data between *all* projects by using the **--target-all-projects** argument instead of **--target-project <target-project>**. By default, only the admin user has this privilege.

--action access_as_shared

specifies what the project is allowed to do.

--type

indicates that the target object is a security group.

5ba835b7-22b0-4be6-bdbe-e0722d1b5f24

is the ID of the particular security group which is being granted access to.

The target project is able to access the security group when running the OpenStack Client **security group** commands, in addition to being able to bind to its ports. No other users (other than administrators and the owner) are able to access the security group.

TIP

To remove access for the target project, delete the RBAC policy that allows it using the **openstack network rbac delete** command.

Additional resources

- Creating a security group in the *Creating and managing instances* guide
- security group create in the *Command line interface reference*
- network rbac create in the *Command line interface reference*

CHAPTER 20. LOGGING SECURITY GROUP ACTIONS

To monitor traffic flows into and out of a virtual machine (VM) instance, you can create packet logs for security groups. Each log generates a stream of data about packet flow events and appends it to a common log file on the Compute host from which the VM instance was launched.

You can associate any instance port with one or more security groups and define one or more rules for each security group. For example, you can create a rule to allow inbound SSH traffic to any virtual machine in a security group. You can create another rule in the same security group to allow virtual machines in that group to initiate and respond to ICMP (ping) messages.

Then you can create logs to record combinations of packet flow events. For example, the following command creates a log to capture all **ACCEPT** events in the security group security-group1.

```
$ openstack network log create my-log1 \
--resource-type security_group \
--resource security-group1 \
--event ACCEPT
```

You can create multiple logs to capture data about specific combinations of security groups and packet flow events.

You can configure the following parameters:

resource-type

You must set this required parameter to **security_group**.

resource (security group names)

You can optionally limit a log to a specific security group with the target argument. For example: **--resource security-group1**. If you do not specify a resource, the log will capture events from all security groups on the specified ports in the project.

event (types of events to log)

You can choose to log the following packet flow events:

- **DROP**: Log one **DROP** log entry for each incoming or outgoing session that is dropped.

  NOTE

  If you log dropped traffic on one or more security groups, the Networking service logs dropped traffic on all security groups.

- **ACCEPT**: Log one **ACCEPT** log entry for each new session that is allowed by the security group.

- **ALL** (drop and accept): Log all **DROP** and **ACCEPT** events. If you do not set –event **ACCEPT** or –event **DROP**, the Networking service defaults to **ALL**.

NOTE

The Networking service writes all log data to the same file on every Compute node: **/var/log/containers/openvswitch/ovn-controller.log**.

20.1. VERIFYING THAT SECURITY GROUP LOGGING IS ENABLED

To prepare your deployment for network packet logging, ensure that the logging service plug-in and logging extension are configured.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.

2. Enter the following command.

   ```
   $ openstack extension list --max-width 80 | grep logging
   ```

   If the logging service plug-in and extension are configured properly, the output includes the following:

   ```
   | Logging API Extension   | logging       | Provides a logging API   |
   ```

3. If the openstack extension list output does not include the Logging API Extension:

   a. Add **log** to the **NeutronPluginExtensions** parameter in an environment file.

   #### Example

   ```
   parameter_defaults:
     NeutronPluginExtensions: "qos,port_security,log"
   ```

   b. Run the **openstack overcloud deploy** command and include the core Orchestration templates, environment files, and this environment file.

Additional resources

- Creating your overcloud

20.2. CREATING LOG OBJECTS FOR SECURITY GROUPS

Create log objects with the resource type **security_group**.

Prerequisites

- You have created security groups

- You have created security group rules for the security groups

- You have assigned ports to the security groups

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.

2. Create a log by using the **openstack network log create** command with the appropriate set of arguments.

   #### Example 1: Log ACCEPT events from the security group sg1 on all ports

```
$ openstack network log create my-log1 \
--resource-type security_group \
--resource sg1 \
–event ACCEPT
```

Example 2: Log `ACCEPT` events from all security groups on all ports

```
openstack network log create my-log3 \
--resource-type security_group \
–event ACCEPT
```

3. Verify that the log was created:

   ```
   $ openstack network log list
   ```

20.3. LISTING AND VIEWING LOG OBJECTS FOR SECURITY GROUPS

You can list and view security group log objects.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.
2. To list all log objects in a project:

   ```
   $ openstack network log list
   ```

3. To view details of a log object:

   ```
   $ openstack network log show <log_object_name>
   ```

   Replace <log_object_name> with the name of the log object.

20.4. ENABLING AND DISABLING LOG OBJECTS FOR SECURITY GROUPS

When you create a log object, it is enabled by default. You can disable or enable a log object.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.
2. To disable a log object, enter the following command:

   ```
   $ openstack network log set --disable <log_object_name>
   ```

   Replace <log_object_name> with the name of the log object.

3. To enable a log object, enter the following command:

   ```
   $ openstack network log set --enable <log_object_name>
   ```

Replace <log_object_name> with the name of the log object.

20.5. RENAMING A LOG OBJECT FOR SECURITY GROUPS

You can change the name of a log object.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.
2. To rename a log object, enter the following command:

   ```
   $ openstack network log set --name <new_log_object_name> <object>
   ```

   Replace <new_log_object_name> with the new name of the log object. Replace <object> with the old name or ID of the log object.

20.6. DELETING A LOG OBJECT FOR SECURITY GROUPS

You can delete log objects.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.
2. To delete one or more log objects, enter the following command:

   ```
   $ openstack network log delete <log_object_name> [<log_object_name> ...]
   ```

   Replace <log_object_name> with the name of the log object to delete. To delete multiple log objects, enter a list of log object names, separated by spaces.

20.7. ACCESSING SECURITY GROUP LOG CONTENT

The Networking service aggregates security group logs from all VM instances on a Compute node in one location on the Compute node host: **/var/log/containers/openvswitch/ovn-controller.log**.

The log file contains other log objects. Security group log entries include the string **acl_log**.

20.8. SAMPLE SECURITY GROUP LOG CONTENT

Log content includes the following data:

- A timestamp of the packet flow.
- A status of the flow: **ACCEPT** or **DROP**.
- An indication of the originator of the flow. For example, which project or log resource generated the events.
- An identifier of the associated instance interface (Neutron port ID).
- Layer 2, 3 and 4 information such as MAC, address, port, and protocol.

Example: logged data from an ACCEPT event

```
2022-11-30T03:29:12.868Z|00111|acl_log(ovn_pinctrl1)|INFO|name="neutron-bc53f8df-2318-4d08-
8e12-89e92b08deec", verdict=allow, severity=info, direction=from-lport:
udp,vlan_tci=0x0000,dl_src=fa:16:3e:70:c4:45,dl_dst=fa:16:3e:66:8b:18,nw_src=192.168.100.59,nw_ds
=192.168.100.1,nw_tos=0,nw_ecn=0,nw_ttl=64,tp_src=68,tp_dst=67
```

20.9. ADJUSTING RATE AND BURST LIMITS FOR SECURITY GROUP LOGGING

To avoid overwhelming the control plane with the transmission of logging data, the Networking service sets limits on the maximum number of packets logged per second. You can change this limit using the **NeutronOVNLoggingRateLimit** parameter.

When logging packet transmission reaches the rate limit, the Networking service queues the excess packets to be logged. You can change the maximum number of queued packets using the **NeutronOVNLoggingBurstLimit** parameter.

The default values are **NeutronOVNLoggingRateLimit**:100 packets per second and **NeutronOVNLoggingBurstLimit**:25 packets in queue. These are also the minimum required values. The limits do not operate correctly with lower values.

Logging rate and burst limits do not limit control of data traffic. They limit only the transmission of logging data.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Set the parameters in a custom environment file. For example, **sg-logging.yaml**.

   #### Example

   ```
   parameter_defaults:
   ...
      NeutronOVNLoggingRateLimit=450
      NeutronOVNLoggingBurstLimit=50
   ```

4. Run the deployment command and include the core Heat templates, other environment files, and the custom roles data file in your deployment command with the **-r** option.

   #### IMPORTANT

   The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

   #### Example

```
$ openstack overcloud deploy --templates <core_heat_templates> \
-e <other_environment_files> \
-e /home/stack/templates/neutron-ovn-dvr-ha.yaml
```

CHAPTER 21. COMMON ADMINISTRATIVE NETWORKING TASKS

Sometimes you might need to perform administration tasks on the Red Hat OpenStack Platform Networking service (neutron) such as configuring the Layer 2 Population driver or specifying the name assigned to ports by the internal DNS.

21.1. CONFIGURING THE L2 POPULATION DRIVER

The L2 Population driver is used in Networking service (neutron) ML2/OVS environments to enable broadcast, multicast, and unicast traffic to scale out on large overlay networks. By default, Open vSwitch GRE and VXLAN replicate broadcasts to every agent, including those that do not host the destination network. This design requires the acceptance of significant network and processing overhead. The alternative design introduced by the L2 Population driver implements a partial mesh for ARP resolution and MAC learning traffic; it also creates tunnels for a particular network only between the nodes that host the network. This traffic is sent only to the necessary agent by encapsulating it as a targeted unicast.

Prerequisites

- You must have RHOSP administrator privileges.
- The Networking service must be using the ML2/OVS mechanism driver.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Create a custom YAML environment file.

Example

```
$ vi /home/stack/templates/my-environment.yaml
```

4. Your environment file must contain the keywords **parameter_defaults**. Under these keywords, add the following lines:

```
parameter_defaults:
  NeutronMechanismDrivers: ['openvswitch', 'l2population']
  NeutronEnableL2Pop: 'True'
  NeutronEnableARPResponder: true
```

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /home/stack/templates/my-environment.yaml
```

Verification

1. Obtain the IDs for the OVS agents.

```
$ openstack network agent list -c ID -c Binary
```

Sample output

```
+--------------------------------------+---------------------------+
| ID                                   | Binary                    |
+--------------------------------------+---------------------------+
| 003a8750-a6f9-468b-9321-a6c03c77aec7 | neutron-openvswitch-agent |
| 02bbbb8c-4b6b-4ce7-8335-d1132df31437 | neutron-l3-agent          |
| 0950e233-60b2-48de-94f6-483fd0af16ea | neutron-openvswitch-agent |
| 115c2b73-47f5-4262-bc66-8538d175029f | neutron-openvswitch-agent |
| 2a9b2a15-e96d-468c-8dc9-18d7c2d3f4bb | neutron-metadata-agent    |
| 3e29d033-c80b-4253-aaa4-22520599d62e | neutron-dhcp-agent        |
| 3ede0b64-213d-4a0d-9ab3-04b5dfd16baa | neutron-dhcp-agent        |
| 462199be-0d0f-4bba-94da-603f1c9e0ec4 | neutron-sriov-nic-agent   |
| 54f7c535-78cc-464c-bdaa-6044608a08d7 | neutron-l3-agent          |
| 6657d8cf-566f-47f4-856c-75600bf04828 | neutron-metadata-agent    |
| 733c66f1-a032-4948-ba18-7d1188a58483 | neutron-l3-agent          |
| 7e0a0ce3-7ebb-4bb3-9b89-8cccf8cb716e | neutron-openvswitch-agent |
| dfc36468-3a21-4a2d-84c3-2bc40f224235 | neutron-metadata-agent    |
| eb7d7c10-69a2-421e-bd9e-aec3edfe1b7c | neutron-openvswitch-agent |
| ef5219b4-ee49-4635-ad04-048291209373 | neutron-sriov-nic-agent   |
| f36c7af0-e20c-400b-8a37-4ffc5d4da7bd | neutron-dhcp-agent        |
+--------------------------------------+---------------------------+
```

2. Using an ID from one of the OVS agents, confirm that the L2 Population driver is set on the OVS agent.

Example

This example verifies the configuration of the L2 Population driver on the **neutron-openvswitch-agent** with ID **003a8750-a6f9-468b-9321-a6c03c77aec7**:

```
$ openstack network agent show 003a8750-a6f9-468b-9321-a6c03c77aec7 -c configuration
-f json | grep l2_population
```

Sample output

```
    "l2_population": true,
```

3. Ensure that the ARP responder feature is enabled for the OVS agent.

Example

```
$ openstack network agent show 003a8750-a6f9-468b-9321-a6c03c77aec7 -c configuration
-f json | grep arp_responder_enabled
```

Sample output

```
    "arp_responder_enabled": true,
```

Additional resources

- OVN supported DHCP options
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

21.2. TUNING KEEPALIVED TO AVOID VRRP PACKET LOSS

If the number of highly available (HA) routers on a single host is high, when an HA router fail over occurs, the Virtual Router Redundancy Protocol (VRRP) messages might overflow the IRQ queues. This overflow stops Open vSwitch (OVS) from responding and forwarding those VRRP messages.

To avoid VRRP packet overload, you must increase the VRRP advertisement interval using the **ha_vrrp_advert_int** parameter in the **ExtraConfig** section for the Controller role.

Procedure

1. Log in to the undercloud as the stack user, and source the **stackrc** file to enable the director command line tools.

   Example

   ```
   $ source ~/stackrc
   ```

2. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

   TIP

   The Red Hat OpenStack Platform Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

3. In the YAML environment file, increase the VRRP advertisement interval using the **ha_vrrp_advert_int** argument with a value specific for your site. (The default is **2** seconds.) You can also set values for gratuitous ARP messages:

   ha_vrrp_garp_master_repeat

The number of gratuitous ARP messages to send at one time after the transition to the master state. (The default is 5 messages.)

ha_vrrp_garp_master_delay

The delay for second set of gratuitous ARP messages after the lower priority advert is received in the master state. (The default is 5 seconds.)

Example

```
parameter_defaults:
  ControllerExtraConfig:
    neutron::agents::l3::ha_vrrp_advert_int: 7
    neutron::config::l3_agent_config:
      DEFAULT/ha_vrrp_garp_master_repeat:
        value: 5
      DEFAULT/ha_vrrp_garp_master_delay:
        value: 5
```

4. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-environment.yaml
```

Additional resources

- 2.1.2 Data Forwarding Rules, Subsection 2 in *RFC 4541*
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

21.3. SPECIFYING THE NAME THAT DNS ASSIGNS TO PORTS

You can specify the name assigned to ports by the internal DNS when you enable the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) DNS domain for ports extension (**dns_domain_ports**).

You enable the DNS domain for ports extension by declaring the RHOSP Orchestration (heat) **NeutronPluginExtensions** parameter in a YAML-formatted environment file. Using a corresponding parameter, **NeutronDnsDomain**, you specify your domain name, which overrides the default value, **openstacklocal**. After redeploying your overcloud, you can use the OpenStack Client port commands, **port set** or **port create**, with **--dns-name** to assign a port name.

IMPORTANT

You must enable the DNS domain for ports extension (**dns_domain_ports**) for DNS to internally resolve names for ports in your RHOSP environment. Using the **NeutronDnsDomain** default value, **openstacklocal**, means that the Networking service does not internally resolve port names for DNS.

Also, when the DNS domain for ports extension is enabled, the Compute service automatically populates the **dns_name** attribute with the **hostname** attribute of the instance during the boot of VM instances. At the end of the boot process, dnsmasq recognizes the allocated ports by their instance hostname.

Procedure

1. Log in to the undercloud as the stack user, and source the **stackrc** file to enable the director command line tools.

   Example

   ```
   $ source ~/stackrc
   ```

2. Create a custom YAML environment file (**my-neutron-environment.yaml**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

   TIP

   The undercloud includes a set of Orchestration service templates that form the plan for your overcloud creation. You can customize aspects of the overcloud with environment files, which are YAML-formatted files that override parameters and resources in the core Orchestration service template collection. You can include as many environment files as necessary.

3. In the environment file, add a **parameter_defaults** section. Under this section, add the DNS domain for ports extension, **dns_domain_ports**.

   Example

   ```
   parameter_defaults:
     NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
   ```

   NOTE

   If you set **dns_domain_ports**, ensure that the deployment does not also use **dns_domain**, the DNS Integration extension. These extensions are incompatible, and both extensions cannot be defined simultaneously.

4. Also in the **parameter_defaults** section, add your domain name (**example.com**) using the **NeutronDnsDomain** parameter.

   ### Example

   ```
   parameter_defaults:
     NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
     NeutronDnsDomain: "example.com"
   ```

5. Run the **openstack overcloud deploy** command and include the core Orchestration templates, environment files, and this new environment file.

   IMPORTANT

   The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

   ### Example

   ```
   $ openstack overcloud deploy --templates \
   -e [your-environment-files] \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
   environment.yaml
   ```

Verification

1. Log in to the overcloud, and create a new port (**new_port**) on a network (**public**). Assign a DNS name (**my_port**) to the port.

   ### Example

   ```
   $ source ~/overcloudrc
   $ openstack port create --network public --dns-name my_port new_port
   ```

2. Display the details for your port (**new_port**).

   ### Example

   ```
   $ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
   ```

   ### Output

   ```
   +-------------------------+----------------------------------------------+
   | Field                   | Value                                        |
   +-------------------------+----------------------------------------------+
   | dns_assignment          | fqdn='my_port.example.com',                  |
   |                         | hostname='my_port',                          |
   |                         | ip_address='10.65.176.113'                   |
   | dns_domain              | example.com                                  |
   | dns_name                | my_port                                      |
   | name                    | new_port                                     |
   +-------------------------+----------------------------------------------+
   ```

Under **dns_assignment**, the fully qualified domain name (**fqdn**) value for the port contains a concatenation of the DNS name (**my_port**) and the domain name (**example.com**) that you set earlier with **NeutronDnsDomain**.

3. Create a new VM instance (**my_vm**) using the port (**new_port**) that you just created.

   Example

   ```
   $ openstack server create --image rhel --flavor m1.small --port new_port my_vm
   ```

4. Display the details for your port (**new_port**).

   Example

   ```
   $ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
   ```

   Output

   ```
   +-------------------------+----------------------------------------------+
   | Field                   | Value                                        |
   +-------------------------+----------------------------------------------+
   | dns_assignment          | fqdn='my_vm.example.com',                    |
   |                         | hostname='my_vm',                            |
   |                         | ip_address='10.65.176.113'                   |
   | dns_domain              | example.com                                  |
   | dns_name                | my_vm                                        |
   | name                    | new_port                                     |
   +-------------------------+----------------------------------------------+
   ```

   Note that the Compute service changes the **dns_name** attribute from its original value (**my_port**) to the name of the instance with which the port is associated (**my_vm**).

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- port in the *Command line interface reference*
- server create in the *Command line interface reference*

21.4. ASSIGNING DHCP ATTRIBUTES TO PORTS

You can use Red Hat Openstack Plaform (RHOSP) Networking service (neutron) extensions to add networking functions. You can use the extra DHCP option extension (**extra_dhcp_opt**) to configure ports of DHCP clients with DHCP attributes. For example, you can add a PXE boot option such as **tftp-server**, **server-ip-address**, or **bootfile-name** to a DHCP client port.

The value of the **extra_dhcp_opt** attribute is an array of DHCP option objects, where each object contains an **opt_name** and an **opt_value**. IPv4 is the default version, but you can change this to IPv6 by including a third option, **ip-version=6**.

When a VM instance starts, the RHOSP Networking service supplies port information to the instance using DHCP protocol. If you add DHCP information to a port already connected to a running instance, the instance only uses the new DHCP port information when the instance is restarted.

Some of the more common DHCP port attributes are: **bootfile-name**, **dns-server**, **domain-name**, **mtu**, **server-ip-address**, and **tftp-server**. For the complete set of acceptable values for **opt_name**, refer to the DHCP specification.

Prerequisites

- You must have RHOSP administrator privileges.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-environment.yaml
   ```

4. Your environment file must contain the keywords **parameter_defaults**. Under these keywords, add the extra DHCP option extension, **extra_dhcp_opt**.

   Example

   ```
   parameter_defaults:
     NeutronPluginExtensions: "qos,port_security,extra_dhcp_opt"
   ```

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   IMPORTANT

   The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

   Example

   ```
   $ openstack overcloud deploy --templates \
   -e <your_environment_files> \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-environment.yaml
   ```

Verification

1. Source your credentials file.

Example

```
$ source ~/overcloudrc
```

2. Create a new port (**new_port**) on a network (**public**). Assign a valid attribute from the DHCP specification to the new port.

   Example

   ```
   $ openstack port create --extra-dhcp-option \
   name=domain-name,value=test.domain --extra-dhcp-option \
   name=ntp-server,value=192.0.2.123 --network public new_port
   ```

3. Display the details for your port (**new_port**).

   Example

   ```
   $ openstack port show new_port -c extra_dhcp_opts
   ```

   Sample output

   ```
   +-----------------+-----------------------------------------------------------------+
   | Field           | Value                                                           |
   +-----------------+-----------------------------------------------------------------+
   | extra_dhcp_opts | ip_version='4', opt_name='domain-name', opt_value='test.domain' |
   |                 | ip_version='4', opt_name='ntp-server', opt_value='192.0.2.123'  |
   +-----------------+-----------------------------------------------------------------+
   ```

Additional resources

- OVN supported DHCP options
- Dynamic Host Configuration Protocol (DHCP) and Bootstrap Protocol (BOOTP) Parameters
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- port create in the *Command line interface reference*
- port show in the *Command line interface reference*

21.5. ENABLING NUMA AFFINITY ON PORTS

To enable users to create instances with NUMA affinity on the port, you must load the Red Hat Openstack Plaform (RHOSP) Networking service (neutron) extension, **port_numa_affinity_policy**.

Prerequisites

- Access to the undercloud host and credentials for the stack user.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. To enable the **port_numa_affinity_policy** extension, open the environment file where the **NeutronPluginExtensions** parameter is defined, and add **port_numa_affinity_policy** to the list:

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_numa_affinity_policy"
```

4. Add the environment file that you modified to the stack with your other environment files, and redeploy the overcloud:

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /home/stack/templates/<custom_environment_file>.yaml
```

Verification

1. Source your credentials file.

Example

```
$ source ~/overcloudrc
```

2. Create a new port.
When you create a port, use one of the following options to specify the NUMA affinity policy to apply to the port:

- **--numa-policy-required** - NUMA affinity policy required to schedule this port.

- **--numa-policy-preferred** - NUMA affinity policy preferred to schedule this port.

- **--numa-policy-legacy** - NUMA affinity policy using legacy mode to schedule this port.

Example

```
$ openstack port create --network public \
  --numa-policy-legacy  myNUMAAffinityPort
```

3. Display the details for your port.

Example

```
$ openstack port show myNUMAAffinityPort -c numa_affinity_policy
```

Sample output

When the extension is loaded, the **Value** column should read, **legacy**, **preferred** or **required**. If the extension has failed to load, **Value** reads **None**:

```
+----------------------+--------+
| Field                | Value  |
+----------------------+--------+
| numa_affinity_policy | legacy |
+----------------------+--------+
```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Creating an instance with NUMA affinity on the port in the *Creating and managing instances* guide

21.6. LOADING KERNEL MODULES

Some features in Red Hat OpenStack Platform (RHOSP) require certain kernel modules to be loaded. For example, the OVS firewall driver requires you to load the **nf_conntrack_proto_gre** kernel module to support GRE tunneling between two VM instances.

By using a special Orchestration service (heat) parameter, **ExtraKernelModules**, you can ensure that heat stores configuration information about the required kernel modules needed for features like GRE tunneling. Later, during normal module management, these required kernel modules are loaded.

Procedure

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   ### Example

   ```
   $ vi /home/stack/templates/my-modules-environment.yaml
   ```

   ### TIP

   Heat uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

2. In the YAML environment file under **parameter_defaults**, set **ExtraKernelModules** to the name of the module that you want to load.

   ### Example

   ```
   ComputeParameters:
   ```

```
  ExtraKernelModules:
    nf_conntrack_proto_gre: {}
ControllerParameters:
  ExtraKernelModules:
    nf_conntrack_proto_gre: {}
```

3. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-modules-
environment.yaml
```

Verification

- If heat has properly loaded the module, you should see output when you run the **lsmod** command on the Compute node:

Example

```
sudo lsmod | grep nf_conntrack_proto_gre
```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

21.7. LIMITING QUERIES TO THE METADATA SERVICE

To protect the RHOSP environment against cyber threats such as denial of service (DoS) attacks, the Networking service (neutron) offers administrators the ability to limit the rate at which VM instances can query the Compute metadata service. Administrators do this by assigning values to a set of parameters in the **metadata_rate_limiting** section of the **neutron.conf** configuration file. The Networking service uses these parameters to configure HAProxy servers to perform the rate limiting. The HAProxy servers run inside L3 routers and DHCP agents in the OVS back end, and inside the metadata service in the OVN back end.

Prerequisites

- You have access to the RHOSP Compute nodes and permission to update configuration files.
- Your RHOSP environment uses IPv4 networking. Currently, the Networking service does not support metadata rate limiting on IPv6 networks.

- This procedure requires you to restart the OVN metadata service or the OVS metadata agent. Schedule this activity for a maintenance window to minimize the operational impact of any potential disruption.

Procedure

1. On every Compute node, in the **metadata_rate_limiting** section of **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf**, set values for the following parameters:

   rate_limit_enabled

   enables you to limit the rate of metadata requests. The default value is **false**. Set the value to **true** to enable metadata rate limiting.

   ip_versions

   the IP version, **4**, used for metadata IP addresses on which you want to control query rates. RHOSP does not yet support metadata rate limiting for IPv6 networks.

   base_window_duration

   the time span, in seconds, during which query requests are limited. The default value is **10** seconds.

   base_query_rate_limit

   the maximum number of requests allowed during the **base_window_duration**. The default value is **10** requests.

   burst_window_duration

   the time span, in seconds, that a request rate higher than the **base_window_duration** is allowed. The default value is **10** seconds.

   burst_query_rate_limit

   the maximum number of requests allowed during the **burst_window_duration**. The default value is **10** requests.

   #### Example

   In this example, the Networking service is configured for a **base** time and rate that allows instances to query the IPv4 metadata service IP address 6 times over a 60 second period. The Networking service is also configured for a **burst** time and rate that allows a higher rate of 2 queries during shorter periods of 10 seconds each:

   ```
   [metadata_rate_limiting]
   rate_limit_enabled = True
   ip_versions = 4
   base_window_duration = 60
   base_query_rate_limit = 6
   burst_window_duration = 10
   burst_query_rate_limit = 2
   ```

2. Restart the metadata service.
   Depending on the Networking service mechanism driver your deployment uses, do one of the following:

   #### ML2/OVN

   On the Compute nodes, restart **tripleo_ovn_metadata_agent.service**.

ML2/OVS

On the Compute nodes, restart **tripleo_neutron_metadata_agent.service**.

CHAPTER 22. CONFIGURING LAYER 3 HIGH AVAILABILITY (HA)

22.1. RHOSP NETWORKING SERVICE WITHOUT HIGH AVAILABILITY (HA)

Red Hat OpenStack Platform (RHOSP) Networking service deployments without any high availability (HA) features are vulnerable to physical node failures.

In a typical deployment, projects create virtual routers, which are scheduled to run on physical Networking service Layer 3 (L3) agent nodes. This becomes an issue when you lose an L3 agent node and the dependent virtual machines subsequently lose connectivity to external networks. Any floating IP addresses are also unavailable. In addition, connectivity is lost between any networks that the router hosts.

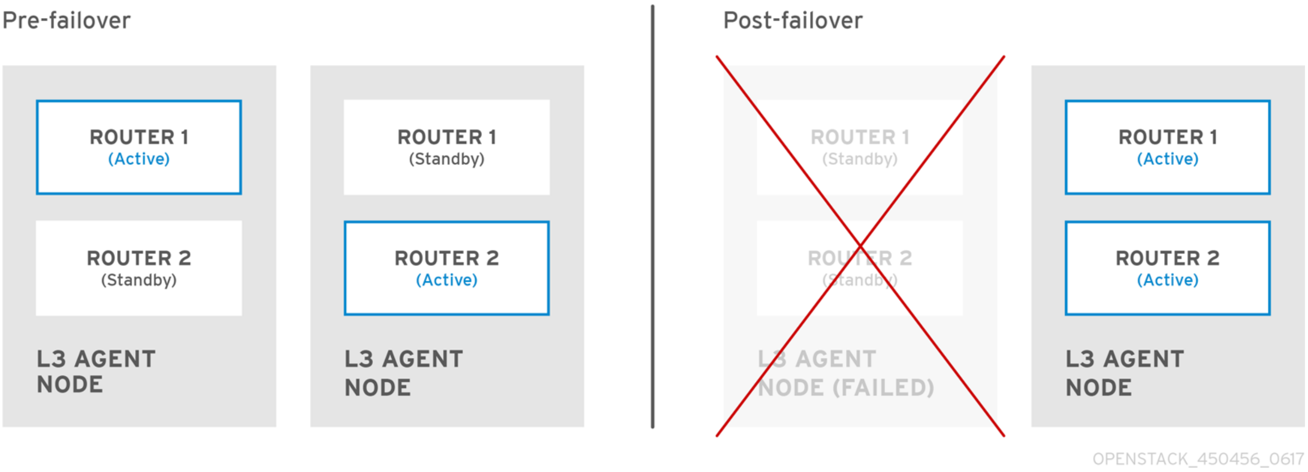
22.2. OVERVIEW OF LAYER 3 HIGH AVAILABILITY (HA)

This active/passive high availability (HA) configuration uses the industry standard VRRP (as defined in RFC 3768) to protect project routers and floating IP addresses. A virtual router is randomly scheduled across multiple Red Hat OpenStack Platform (RHOSP) Networking service nodes, with one designated as the *active* router, and the remainder serving in a *standby* role.

NOTE

To deploy Layer 3 (L3) HA, you must maintain similar configuration on the redundant Networking service nodes, including floating IP ranges and access to external networks.

In the following diagram, the active **Router1** and **Router2** routers are running on separate physical L3 Networking service agent nodes. L3 HA has scheduled backup virtual routers on the corresponding nodes, ready to resume service in the case of a physical node failure. When the L3 agent node fails, L3 HA reschedules the affected virtual router and floating IP addresses to a working node:



During a failover event, instance TCP sessions through floating IPs remain unaffected, and migrate to the new L3 node without disruption. Only SNAT traffic is affected by failover events.

The L3 agent is further protected when in an active/active HA mode.

Additional resources

- Virtual Router Redundancy Protocol (VRRP)

22.3. LAYER 3 HIGH AVAILABILITY (HA) FAILOVER CONDITIONS

Layer 3 (L3) high availability (HA) for the Red Hat OpenStack Platform (RHOSP) Networking service automatically reschedules protected resources in the following events:

- The Networking service L3 agent node shuts down or otherwise loses power because of a hardware failure.
- The L3 agent node becomes isolated from the physical network and loses connectivity.

NOTE

Manually stopping the L3 agent service does not induce a failover event.

22.4. PROJECT CONSIDERATIONS FOR LAYER 3 HIGH AVAILABILITY (HA)

Red Hat OpenStack Platform (RHOSP) Networking service Layer 3 (L3) high availability (HA) configuration occurs in the back end and is invisible to the project. Projects can continue to create and manage their virtual routers as usual, however there are some limitations to be aware of when designing your L3 HA implementation:

- L3 HA supports up to 255 virtual routers per project.
- Internal VRRP messages are transported within a separate internal network, created automatically for each project. This process occurs transparently to the user.
- When implementing high availability (HA) routers on ML2/OVS, each L3 agent spawns **haproxy** and **neutron-keepalived-state-change-monitor** processes for each router. Each process consumes approximately 20MB of memory. By default, each HA router resides on three L3 agents and consumes resources on each of the nodes. Therefore, when sizing your RHOSP networks, ensure that you have allocated enough memory to support the number of HA routers that you plan to implement.

22.5. HIGH AVAILABILITY (HA) CHANGES TO THE RHOSP NETWORKING SERVICE

The Red Hat OpenStack Platform (RHOSP) Networking service (neutron) API has been updated to allow administrators to set the **--ha=True/False** flag when creating a router, which overrides the default configuration of **l3_ha** in **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf**.

- **High availability (HA) changes to neutron-server:**
    - Layer 3 (L3) HA assigns the active role randomly, regardless of the scheduler used by the Networking service (whether random or leastrouter).
    - The database schema has been modified to handle allocation of virtual IP addresses (VIPs) to virtual routers.
    - A transport network is created to direct L3 HA traffic.
- **HA changes to the Networking service L3 agent:**

- A new keepalived manager has been added, providing load-balancing and HA capabilities.
- IP addresses are converted to VIPs.

22.6. ENABLING LAYER 3 HIGH AVAILABILITY (HA) ON RHOSP NETWORKING SERVICE NODES

During installation, Red Hat OpenStack Platform (RHOSP) director enables high availability (HA) for virtual routers by default when you have at least two RHOSP Controllers and are not using distributed virtual routing (DVR). Using an RHOSP Orchestration service (heat) parameter, **max_l3_agents_per_router**, you can set the maximum number of RHOSP Networking service Layer 3 (L3) agents on which an HA router is scheduled.

Prerequisites

- Your RHOSP deployment does not use DVR.
- You have at least two RHOSP Controllers deployed.

Procedure

1. Log in to the undercloud as the stack user, and source the **stackrc** file to enable the director command line tools.

   Example

   ```
   $ source ~/stackrc
   ```

2. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

   TIP

   The Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

3. Set the **NeutronL3HA** parameter to **true** in the YAML environment file. This ensures HA is enabled even if director did not set it by default.

   ```
   parameter_defaults:
     NeutronL3HA: 'true'
   ```

4. Set the maximum number of L3 agents on which an HA router is scheduled.
   Set the **max_l3_agents_per_router** parameter to a value between the minimum and total number of network nodes in your deployment. (A zero value indicates that the router is scheduled on every agent.)

   Example

   –

```
parameter_defaults:
  NeutronL3HA: 'true'
  ControllerExtraConfig:
    neutron::server::max_l3_agents_per_router: 2
```

In this example, if you deploy four Networking service nodes, only two L3 agents protect each HA virtual router: one active, and one standby.

If you set the value of **max_l3_agents_per_router** to be greater than the number of available network nodes, you can scale out the number of standby routers by adding new L3 agents. For every new L3 agent node that you deploy, the Networking service schedules additional standby versions of the virtual routers until the **max_l3_agents_per_router** limit is reached.

5. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-environment.yaml
```

NOTE

When **NeutronL3HA** is set to **true**, all virtual routers that are created default to HA routers. When you create a router, you can override the HA option by including the **--no-ha** option in the **openstack router create** command:

```
# openstack router create --no-ha
```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

22.7. REVIEWING HIGH AVAILABILITY (HA) RHOSP NETWORKING SERVICE NODE CONFIGURATIONS

Procedure

- Run the **ip address** command within the virtual router namespace to return a high availability (HA) device in the result, prefixed with *ha-*.

```
# ip netns exec qrouter-b30064f9-414e-4c98-ab42-646197c74020 ip address
```

```
<snip>
2794: ha-45249562-ec: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state DOWN group default
link/ether 12:34:56:78:2b:5d brd ff:ff:ff:ff:ff:ff
inet 169.254.0.2/24 brd 169.254.0.255 scope global ha-54b92d86-4f
```

With Layer 3 HA enabled, virtual routers and floating IP addresses are protected against individual node failure.

CHAPTER 23. USING AVAILABILITY ZONES TO MAKE NETWORK RESOURCES HIGHLY AVAILABLE

Starting with version 16.2, Red Hat OpenStack Platform (RHOSP) supports RHOSP Networking service (neutron) availability zones (AZs).

AZs enable you to make your RHOSP network resources highly available. You can group network nodes that are attached to different power sources on different AZs, and then schedule crucial services to be on separate AZs.

Often Networking service AZs are used in conjunction with Compute service (nova) AZs to ensure that customers use specific virtual networks that are local to a physical site that workloads run on. For more information on Distributed Compute Node architecture see, the *Deploying a Distributed Compute Node architecture* guide.

23.1. ABOUT NETWORKING SERVICE AVAILABILITY ZONES

The required extensions that provide Red Hat OpenStack Platform (RHOSP) Networking service (neutron) availability zones (AZ) functionality are **availability_zone**, **router_availability_zone**, and **network_availability_zone**. The Modular Layer 2 plug-in with the Open vSwitch (ML2/OVS) mechanism driver support all of these extensions.

NOTE

The Modular Layer 2 plug-in with the Open Virtual Network (ML2/OVN) mechanism driver supports only router availability zones. ML2/OVN has a distributed DHCP server, so supporting network AZs is unnecessary.

When you create your network resource, you can specify an AZ by using the OpenStack client command line option, **--availability-zone-hint**. The AZ that you specify is added to the AZ hint list. However, the AZ attribute is not actually set until the resource is scheduled. The actual AZ that is assigned to your network resource can vary from the AZ that you specified with the hint option. The reasons for this mismatch can be that there is a zone failure, or that the zone specified has no remaining capacity.

You can configure the Networking service for a default AZ, in case users fail to specify an AZ when they create a network resource. In addition to setting a default AZ, you can also configure specific drivers to schedule networks and routers on AZs.

Additional resources

- Configuring Network service availability zones with ML2/OVS
- Configuring Network service availability zones with ML2/OVN
- Manually Assigning availability zones to networks and routers

23.2. CONFIGURING NETWORK SERVICE AVAILABILITY ZONES FOR ML2/OVS

You can set one or more default availability zones (AZs) that are automatically assigned by the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) when users create networks and routers. In addition, you can also set the network and router drivers that the Networking service uses to schedule these resources for a respective AZ.

The information contained in this topic is for deployments that run the RHOSP Networking service that uses the Module Layer 2 plug-in with the Open vSwitch mechanism driver (ML2/OVS).

Prerequisites

- Deployed RHOSP 16.2 or later.
- Running the RHOSP Networking service that uses the ML2/OVS mechanism driver.
- When using Networking service AZs in distributed compute node (DCN) environments, you must match the Networking service AZ names to the Compute service (nova) AZ names.
  For more information, see the *Deploying a Distributed Compute Node architecture* guide.

Procedure

1. Log in to the undercloud as the **stack** user, and source the **stackrc** file to enable the director command line tools.

   Example

   ```
   $ source ~/stackrc
   ```

2. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

   TIP

   The Red Hat OpenStack Platform Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

3. In the YAML environment file, under **parameter_defaults**, enter the **NeutronDefaultAvailabilityZones** parameter and one or more AZs. The Networking service assigns these AZs if a user fails to specify an AZ with the **--availability-zone-hint** option when creating a network or a router.

   IMPORTANT

   In DCN environments, you must match the Networking service AZ names with Compute service AZ names.

   Example

   ```
   parameter_defaults:
     NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
   ```

4. Determine the AZs for the DHCP and the L3 agents, by entering values for the parameters, **NeutronDhcpAgentAvailabilityZone** and **NeutronL3AgentAvailabilityZone**, respectively.

Example

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  NeutronL3AgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronDhcpAgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
```

> **IMPORTANT**
>
> In DCN environments, define a single AZ for **NeutronDhcpAgentAvailabilityZone** so that ports are scheduled in the AZ relevant to the particular edge site.

5. By default, the network and router schedulers are **AZAwareWeightScheduler** and **AZLeastRoutersScheduler**, respectively. If you want to change one or both of these, enter the new schedulers with the **NeutronNetworkSchedulerDriver** and **NeutronRouterSchedulerDriver** parameters, respectively.

Example

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  NeutronL3AgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronDhcpAgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronNetworkSchedulerDriver:
'neutron.scheduler.dhcp_agent_scheduler.AZAwareWeightScheduler'
  NeutronRouterSchedulerDriver:
'neutron.scheduler.l3_agent_scheduler.AZLeastRoutersScheduler'
```

6. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

> **IMPORTANT**
>
> The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <your-environment-files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
my-neutron-environment.yaml
```

Verification

- Confirm that availability zones are properly defined, by running the **availability zone list** command.

Example

```
$ openstack availability zone list
```

Sample output

```
+----------------+-------------+
| Zone Name      | Zone Status |
+----------------+-------------+
| az-central     | available   |
| az-datacenter1 | available   |
| az-datacenter2 | available   |
+----------------+-------------+
```

Additional resources

- About Networking service availability zones
- Configuring Network service availability zones with ML2/OVN
- Manually Assigning availability zones to networks and routers

23.3. CONFIGURING NETWORK SERVICE AVAILABILITY ZONES WITH ML2/OVN

You can set one or more default availability zones (AZs) that are automatically assigned by the Red Hat OpenStack Platform (RHOSP) Networking service (neutron) when users create routers. In addition, you can also set the router driver that the Networking service uses to schedule these resources for a respective AZ.

The information contained in this topic is for deployments that run the RHOSP Networking service that uses the Modular Layer 2 plug-in with the Open Virtual Network (ML2/OVN) mechanism driver.

NOTE

The ML2/OVN mechanism driver supports only router availability zones. ML2/OVN has a distributed DHCP server, so supporting network AZs is unnecessary.

Prerequisites

- Deployed RHOSP 16.2 or later.
- Running the RHOSP Networking service that uses the ML2/OVN mechanism driver.
- When using Networking service AZs in distributed compute node (DCN) environments, you must match the Networking service AZ names to the Compute service (nova) AZ names.
  For more information, see the *Deploying a Distributed Compute Node architecture* guide.

  IMPORTANT

  Ensure that all router gateway ports reside on the OpenStack Controller nodes by setting **OVNCMSOptions: 'enable-chassis-as-gw'** and by providing one or more AZ values for the **OVNAvailabilityZone** parameter. Performing these actions prevent the routers from scheduling all chassis as potential hosts for the router gateway ports.

Procedure

1. Log in to the undercloud as the stack user, and source the **stackrc** file to enable the director command line tools.

   Example

   ```
   $ source ~/stackrc
   ```

2. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-neutron-environment.yaml
   ```

   TIP

   The Red Hat OpenStack Platform Orchestration service (heat) uses a set of plans called *templates* to install and configure your environment. You can customize aspects of the overcloud with a *custom environment file* , which is a special type of template that provides customization for your heat templates.

3. In the YAML environment file, under **parameter_defaults**, enter the **NeutronDefaultAvailabilityZones** parameter and one or more AZs.

   IMPORTANT

   In DCN environments, you must match the Networking service AZ names with Compute service AZ names.

   The Networking service assigns these AZs if a user fails to specify an AZ with the **--availability-zone-hint** option when creating a network or a router.

   Example

   ```
   parameter_defaults:
     NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
   ```

4. Determine the AZs for the gateway nodes (Controllers and Network nodes), by entering values for the parameter, **OVNAvailabilityZone**.

   IMPORTANT

   The **OVNAvailabilityZone** parameter replaces the use of AZ values in the **OVNCMSOptions** parameter. If you use the **OVNAvailabilityZone** parameter, ensure that there are no AZ values in the **OVNCMSOptions** parameter.

   Example

   In this example, roles have been predefined for Controllers for the **az-central** AZ, and Networkers for the **datacenter1** and **datacenter2** AZs:

   ```
   parameter_defaults:
     NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
   ```

```
  ControllerCentralParameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NetworkerDatacenter1Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter1'
  NetworkerDatacenter2Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter2'
```

IMPORTANT

In DCN environments, define a single AZ for **ControllerCentralParameter** so that ports are scheduled in the AZ relevant to the particular edge site.

5. By default, the router scheduler is **AZLeastRoutersScheduler**. If you want to change this, enter the new scheduler with the **NeutronRouterSchedulerDriver** parameters.

Example

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  ControllerCentralParameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NetworkerDCN1Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter1'
  NetworkerDCN2Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter2'
  NeutronRouterSchedulerDriver:
'neutron.scheduler.l3_agent_scheduler.AZLeastRoutersScheduler'
```

6. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <your-environment-files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
my-neutron-environment.yaml
```

Verification

- Confirm that availability zones are properly defined, by running the **availability zone list** command.

Example

```
$ openstack availability zone list
```

Sample output

```
+----------------+-------------+
| Zone Name      | Zone Status |
+----------------+-------------+
| az-central     | available   |
| az-datacenter1 | available   |
| az-datacenter2 | available   |
+----------------+-------------+
```

Additional resources

- About Networking service availability zones
- Configuring Network service availability zones with ML2/OVS
- Manually Assigning availability zones to networks and routers

23.4. MANUALLY ASSIGNING AVAILABILITY ZONES TO NETWORKS AND ROUTERS

You can manually assign a Red Hat OpenStack Platform (RHOSP) Networking service (neutron) availability zone (AZ) when you create a RHOSP network or a router. AZs enable you to make your RHOSP network resources highly available. You can group network nodes that are attached to different power sources on different AZs, and then schedule nodes running crucial services to be on separate AZs.

> **NOTE**
>
> If you fail to assign an AZ when creating a network or a router, the RHOSP Networking service automatically assigns to the resource the value that was specified to the RHOSP Orchestration service (heat) parameter. If no value is defined for **NeutronDefaultAvailabilityZones** the resources are scheduled without any AZ attributes.
>
> For RHOSP Networking service agents that use the Modular Layer 2 plug-in with the Open vSwitch (ML2/OVS) mechanism driver, if no AZ hint is supplied and no value specified for **NeutronDefaultAvailabilityZones**, then the Compute service (nova) AZ value is used to schedule the agent.

Prerequisites

- Deployed RHOSP 16.2 or later.
- Running the RHOSP Networking service that uses either the ML2/OVS or ML2/OVN (Open Virtual Network) mechanism drivers.

Procedure

- When you create a network on the overcloud using the OpenStack client, use the **--availability-zone-hint** option.

  NOTE

  The ML2/OVN mechanism driver supports only router availability zones. ML2/OVN has a distributed DHCP server, so supporting network AZs is unnecessary.

  In the following example, a network (**net1**) is created and assigned to either AZ **zone-1** or **zone-2**:

  ### Network example

  ```
  $ openstack network create --availability-zone-hint zone-1 \
  --availability-zone-hint zone-2 net1
  ```

  ### Sample output

  ```
  +---------------------------+--------------------------------------+
  | Field                     | Value                                |
  +---------------------------+--------------------------------------+
  | admin_state_up            | UP                                   |
  | availability_zone_hints   | zone-1                               |
  |                           | zone-2                               |
  | availability_zones        |                                      |
  | created_at                | 2021-07-31T22:14:12Z                 |
  | description               |                                      |
  | headers                   |                                      |
  | id                        | ad88e059-e7fa-4cf7-8857-6731a2a3a554 |
  | ipv4_address_scope        | None                                 |
  | ipv6_address_scope        | None                                 |
  | mtu                       | 1450                                 |
  | name                      | net1                                 |
  | port_security_enabled     | True                                 |
  | project_id                | cfd1889ac7d64ad891d4f20aef9f8d7c     |
  | provider:network_type     | vxlan                                |
  | provider:physical_network | None                                 |
  | provider:segmentation_id  | 77                                   |
  | revision_number           | 3                                    |
  | router:external           | Internal                             |
  | shared                    | False                                |
  | status                    | ACTIVE                               |
  | subnets                   |                                      |
  | tags                      | []                                   |
  | updated_at                | 2021-07-31T22:14:13Z                 |
  +---------------------------+--------------------------------------+
  ```

- When you create a router on the overcloud using the OpenStack client, use the **--ha** and **--availability-zone-hint** options.
  In the following example, a router (**router1**) is created and assigned to either AZ **zone-1** or **zone-2**:

  ### Router example

```
$ openstack router create --ha --availability-zone-hint zone-1 \
--availability-zone-hint zone-2 router1
```

Sample output

```
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints | zone-1                               |
|                         | zone-2                               |
| availability_zones      |                                      |
| created_at              | 2021-07-31T22:16:54Z                 |
| description             |                                      |
| distributed             | False                                |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| headers                 |                                      |
| id                      | ced10262-6cfe-47c1-8847-cd64276a868c |
| name                    | router1                              |
| project_id              | cfd1889ac7d64ad891d4f20aef9f8d7c     |
| revision_number         | 3                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    | []                                   |
| updated_at              | 2021-07-31T22:16:56Z                 |
+-------------------------+--------------------------------------+
```

Notice that the actual AZ is not assigned at the time that you create the network resource. The RHOSP Networking service assigns the AZ when it schedules the resource.

Verification

- Enter the appropriate OpenStack client **show** command to confirm in which zone the resource is hosted.

Example

```
$ openstack network show net1
```

Sample output

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   | zone-1                               |
|                           | zone-2                               |
| availability_zones        | zone-1                               |
|                           | zone-2                               |
| created_at                | 2021-07-31T22:14:12Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | ad88e059-e7fa-4cf7-8857-6731a2a3a554 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | net1                                 |
| port_security_enabled     | True                                 |
| project_id                | cfd1889ac7d64ad891d4f20aef9f8d7c     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 77                                   |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2021-07-31T22:14:13Z                 |
+---------------------------+--------------------------------------+
```

Additional resources

- About Networking service availability zones
- Configuring Network service availability zones with ML2/OVS
- Configuring Network service availability zones with ML2/OVN