Neutron Documentation

Release 17.4.2.dev107

Neutron development team

Nov 14, 2023

CONTENTS

Neutron is an OpenStack project to provide network connectivity as a service between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova). It implements the OpenStack Networking API.

This documentation is generated by the Sphinx toolkit and lives in the source tree. Additional documentation on Neutron and other components of OpenStack can be found on the OpenStack wiki and the *Neutron section of the wiki*. The Neutron Development wiki is also a good resource for new contributors.

Enjoy!

CHAPTER

ONE

OVERVIEW

The OpenStack project is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack provides an Infrastructure-as-a-Service (IaaS) solution through a variety of complementary services. Each service offers an Application Programming Interface (API) that facilitates this integration.

This guide covers step-by-step deployment of the major OpenStack services using a functional example architecture suitable for new users of OpenStack with sufficient Linux experience. This guide is not intended to be used for production system installations, but to create a minimum proof-of-concept for the purpose of learning about OpenStack.

After becoming familiar with basic installation, configuration, operation, and troubleshooting of these OpenStack services, you should consider the following steps toward deployment using a production architecture:

- Determine and implement the necessary core and optional services to meet performance and redundancy requirements.
- Increase security using methods such as firewalls, encryption, and service policies.
- Implement a deployment tool such as Ansible, Chef, Puppet, or Salt to automate deployment and management of the production environment.

1.1 Example architecture

The example architecture requires at least two nodes (hosts) to launch a basic virtual machine (VM) or instance. Optional services such as Block Storage and Object Storage require additional nodes.

Important: The example architecture used in this guide is a minimum configuration, and is not intended for production system installations. It is designed to provide a minimum proof-of-concept for the purpose of learning about OpenStack. For information on creating architectures for specific use cases, or how to determine which architecture is required, see the Architecture Design Guide.

This example architecture differs from a minimal production architecture as follows:

- Networking agents reside on the controller node instead of one or more dedicated network nodes.
- Overlay (tunnel) traffic for self-service networks traverses the management network instead of a dedicated network.

For more information on production architectures, see the Architecture Design Guide, OpenStack Operations Guide, and *OpenStack Networking Guide*.

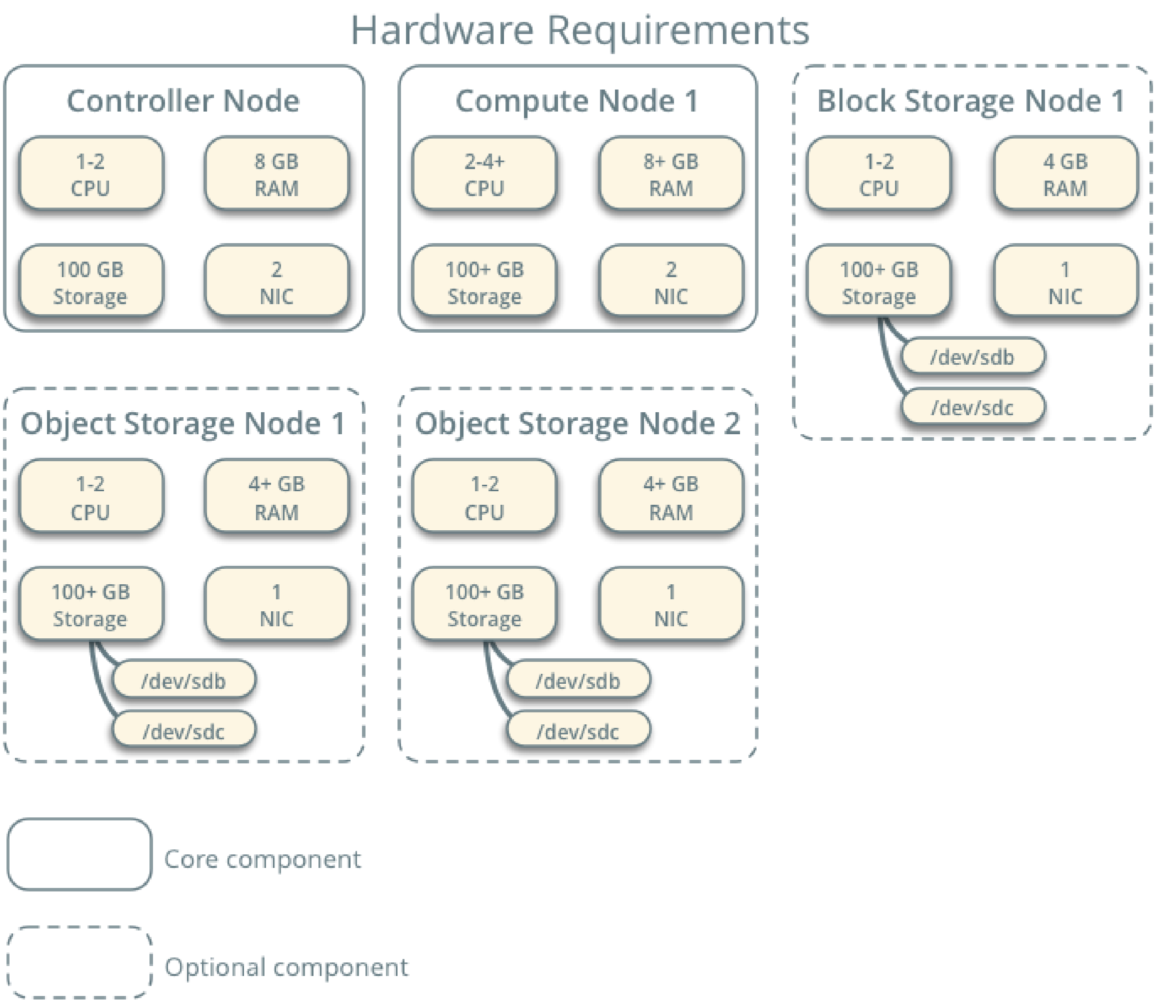


Fig. 1: **Hardware requirements**

1.1.1 Controller

The controller node runs the Identity service, Image service, management portions of Compute, management portion of Networking, various Networking agents, and the Dashboard. It also includes supporting services such as an SQL database, message queue, and Network Time Protocol (NTP).

Optionally, the controller node runs portions of the Block Storage, Object Storage, Orchestration, and Telemetry services.

The controller node requires a minimum of two network interfaces.

1.1.2 Compute

The compute node runs the hypervisor portion of Compute that operates instances. By default, Compute uses the kernel-based VM (KVM) hypervisor. The compute node also runs a Networking service agent that connects instances to virtual networks and provides firewalling services to instances via security groups.

You can deploy more than one compute node. Each node requires a minimum of two network interfaces.

1.1.3 Block Storage

The optional Block Storage node contains the disks that the Block Storage and Shared File System services provision for instances.

For simplicity, service traffic between compute nodes and this node uses the management network. Production environments should implement a separate storage network to increase performance and security.

You can deploy more than one block storage node. Each node requires a minimum of one network interface.

1.1.4 Object Storage

The optional Object Storage node contain the disks that the Object Storage service uses for storing accounts, containers, and objects.

For simplicity, service traffic between compute nodes and this node uses the management network. Production environments should implement a separate storage network to increase performance and security.

This service requires two nodes. Each node requires a minimum of one network interface. You can deploy more than two object storage nodes.

1.2 Networking

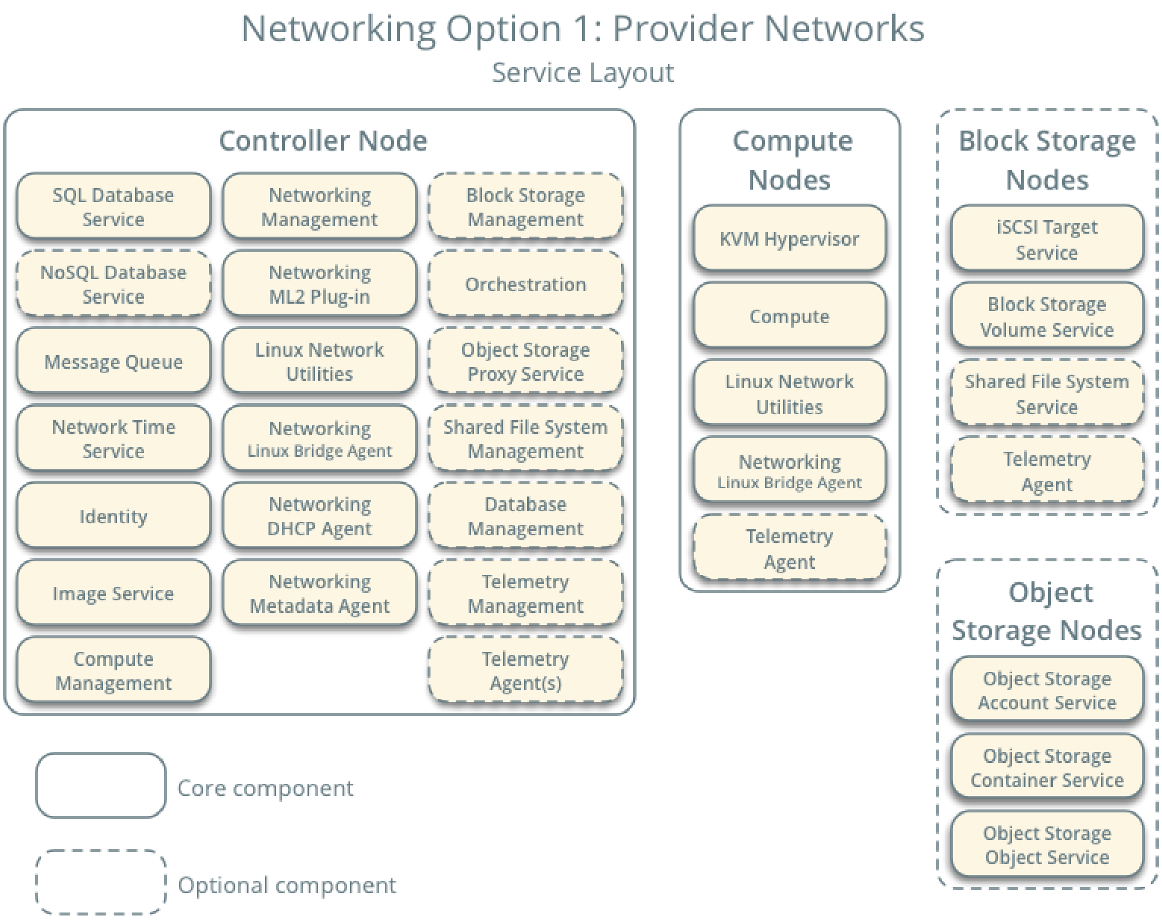
Choose one of the following virtual networking options.

1.2.1 Networking Option 1: Provider networks

The provider networks option deploys the OpenStack Networking service in the simplest way possible with primarily layer-2 (bridging/switching) services and VLAN segmentation of networks. Essentially, it bridges virtual networks to physical networks and relies on physical network infrastructure for layer-3 (routing) services. Additionally, a DHCP<Dynamic Host Configuration Protocol (DHCP) service provides IP address information to instances.

The OpenStack user requires more information about the underlying network infrastructure to create a virtual network to exactly match the infrastructure.

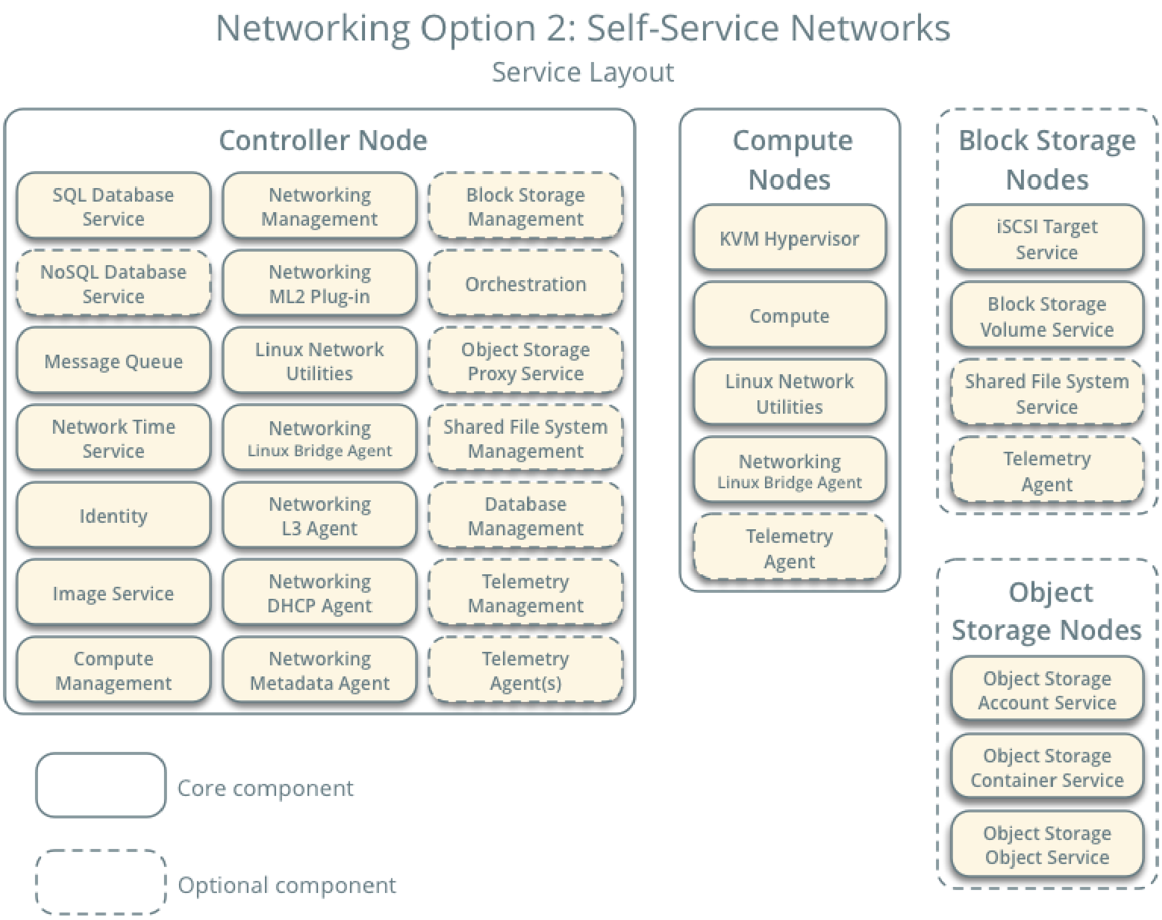
> **Warning:** This option lacks support for self-service (private) networks, layer-3 (routing) services, and advanced services such as FireWall-as-a-Service (FWaaS). Consider the self-service networks option below if you desire these features.



1.2.2 Networking Option 2: Self-service networks

The self-service networks option augments the provider networks option with layer-3 (routing) services that enable self-service networks using overlay segmentation methods such as Virtual Extensible LAN (VXLAN). Essentially, it routes virtual networks to physical networks using Network Address Translation (NAT). Additionally, this option provides the foundation for advanced services such as FWaaS.

The OpenStack user can create virtual networks without the knowledge of underlying infrastructure on the data network. This can also include VLAN networks if the layer-2 plug-in is configured accordingly.


Networking Option 2: Self-Service Networks
Service Layout
Controller Node
SQL Database Service
Networking Management
Block Storage Management
NoSQL Database Service
Networking ML2 Plug-in
Orchestration
Message Queue
Linux Network Utilities
Object Storage Proxy Service
Network Time Service
Networking Linux Bridge Agent
Shared File System Management
Identity
Networking L3 Agent
Database Management
Image Service
Networking DHCP Agent
Telemetry Management
Compute Management
Networking Metadata Agent
Telemetry Agent(s)
Compute Nodes
KVM Hypervisor
Compute
Linux Network Utilities
Networking Linux Bridge Agent
Telemetry Agent
Block Storage Nodes
iSCSI Target Service
Block Storage Volume Service
Shared File System Service
Telemetry Agent
Object Storage Nodes
Object Storage Account Service
Object Storage Container Service
Object Storage Object Service
Core component
Optional component

CHAPTER

TWO

NETWORKING SERVICE OVERVIEW

OpenStack Networking (neutron) allows you to create and attach interface devices managed by other OpenStack services to networks. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

It includes the following components:

neutron-server Accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.

OpenStack Networking plug-ins and agents Plug and unplug ports, create networks or subnets, and provide IP addressing. These plug-ins and agents differ depending on the vendor and technologies used in the particular cloud. OpenStack Networking ships with plug-ins and agents for Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, and the VMware NSX product.

The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent.

Messaging queue Used by most OpenStack Networking installations to route information between the neutron-server and various agents. Also acts as a database to store networking state for particular plug-ins.

OpenStack Networking mainly interacts with OpenStack Compute to provide networks and connectivity for its instances.

CHAPTER

THREE

NETWORKING (NEUTRON) CONCEPTS

OpenStack Networking (neutron) manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in your OpenStack environment. OpenStack Networking enables projects to create advanced virtual network topologies which may include services such as a firewall, and a virtual private network (VPN).

Networking provides networks, subnets, and routers as object abstractions. Each abstraction has functionality that mimics its physical counterpart: networks contain subnets, and routers route traffic between different subnets and networks.

Any given Networking set up has at least one external network. Unlike the other networks, the external network is not merely a virtually defined network. Instead, it represents a view into a slice of the physical, external network accessible outside the OpenStack installation. IP addresses on the external network are accessible by anybody physically on the outside network.

In addition to external networks, any Networking set up has one or more internal networks. These software-defined networks connect directly to the VMs. Only the VMs on any given internal network, or those on subnets connected through interfaces to a similar router, can access VMs connected to that network directly.

For the outside network to access VMs, and vice versa, routers between the networks are needed. Each router has one gateway that is connected to an external network and one or more interfaces connected to internal networks. Like a physical router, subnets can access machines on other subnets that are connected to the same router, and machines can access the outside network through the gateway for the router.

Additionally, you can allocate IP addresses on external networks to ports on the internal network. Whenever something is connected to a subnet, that connection is called a port. You can associate external network IP addresses with ports to VMs. This way, entities on the outside network can access VMs.

Networking also supports *security groups*. Security groups enable administrators to define firewall rules in groups. A VM can belong to one or more security groups, and Networking applies the rules in those security groups to block or unblock ports, port ranges, or traffic types for that VM.

Each plug-in that Networking uses has its own concepts. While not vital to operating the VNI and OpenStack environment, understanding these concepts can help you set up Networking. All Networking installations use a core plug-in and a security group plug-in (or just the No-Op security group plug-in). Additionally, Firewall-as-a-Service (FWaaS) is available.

CHAPTER

FOUR

INSTALL AND CONFIGURE FOR OPENSUSE AND SUSE LINUX ENTERPRISE

4.1 Host networking

After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. For more information on how to configure networking on your distribution, see the SLES 12 or openSUSE documentation.

All nodes require Internet access for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP). In most cases, nodes should obtain Internet access through the management network interface. To highlight the importance of network separation, the example architectures use private address space for the management network and assume that the physical network infrastructure provides Internet access via Network Address Translation (NAT) or other methods. The example architectures use routable IP address space for the provider (external) network and assume that the physical network infrastructure provides direct Internet access.

In the provider networks architecture, all instances attach directly to the provider network. In the self-service (private) networks architecture, instances can attach to a self-service or provider network. Self-service networks can reside entirely within OpenStack or provide some level of external network access using Network Address Translation (NAT) through the provider network.

The example architectures assume use of the following networks:

- Management on 10.0.0.0/24 with gateway 10.0.0.1

  This network requires a gateway to provide Internet access to all nodes for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP).

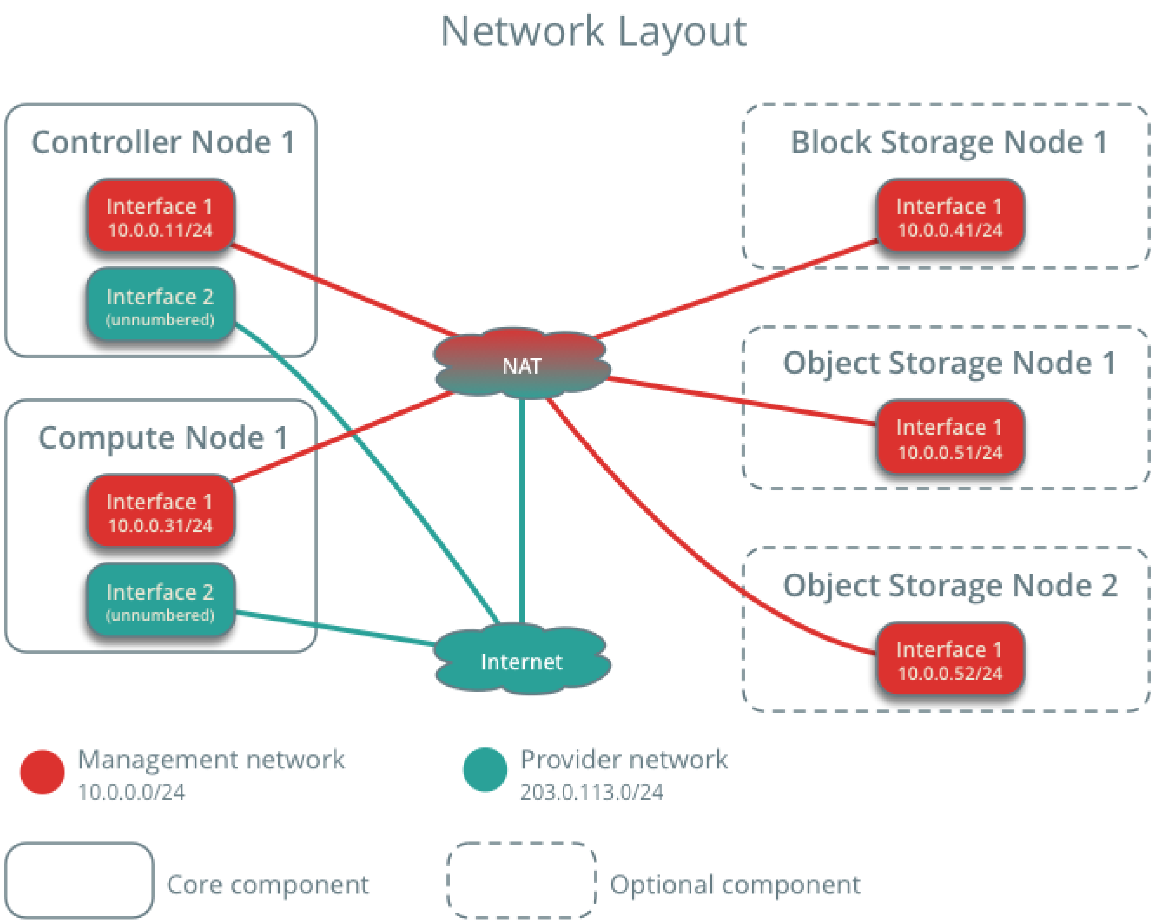
- Provider on 203.0.113.0/24 with gateway 203.0.113.1

  This network requires a gateway to provide Internet access to instances in your OpenStack environment.

You can modify these ranges and gateways to work with your particular network infrastructure.

Network interface names vary by distribution. Traditionally, interfaces use `eth` followed by a sequential number. To cover all variations, this guide refers to the first interface as the interface with the lowest number and the second interface as the interface with the highest number.

Unless you intend to use the exact configuration provided in this example architecture, you must modify the networks in this procedure to match your environment. Each node must resolve the other nodes by


Network Layout
Controller Node 1
Interface 1
10.0.0.11/24
Interface 2
(unnumbered)
Compute Node 1
Interface 1
10.0.0.31/24
Interface 2
(unnumbered)
NAT
Internet
Block Storage Node 1
Interface 1
10.0.0.41/24
Object Storage Node 1
Interface 1
10.0.0.51/24
Object Storage Node 2
Interface 1
10.0.0.52/24
Management network
10.0.0.0/24
Provider network
203.0.113.0/24
Core component
Optional component

name in addition to IP address. For example, the `controller` name must resolve to `10.0.0.11`, the IP address of the management interface on the controller node.

> **Warning:** Reconfiguring network interfaces will interrupt network connectivity. We recommend using a local terminal session for these procedures.

Note: Your distribution enables a restrictive firewall by default. During the installation process, certain steps will fail unless you alter or disable the firewall. For more information about securing your environment, refer to the OpenStack Security Guide.

4.1.1 Controller node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.11

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace `INTERFACE_NAME` with the actual interface name. For example, *eth1* or *ens224*.

- Edit the `/etc/sysconfig/network/ifcfg-INTERFACE_NAME` file to contain the following:

```
STARTMODE='auto'
BOOTPROTO='static'
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `controller`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
```

(continues on next page)

(continued from previous page)

```
10.0.0.51       object1

# object2
10.0.0.52       object2
```

Warning: Some distributions add an extraneous entry in the /etc/hosts file that resolves the actual hostname to another loopback IP address such as 127.0.1.1. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

Note: This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

4.1.2 Compute node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.31

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

   Note: Additional compute nodes should use 10.0.0.32, 10.0.0.33, and so on.

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace INTERFACE_NAME with the actual interface name. For example, *eth1* or *ens224*.

- Edit the /etc/sysconfig/network/ifcfg-INTERFACE_NAME file to contain the following:

```
STARTMODE='auto'
BOOTPROTO='static'
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `compute1`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

# object2
10.0.0.52       object2
```

> **Warning:** Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

> **Note:** This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

4.1.3 Block storage node (Optional)

If you want to deploy the Block Storage service, configure one additional storage node.

Configure network interfaces

- Configure the management interface:
  - IP address: `10.0.0.41`
  - Network mask: `255.255.255.0` (or `/24`)
  - Default gateway: `10.0.0.1`

Configure name resolution

1. Set the hostname of the node to `block1`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

# object2
10.0.0.52       object2
```

> **Warning:** Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

> **Note:** This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

3. Reboot the system to activate the changes.

4.1.4 Verify connectivity

We recommend that you verify network connectivity to the Internet and among the nodes before proceeding further.

1. From the *controller* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. From the *controller* node, test access to the management interface on the *compute* node:

```
# ping -c 4 compute1

PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- compute1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. From the *compute* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

4. From the *compute* node, test access to the management interface on the *controller* node:

```
# ping -c 4 controller

PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

Note: Your distribution enables a restrictive firewall by default. During the installation process, certain steps will fail unless you alter or disable the firewall. For more information about securing your environment, refer to the OpenStack Security Guide.

4.2 Install and configure controller node

4.2.1 Prerequisites

Before you configure the OpenStack Networking (neutron) service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

   ```
   $ mysql -u root -p
   ```

   - Create the `neutron` database:

   ```
   MariaDB [(none)] CREATE DATABASE neutron;
   ```

   - Grant proper access to the `neutron` database, replacing `NEUTRON_DBPASS` with a suitable password:

   ```
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@
   ↪'localhost' \
     IDENTIFIED BY 'NEUTRON_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'
   ↪%' \
     IDENTIFIED BY 'NEUTRON_DBPASS';
   ```

   - Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the service credentials, complete these steps:

   - Create the `neutron` user:

   ```
   $ openstack user create --domain default --password-prompt neutron

   User Password:
   Repeat User Password:
   +---------------------+----------------------------------+
   | Field               | Value                            |
   +---------------------+----------------------------------+
   | domain_id           | default                          |
   | enabled             | True                             |
   | id                  | fdb0f541e28141719b6a43c8944bf1fb |
   | name                | neutron                          |
   | options             | {}                               |
   | password_expires_at | None                             |
   +---------------------+----------------------------------+
   ```

   - Add the `admin` role to the `neutron` user:

   ```
   $ openstack role add --project service --user neutron admin
   ```

Note: This command provides no output.

- Create the `neutron` service entity:

```
$ openstack service create --name neutron \
  --description "OpenStack Networking" network

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | f71529314dab4a4d8eca427e701d209e |
| name        | neutron                          |
| type        | network                          |
+-------------+----------------------------------+
```

4. Create the Networking service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  network public http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 85d80a6d02fc4b7683f611d7fc1493a3 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network internal http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 09753b537ac74422a68d2d791cf3714f |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network admin http://controller:9696
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 1ee14289c9374dffb5db92a5c112fc4e |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+
```

4.2.2 Configure networking options

You can deploy the Networking service using one of two architectures represented by options 1 and 2.

Option 1 deploys the simplest possible architecture that only supports attaching instances to provider (external) networks. No self-service (private) networks, routers, or floating IP addresses. Only the `admin` or other privileged user can manage provider networks.

Option 2 augments option 1 with layer-3 services that support attaching instances to self-service networks. The `demo` or other unprivileged user can manage self-service networks including routers that provide connectivity between self-service and provider networks. Additionally, floating IP addresses provide connectivity to instances using self-service networks from external networks such as the Internet.

Self-service networks typically use overlay networks. Overlay network protocols such as VXLAN include additional headers that increase overhead and decrease space available for the payload or user data. Without knowledge of the virtual network infrastructure, instances attempt to send packets using the default Ethernet maximum transmission unit (MTU) of 1500 bytes. The Networking service automatically provides the correct MTU value to instances via DHCP. However, some cloud images do not use DHCP or ignore the DHCP MTU option and require configuration using metadata or a script.

Note: Option 2 also supports attaching instances to provider networks.

Choose one of the following networking options to configure services specific to it. Afterwards, return here and proceed to *Configure the metadata agent*.

Networking Option 1: Provider networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# zypper install --no-recommends openstack-neutron \
  openstack-neutron-server openstack-neutron-linuxbridge-agent \
  openstack-neutron-dhcp-agent openstack-neutron-metadata-agent \
  bridge-utils
```

Configure the server component

The Networking server component configuration includes the database, authentication mechanism, message queue, topology change notifications, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace `NEUTRON_DBPASS` with the password you chose for the database.

    Note: Comment out or remove any other `connection` options in the `[database]` section.

  - In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in and disable additional plug-ins:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins =
    ```

  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] and [nova] sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file and complete the following actions:
  - In the `[ml2]` section, enable flat and VLAN networks:

    ```
    [ml2]
    # ...
    type_drivers = flat,vlan
    ```

  - In the `[ml2]` section, disable self-service networks:

    ```
    [ml2]
    # ...
    tenant_network_types =
    ```

  - In the `[ml2]` section, enable the Linux bridge mechanism:

    ```
    [ml2]
    # ...
    mechanism_drivers = linuxbridge
    ```

    Warning: After you configure the ML2 plug-in, removing values in the `type_drivers` option can lead to database inconsistency.

  - In the `[ml2]` section, enable the port security extension driver:

    ```
    [ml2]
    # ...
    extension_drivers = port_security
    ```

  - In the `[ml2_type_flat]` section, configure the provider virtual network as a flat network:

    ```
    [ml2_type_flat]
    # ...
    flat_networks = provider
    ```

  - In the `[securitygroup]` section, enable ipset to increase efficiency of security group rules:

    ```
    [securitygroup]
    # ...
    enable_ipset = true
    ```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the `/etc/neutron/plugins/ml2/linuxbridge_agent.ini` file and complete the following actions:
  - In the `[linux_bridge]` section, map the provider virtual network to the provider physical network interface:

    ```
    [linux_bridge]
    physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
    ```

    Replace `PROVIDER_INTERFACE_NAME` with the name of the underlying provider physical network interface. See *Host networking* for more information.
  - In the `[vxlan]` section, disable VXLAN overlay networks:

    ```
    [vxlan]
    enable_vxlan = false
    ```
  - In the `[securitygroup]` section, enable security groups and configure the Linux bridge iptables firewall driver:

    ```
    [securitygroup]
    # ...
    enable_security_group = true
    firewall_driver = neutron.agent.linux.iptables_firewall.
    ↪IptablesFirewallDriver
    ```
  - Ensure your Linux operating system kernel supports network bridge filters by verifying all the following `sysctl` values are set to `1`:

    ```
    net.bridge.bridge-nf-call-iptables
    net.bridge.bridge-nf-call-ip6tables
    ```

    To enable networking bridge support, typically the `br_netfilter` kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the `/etc/neutron/dhcp_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

    ```
    [DEFAULT]
    # ...
    interface_driver = linuxbridge
    ```

(continues on next page)

(continued from previous page)

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Create the provider network

Follow this provider network document from the General Installation Guide.

Return to *Networking controller node configuration*.

Networking Option 2: Self-service networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# zypper install --no-recommends openstack-neutron \
  openstack-neutron-server openstack-neutron-linuxbridge-agent \
  openstack-neutron-l3-agent openstack-neutron-dhcp-agent \
  openstack-neutron-metadata-agent bridge-utils
```

Configure the server component

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace `NEUTRON_DBPASS` with the password you chose for the database.

    Note: Comment out or remove any other `connection` options in the `[database]` section.

  - In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins = router
    allow_overlapping_ips = true
    ```

  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
# ...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.

– In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

– In the `[DEFAULT]` and `[nova]` sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace `NOVA_PASS` with the password you chose for the `nova` user in the Identity service.

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file and complete the following actions:
  - In the `[ml2]` section, enable flat, VLAN, and VXLAN networks:

    ```
    [ml2]
    # ...
    type_drivers = flat,vlan,vxlan
    ```

  - In the `[ml2]` section, enable VXLAN self-service networks:

    ```
    [ml2]
    # ...
    tenant_network_types = vxlan
    ```

  - In the `[ml2]` section, enable the Linux bridge and layer-2 population mechanisms:

    ```
    [ml2]
    # ...
    mechanism_drivers = linuxbridge,l2population
    ```

    Warning: After you configure the ML2 plug-in, removing values in the `type_drivers` option can lead to database inconsistency.

    Note: The Linux bridge agent only supports VXLAN overlay networks.

  - In the `[ml2]` section, enable the port security extension driver:

    ```
    [ml2]
    # ...
    extension_drivers = port_security
    ```

  - In the `[ml2_type_flat]` section, configure the provider virtual network as a flat network:

    ```
    [ml2_type_flat]
    # ...
    flat_networks = provider
    ```

  - In the `[ml2_type_vxlan]` section, configure the VXLAN network identifier range for self-service networks:

```
[ml2_type_vxlan]
# ...
vni_ranges = 1:1000
```

- In the [securitygroup] section, enable ipset to increase efficiency of security group rules:

```
[securitygroup]
# ...
enable_ipset = true
```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

```
[vxlan]
enable_vxlan = true
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
l2_population = true
```

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the controller node. See *Host networking* for more information.

  - In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

  - Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the `br_netfilter` kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the layer-3 agent

The Layer-3 (L3) agent provides routing and NAT services for self-service virtual networks.

- Edit the `/etc/neutron/l3_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the Linux bridge interface driver:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
```

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the `/etc/neutron/dhcp_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Return to *Networking controller node configuration*.

4.2.3 Configure the metadata agent

The metadata agent provides configuration information such as credentials to instances.

- Edit the `/etc/neutron/metadata_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the metadata host and shared secret:

```
[DEFAULT]
# ...
nova_metadata_host = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

Replace `METADATA_SECRET` with a suitable secret for the metadata proxy.

4.2.4 Configure the Compute service to use the Networking service

Note: The Nova compute service must be installed to complete this step. For more details see the compute install guide found under the *Installation Guides* section of the docs website.

- Edit the `/etc/nova/nova.conf` file and perform the following actions:
  - In the `[neutron]` section, configure access parameters, enable the metadata proxy, and configure the secret:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET
```

    Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

    Replace `METADATA_SECRET` with the secret you chose for the metadata proxy.

    See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

4.2.5 Finalize installation

Note: SLES enables apparmor by default and restricts dnsmasq. You need to either completely disable apparmor or disable only the dnsmasq profile:

```
# ln -s /etc/apparmor.d/usr.sbin.dnsmasq /etc/apparmor.d/disable/
# systemctl restart apparmor
```

1. Restart the Compute API service:

```
# systemctl restart openstack-nova-api.service
```

2. Start the Networking services and configure them to start when the system boots.

   For both networking options:

```
# systemctl enable openstack-neutron.service \
  openstack-neutron-linuxbridge-agent.service \
  openstack-neutron-dhcp-agent.service \
  openstack-neutron-metadata-agent.service
# systemctl start openstack-neutron.service \
```

(continues on next page)

(continued from previous page)

```
openstack-neutron-linuxbridge-agent.service \
openstack-neutron-dhcp-agent.service \
openstack-neutron-metadata-agent.service
```

For networking option 2, also enable and start the layer-3 service:

```
# systemctl enable openstack-neutron-l3-agent.service
# systemctl start openstack-neutron-l3-agent.service
```

4.3 Install and configure compute node

The compute node handles connectivity and security groups for instances.

4.3.1 Install the components

```
# zypper install --no-recommends \
  openstack-neutron-linuxbridge-agent bridge-utils
```

4.3.2 Configure the common component

The Networking common component configuration includes the authentication mechanism, message queue, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, comment out any `connection` options because compute nodes do not directly access the database.
  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.
  - In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

    ```
    [DEFAULT]
    # ...
    auth_strategy = keystone
    ```

(continues on next page)

(continued from previous page)

```
[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

4.3.3 Configure networking options

Choose the same networking option that you chose for the controller node to configure services specific to it. Afterwards, return here and proceed to *Configure the Compute service to use the Networking service*.

Networking Option 1: Provider networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the `/etc/neutron/plugins/ml2/linuxbridge_agent.ini` file and complete the following actions:
  - In the `[linux_bridge]` section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace `PROVIDER_INTERFACE_NAME` with the name of the underlying provider physical network interface. See *Host networking* for more information.

- In the [vxlan] section, disable VXLAN overlay networks:

```
[vxlan]
enable_vxlan = false
```

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*

Networking Option 2: Self-service networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

```
[vxlan]
enable_vxlan = true
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
l2_population = true
```

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the compute node. See *Host networking* for more information.

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*.

4.3.4 Configure the Compute service to use the Networking service

- Edit the /etc/nova/nova.conf file and complete the following actions:
  - In the [neutron] section, configure access parameters:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

4.3.5 Finalize installation

1. The Networking service initialization scripts expect the variable `NEUTRON_PLUGIN_CONF` in the `/etc/sysconfig/neutron` file to reference the ML2 plug-in configuration file. Ensure that the `/etc/sysconfig/neutron` file contains the following:

```
NEUTRON_PLUGIN_CONF="/etc/neutron/plugins/ml2/ml2_conf.ini"
```

2. Restart the Compute service:

```
# systemctl restart openstack-nova-compute.service
```

3. Start the Linux Bridge agent and configure it to start when the system boots:

```
# systemctl enable openstack-neutron-linuxbridge-agent.service
# systemctl start openstack-neutron-linuxbridge-agent.service
```

4.4 Verify operation

Note: Perform these commands on the controller node.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. List loaded extensions to verify successful launch of the `neutron-server` process:

```
$ openstack extension list --network

+---------------------------+---------------------------+-------------
↪----------------+
| Name                      | Alias                     | Description␣
↪                |
+---------------------------+---------------------------+-------------
↪----------------+
| Default Subnetpools       | default-subnetpools       | Provides␣
↪ability to mark    |
|                           |                           | and use a␣
↪subnetpool as     |
|                           |                           | the default␣
↪                |
| Availability Zone         | availability_zone         | The␣
↪availability zone       |
|                           |                           | extension. ␣
↪                |
| Network Availability Zone | network_availability_zone |␣
↪Availability zone support   |
|                           |                           | for network.
↪                |
| Port Binding              | binding                   | Expose port␣
↪bindings of a   |
|                           |                           | virtual␣
↪port to external    |
```

(continues on next page)

(continued from previous page)

```
|                            |                            | application␣
↪              |
| agent                      | agent                      | The agent␣
↪management        |
|                            |                            | extension. ␣
↪              |
| Subnet Allocation          | subnet_allocation          | Enables␣
↪allocation of       |
|                            |                            | subnets␣
↪from a subnet pool |
| DHCP Agent Scheduler       | dhcp_agent_scheduler       | Schedule␣
↪networks among     |
|                            |                            | dhcp agents␣
↪              |
| Neutron external network   | external-net               | Adds␣
↪external network        |
|                            |                            | attribute␣
↪to network        |
|                            |                            | resource.  ␣
↪              |
| Neutron Service Flavors    | flavors                    | Flavor␣
↪specification for   |
|                            |                            | Neutron␣
↪advanced services   |
| Network MTU                | net-mtu                    | Provides␣
↪MTU attribute for |
|                            |                            | a network␣
↪resource.         |
| Network IP Availability    | network-ip-availability    | Provides IP␣
↪availability    |
|                            |                            | data for␣
↪each network and  |
|                            |                            | subnet.    ␣
↪              |
| Quota management support   | quotas                     | Expose␣
↪functions for       |
|                            |                            | quotas␣
↪management per      |
|                            |                            | tenant     ␣
↪              |
| Provider Network           | provider                   | Expose␣
↪mapping of virtual  |
|                            |                            | networks to␣
↪physical         |
|                            |                            | networks   ␣
↪              |
| Multi Provider Network     | multi-provider             | Expose␣
↪mapping of virtual  |
|                            |                            | networks to␣
↪multiple         |
|                            |                            | physical␣
↪networks           |
| Address scope              | address-scope              | Address␣
↪scopes extension.  |
| Subnet service types       | subnet-service-types       | Provides␣
↪ability to set     |
```

(continues on next page)

(continued from previous page)

```
|                            |                              | the subnet␣
↪service_types   |
|                            |                              | field     ␣
↪                |
| Resource timestamps        | standard-attr-timestamp      | Adds␣
↪created_at and        |
|                            |                              | updated_at␣
↪fields to all   |
|                            |                              | Neutron␣
↪resources that     |
|                            |                              | have␣
↪Neutron standard      |
|                            |                              | attributes.␣
↪                |
| Neutron Service Type       | service-type                 | API for␣
↪retrieving service |
| Management                 |                              | providers␣
↪for Neutron       |
|                            |                              | advanced␣
↪services          |
| resources: subnet,         |                              | more L2 and␣
↪L3 resources.   |
| subnetpool, port, router   |                              |           ␣
↪                |
| Neutron Extra DHCP opts    | extra_dhcp_opt               | Extra␣
↪options               |
|                            |                              |␣
↪configuration for DHCP.     |
|                            |                              | For example␣
↪PXE boot        |
|                            |                              | options to␣
↪DHCP clients    |
|                            |                              | can be␣
↪specified (e.g.      |
|                            |                              | tftp-server,
↪ server-ip-     |
|                            |                              | address,␣
↪bootfile-name)     |
| Resource revision numbers  | standard-attr-revisions      | This␣
↪extension will        |
|                            |                              | display the␣
↪revision        |
|                            |                              | number of␣
↪neutron           |
|                            |                              | resources. ␣
↪                |
| Pagination support         | pagination                   | Extension␣
↪that indicates    |
|                            |                              | that␣
↪pagination is         |
|                            |                              | enabled.   ␣
↪                |
| Sorting support            | sorting                      | Extension␣
↪that indicates    |
|                            |                              | that␣
↪sorting is enabled.   |
```

(continues on next page)

(continued from previous page)

```
| security-group            | security-group            | The␣
↪security groups        |
|                           |                           | extension. ␣
↪                 |
| RBAC Policies             | rbac-policies             | Allows␣
↪creation and         |
|                           |                           |␣
↪modification of policies   |
|                           |                           | that␣
↪control tenant access |
|                           |                           | to␣
↪resources.               |
| standard-attr-description | standard-attr-description | Extension␣
↪to add               |
|                           |                           |␣
↪descriptions to standard   |
|                           |                           | attributes ␣
↪                 |
| Port Security             | port-security             | Provides␣
↪port security        |
| Allowed Address Pairs     | allowed-address-pairs     | Provides␣
↪allowed address      |
|                           |                           | pairs      ␣
↪                 |
| project_id field enabled  | project-id                | Extension␣
↪that indicates       |
|                           |                           | that␣
↪project_id field is     |
|                           |                           | enabled.   ␣
↪                 |
+---------------------------+---------------------------+-------------
↪----------------+
```

Note: Actual output may differ slightly from this example.

You can perform further testing of your networking using the neutron-sanity-check command line client.

Use the verification section for the networking option that you chose to deploy.

4.4.1 Networking Option 1: Provider networks

- List agents to verify successful launch of the neutron agents:

```
$ openstack network agent list

+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host    ␣
↪   | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
| 0400c2f6-4d3b-44bc-89fa-99093432f3bf | Metadata agent     |␣
↪controller | None              | True  | UP    | neutron-metadata-
↪agent    |
```

(continues on next page)

(continued from previous page)

```
| 83cf853d-a2f2-450a-99d7-e9c6fc08f4c3 | DHCP agent         |␣
↪controller | nova              | True  | UP    | neutron-dhcp-agent␣
↪       |
| ec302e51-6101-43cf-9f19-88a78613cbee | Linux bridge agent | compute␣
↪   | None              | True  | UP    | neutron-linuxbridge-agent |
| fcb9bc6e-22b1-43bc-9054-272dd517d025 | Linux bridge agent |␣
↪controller | None              | True  | UP    | neutron-
↪linuxbridge-agent |
+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
```

The output should indicate three agents on the controller node and one agent on each compute node.

4.4.2 Networking Option 2: Self-service networks

- List agents to verify successful launch of the neutron agents:

```
$ openstack network agent list

+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host    ␣
↪   | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
| f49a4b81-afd6-4b3d-b923-66c8f0517099 | Metadata agent     |␣
↪controller | None              | True  | UP    | neutron-metadata-
↪agent    |
| 27eee952-a748-467b-bf71-941e89846a92 | Linux bridge agent |␣
↪controller | None              | True  | UP    | neutron-
↪linuxbridge-agent |
| 08905043-5010-4b87-bba5-aedb1956e27a | Linux bridge agent |␣
↪compute1   | None              | True  | UP    | neutron-
↪linuxbridge-agent |
| 830344ff-dc36-4956-84f4-067af667a0dc | L3 agent           |␣
↪controller | nova              | True  | UP    | neutron-l3-agent  ␣
↪       |
| dd3644c9-1a3a-435a-9282-eb306b4b0391 | DHCP agent         |␣
↪controller | nova              | True  | UP    | neutron-dhcp-agent␣
↪       |
+--------------------------------------+--------------------+---------
↪---+-------------------+-------+-------+---------------------------+
```

The output should indicate four agents on the controller node and one agent on each compute node.

CHAPTER

FIVE

INSTALL AND CONFIGURE FOR RED HAT ENTERPRISE LINUX AND CENTOS

5.1 Host networking

After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. For more information on how to configure networking on your distribution, see the documentation .

All nodes require Internet access for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP). In most cases, nodes should obtain Internet access through the management network interface. To highlight the importance of network separation, the example architectures use private address space for the management network and assume that the physical network infrastructure provides Internet access via Network Address Translation (NAT) or other methods. The example architectures use routable IP address space for the provider (external) network and assume that the physical network infrastructure provides direct Internet access.

In the provider networks architecture, all instances attach directly to the provider network. In the self-service (private) networks architecture, instances can attach to a self-service or provider network. Self-service networks can reside entirely within OpenStack or provide some level of external network access using Network Address Translation (NAT) through the provider network.

The example architectures assume use of the following networks:

- Management on 10.0.0.0/24 with gateway 10.0.0.1

  This network requires a gateway to provide Internet access to all nodes for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP).

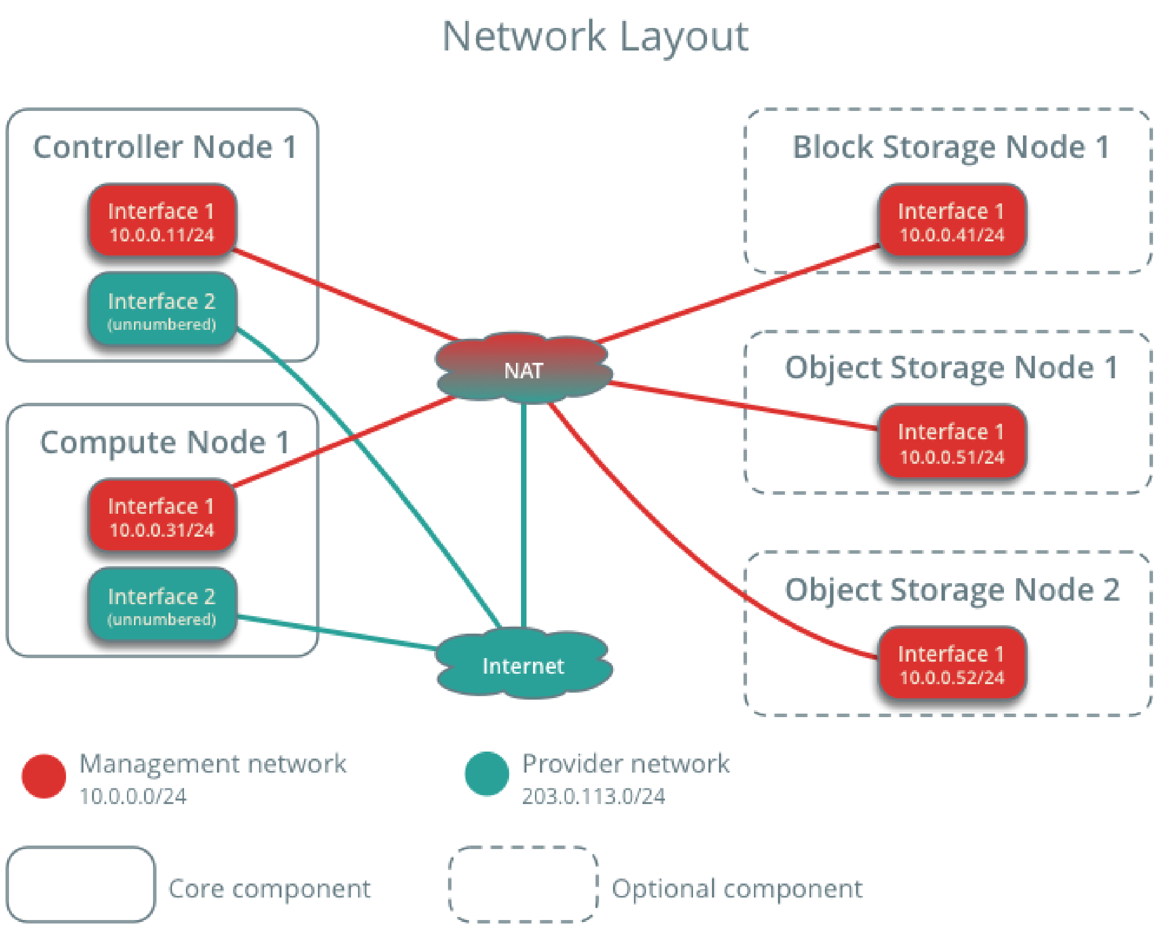
- Provider on 203.0.113.0/24 with gateway 203.0.113.1

  This network requires a gateway to provide Internet access to instances in your OpenStack environment.

You can modify these ranges and gateways to work with your particular network infrastructure.

Network interface names vary by distribution. Traditionally, interfaces use `eth` followed by a sequential number. To cover all variations, this guide refers to the first interface as the interface with the lowest number and the second interface as the interface with the highest number.

Unless you intend to use the exact configuration provided in this example architecture, you must modify the networks in this procedure to match your environment. Each node must resolve the other nodes by


Network Layout
Controller Node 1
Interface 1
10.0.0.11/24
Interface 2
(unnumbered)
Compute Node 1
Interface 1
10.0.0.31/24
Interface 2
(unnumbered)
NAT
Internet
Block Storage Node 1
Interface 1
10.0.0.41/24
Object Storage Node 1
Interface 1
10.0.0.51/24
Object Storage Node 2
Interface 1
10.0.0.52/24
Management network
10.0.0.0/24
Provider network
203.0.113.0/24
Core component
Optional component

name in addition to IP address. For example, the `controller` name must resolve to `10.0.0.11`, the IP address of the management interface on the controller node.

Warning: Reconfiguring network interfaces will interrupt network connectivity. We recommend using a local terminal session for these procedures.

Note: Your distribution enables a restrictive firewall by default. During the installation process, certain steps will fail unless you alter or disable the firewall. For more information about securing your environment, refer to the OpenStack Security Guide.

5.1.1 Controller node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.11

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace `INTERFACE_NAME` with the actual interface name. For example, *eth1* or *ens224*.

- Edit the `/etc/sysconfig/network-scripts/ifcfg-INTERFACE_NAME` file to contain the following:

  Do not change the `HWADDR` and `UUID` keys.

```
DEVICE=INTERFACE_NAME
TYPE=Ethernet
ONBOOT="yes"
BOOTPROTO="none"
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `controller`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

```

(continues on next page)

(continued from previous page)

```
# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

# object2
10.0.0.52       object2
```

Warning: Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

Note: This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

5.1.2 Compute node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.31

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

   Note: Additional compute nodes should use 10.0.0.32, 10.0.0.33, and so on.

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace `INTERFACE_NAME` with the actual interface name. For example, *eth1* or *ens224*.

- Edit the `/etc/sysconfig/network-scripts/ifcfg-INTERFACE_NAME` file to contain the following:

  Do not change the `HWADDR` and `UUID` keys.

```
DEVICE=INTERFACE_NAME
TYPE=Ethernet
ONBOOT="yes"
BOOTPROTO="none"
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `compute1`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

# object2
10.0.0.52       object2
```

> **Warning:** Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

> **Note:** This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

5.1.3 Verify connectivity

We recommend that you verify network connectivity to the Internet and among the nodes before proceeding further.

1. From the *controller* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. From the *controller* node, test access to the management interface on the *compute* node:

```
# ping -c 4 compute1

PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- compute1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. From the *compute* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

4. From the *compute* node, test access to the management interface on the *controller* node:

```
# ping -c 4 controller

PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

Note: Your distribution enables a restrictive firewall by default. During the installation process, certain steps will fail unless you alter or disable the firewall. For more information about securing your environment, refer to the OpenStack Security Guide.

5.2 Install and configure controller node

5.2.1 Prerequisites

Before you configure the OpenStack Networking (neutron) service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `neutron` database:

     ```
     MariaDB [(none)] CREATE DATABASE neutron;
     ```

   - Grant proper access to the `neutron` database, replacing `NEUTRON_DBPASS` with a suitable password:

     ```
     MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@
     ↪'localhost' \
       IDENTIFIED BY 'NEUTRON_DBPASS';
     MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'
     ↪%' \
       IDENTIFIED BY 'NEUTRON_DBPASS';
     ```

   - Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   - Create the `neutron` user:

     ```
     $ openstack user create --domain default --password-prompt neutron

     User Password:
     Repeat User Password:
     +---------------------+----------------------------------+
     | Field               | Value                            |
     +---------------------+----------------------------------+
     | domain_id           | default                          |
     | enabled             | True                             |
     | id                  | fdb0f541e28141719b6a43c8944bf1fb |
     | name                | neutron                          |
     | options             | {}                               |
     | password_expires_at | None                             |
     +---------------------+----------------------------------+
     ```

   - Add the `admin` role to the `neutron` user:

     ```
     $ openstack role add --project service --user neutron admin
     ```

Note: This command provides no output.

- Create the `neutron` service entity:

```
$ openstack service create --name neutron \
  --description "OpenStack Networking" network

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | f71529314dab4a4d8eca427e701d209e |
| name        | neutron                          |
| type        | network                          |
+-------------+----------------------------------+
```

4. Create the Networking service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  network public http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 85d80a6d02fc4b7683f611d7fc1493a3 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network internal http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 09753b537ac74422a68d2d791cf3714f |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network admin http://controller:9696
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 1ee14289c9374dffb5db92a5c112fc4e |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+
```

5.2.2 Configure networking options

You can deploy the Networking service using one of two architectures represented by options 1 and 2.

Option 1 deploys the simplest possible architecture that only supports attaching instances to provider (external) networks. No self-service (private) networks, routers, or floating IP addresses. Only the `admin` or other privileged user can manage provider networks.

Option 2 augments option 1 with layer-3 services that support attaching instances to self-service networks. The `demo` or other unprivileged user can manage self-service networks including routers that provide connectivity between self-service and provider networks. Additionally, floating IP addresses provide connectivity to instances using self-service networks from external networks such as the Internet.

Self-service networks typically use overlay networks. Overlay network protocols such as VXLAN include additional headers that increase overhead and decrease space available for the payload or user data. Without knowledge of the virtual network infrastructure, instances attempt to send packets using the default Ethernet maximum transmission unit (MTU) of 1500 bytes. The Networking service automatically provides the correct MTU value to instances via DHCP. However, some cloud images do not use DHCP or ignore the DHCP MTU option and require configuration using metadata or a script.

Note: Option 2 also supports attaching instances to provider networks.

Choose one of the following networking options to configure services specific to it. Afterwards, return here and proceed to *Configure the metadata agent*.

Networking Option 1: Provider networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# yum install openstack-neutron openstack-neutron-ml2 \
  openstack-neutron-linuxbridge ebtables
```

Configure the server component

The Networking server component configuration includes the database, authentication mechanism, message queue, topology change notifications, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace `NEUTRON_DBPASS` with the password you chose for the database.

    Note: Comment out or remove any other `connection` options in the `[database]` section.

  - In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in and disable additional plug-ins:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins =
    ```

  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] and [nova] sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file and complete the following actions:
  - In the [ml2] section, enable flat and VLAN networks:

    ```
    [ml2]
    # ...
    type_drivers = flat,vlan
    ```

  - In the [ml2] section, disable self-service networks:

    ```
    [ml2]
    # ...
    tenant_network_types =
    ```

  - In the [ml2] section, enable the Linux bridge mechanism:

    ```
    [ml2]
    # ...
    mechanism_drivers = linuxbridge
    ```

    Warning: After you configure the ML2 plug-in, removing values in the type_drivers option can lead to database inconsistency.

  - In the [ml2] section, enable the port security extension driver:

    ```
    [ml2]
    # ...
    extension_drivers = port_security
    ```

  - In the [ml2_type_flat] section, configure the provider virtual network as a flat network:

    ```
    [ml2_type_flat]
    # ...
    flat_networks = provider
    ```

  - In the [securitygroup] section, enable ipset to increase efficiency of security group rules:

    ```
    [securitygroup]
    # ...
    enable_ipset = true
    ```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

    ```
    [linux_bridge]
    physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
    ```

    Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.
  - In the [vxlan] section, disable VXLAN overlay networks:

    ```
    [vxlan]
    enable_vxlan = false
    ```

  - In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

    ```
    [securitygroup]
    # ...
    enable_security_group = true
    firewall_driver = neutron.agent.linux.iptables_firewall.
    ↪IptablesFirewallDriver
    ```

  - Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

    ```
    net.bridge.bridge-nf-call-iptables
    net.bridge.bridge-nf-call-ip6tables
    ```

    To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the /etc/neutron/dhcp_agent.ini file and complete the following actions:
  - In the [DEFAULT] section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

    ```
    [DEFAULT]
    # ...
    interface_driver = linuxbridge
    ```

(continues on next page)

(continued from previous page)

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Create the provider network

Follow this provider network document from the General Installation Guide.

Return to *Networking controller node configuration*.

Networking Option 2: Self-service networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# yum install openstack-neutron openstack-neutron-ml2 \
  openstack-neutron-linuxbridge ebtables
```

Configure the server component

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace `NEUTRON_DBPASS` with the password you chose for the database.

    Note: Comment out or remove any other `connection` options in the `[database]` section.

  - In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins = router
    allow_overlapping_ips = true
    ```

  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

– In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

– In the [DEFAULT] and [nova] sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file and complete the following actions:
  - In the [ml2] section, enable flat, VLAN, and VXLAN networks:

    ```
    [ml2]
    # ...
    type_drivers = flat,vlan,vxlan
    ```

  - In the [ml2] section, enable VXLAN self-service networks:

    ```
    [ml2]
    # ...
    tenant_network_types = vxlan
    ```

  - In the [ml2] section, enable the Linux bridge and layer-2 population mechanisms:

    ```
    [ml2]
    # ...
    mechanism_drivers = linuxbridge,l2population
    ```

    Warning: After you configure the ML2 plug-in, removing values in the type_drivers option can lead to database inconsistency.

    Note: The Linux bridge agent only supports VXLAN overlay networks.

  - In the [ml2] section, enable the port security extension driver:

    ```
    [ml2]
    # ...
    extension_drivers = port_security
    ```

  - In the [ml2_type_flat] section, configure the provider virtual network as a flat network:

    ```
    [ml2_type_flat]
    # ...
    flat_networks = provider
    ```

  - In the [ml2_type_vxlan] section, configure the VXLAN network identifier range for self-service networks:

    ```
    [ml2_type_vxlan]
    # ...
    vni_ranges = 1:1000
    ```

- In the [securitygroup] section, enable ipset to increase efficiency of security group rules:

```
[securitygroup]
# ...
enable_ipset = true
```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

    ```
    [linux_bridge]
    physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
    ```

    Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

    ```
    [vxlan]
    enable_vxlan = true
    local_ip = OVERLAY_INTERFACE_IP_ADDRESS
    l2_population = true
    ```

    Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the controller node. See *Host networking* for more information.

  - In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

    ```
    [securitygroup]
    # ...
    enable_security_group = true
    firewall_driver = neutron.agent.linux.iptables_firewall.
    ↪IptablesFirewallDriver
    ```

  - Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

    ```
    net.bridge.bridge-nf-call-iptables
    net.bridge.bridge-nf-call-ip6tables
    ```

    To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the layer-3 agent

The Layer-3 (L3) agent provides routing and NAT services for self-service virtual networks.

- Edit the `/etc/neutron/l3_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the Linux bridge interface driver:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
```

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the `/etc/neutron/dhcp_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Return to *Networking controller node configuration*.

5.2.3 Configure the metadata agent

The metadata agent provides configuration information such as credentials to instances.

- Edit the `/etc/neutron/metadata_agent.ini` file and complete the following actions:
  - In the `[DEFAULT]` section, configure the metadata host and shared secret:

```
[DEFAULT]
# ...
nova_metadata_host = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

  Replace `METADATA_SECRET` with a suitable secret for the metadata proxy.

5.2.4 Configure the Compute service to use the Networking service

Note: The Nova compute service must be installed to complete this step. For more details see the compute install guide found under the *Installation Guides* section of the docs website.

- Edit the `/etc/nova/nova.conf` file and perform the following actions:
  - In the `[neutron]` section, configure access parameters, enable the metadata proxy, and configure the secret:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET
```

    Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

    Replace `METADATA_SECRET` with the secret you chose for the metadata proxy.

    See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

5.2.5 Finalize installation

1. The Networking service initialization scripts expect a symbolic link `/etc/neutron/plugin.ini` pointing to the ML2 plug-in configuration file, `/etc/neutron/plugins/ml2/ml2_conf.ini`. If this symbolic link does not exist, create it using the following command:

```
# ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

2. Populate the database:

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/
↪neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head"␣
↪neutron
```

   Note: Database population occurs later for Networking because the script requires complete server and plug-in configuration files.

3. Restart the Compute API service:

```
# systemctl restart openstack-nova-api.service
```

4. Start the Networking services and configure them to start when the system boots.

   For both networking options:

```
# systemctl enable neutron-server.service \
  neutron-linuxbridge-agent.service neutron-dhcp-agent.service \
  neutron-metadata-agent.service
# systemctl start neutron-server.service \
  neutron-linuxbridge-agent.service neutron-dhcp-agent.service \
  neutron-metadata-agent.service
```

   For networking option 2, also enable and start the layer-3 service:

```
# systemctl enable neutron-l3-agent.service
# systemctl start neutron-l3-agent.service
```

5.3 Install and configure compute node

The compute node handles connectivity and security groups for instances.

5.3.1 Install the components

```
# yum install openstack-neutron-linuxbridge ebtables ipset
```

5.3.2 Configure the common component

The Networking common component configuration includes the authentication mechanism, message queue, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, comment out any `connection` options because compute nodes do not directly access the database.
  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
# ...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

5.3.3 Configure networking options

Choose the same networking option that you chose for the controller node to configure services specific to it. Afterwards, return here and proceed to *Configure the Compute service to use the Networking service*.

Networking Option 1: Provider networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

– In the [vxlan] section, disable VXLAN overlay networks:

```
[vxlan]
enable_vxlan = false
```

– In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

– Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*

Networking Option 2: Self-service networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:

  – In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

- In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

```
[vxlan]
enable_vxlan = true
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
l2_population = true
```

  Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the compute node. See *Host networking* for more information.

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

  To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*.

5.3.4 Configure the Compute service to use the Networking service

- Edit the /etc/nova/nova.conf file and complete the following actions:
  - In the [neutron] section, configure access parameters:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
```

    Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

5.3.5 Finalize installation

1. Restart the Compute service:

```
# systemctl restart openstack-nova-compute.service
```

2. Start the Linux bridge agent and configure it to start when the system boots:

```
# systemctl enable neutron-linuxbridge-agent.service
# systemctl start neutron-linuxbridge-agent.service
```

CHAPTER

SIX

INSTALL AND CONFIGURE FOR UBUNTU

6.1 Host networking

After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. For more information on how to configure networking on your distribution, see the documentation.

All nodes require Internet access for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP). In most cases, nodes should obtain Internet access through the management network interface. To highlight the importance of network separation, the example architectures use private address space for the management network and assume that the physical network infrastructure provides Internet access via Network Address Translation (NAT) or other methods. The example architectures use routable IP address space for the provider (external) network and assume that the physical network infrastructure provides direct Internet access.

In the provider networks architecture, all instances attach directly to the provider network. In the self-service (private) networks architecture, instances can attach to a self-service or provider network. Self-service networks can reside entirely within OpenStack or provide some level of external network access using Network Address Translation (NAT) through the provider network.

The example architectures assume use of the following networks:

- Management on 10.0.0.0/24 with gateway 10.0.0.1

  This network requires a gateway to provide Internet access to all nodes for administrative purposes such as package installation, security updates, Domain Name System (DNS), and Network Time Protocol (NTP).

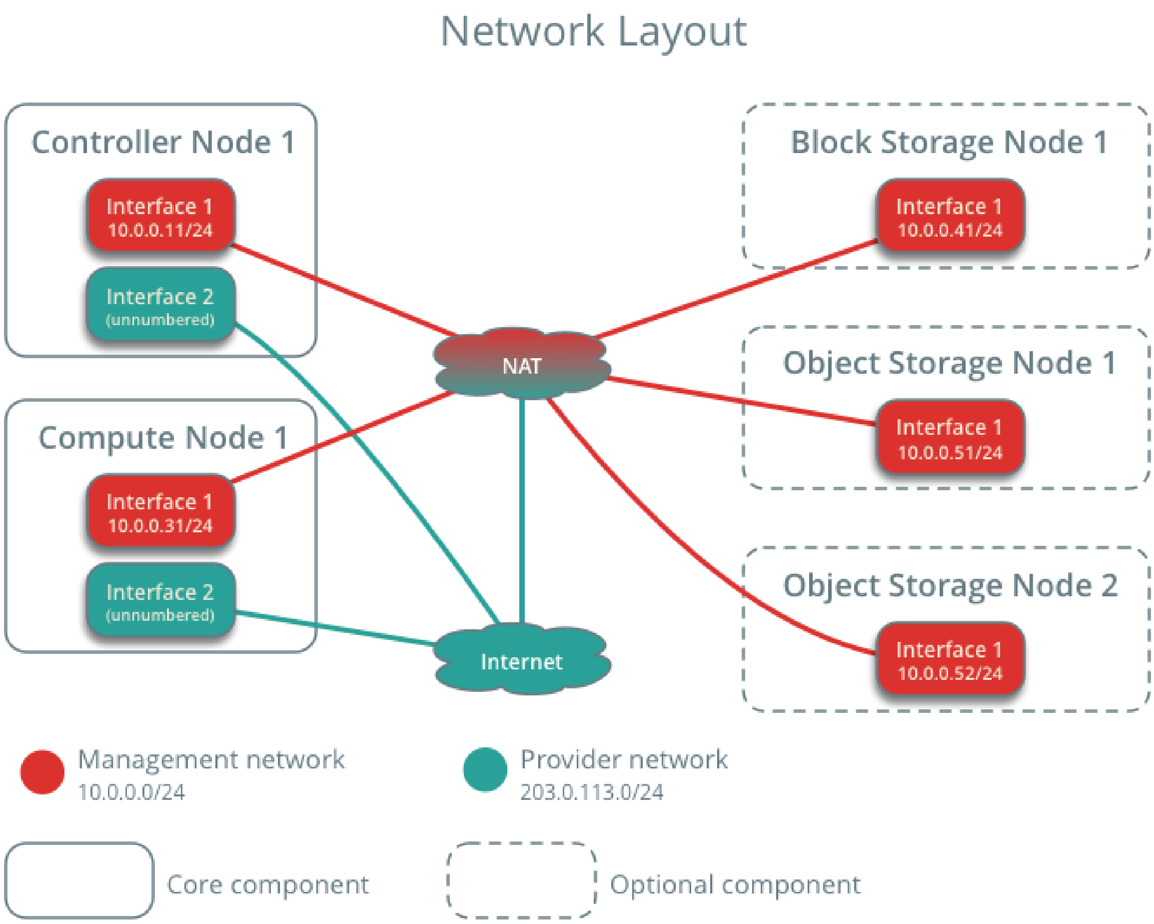
- Provider on 203.0.113.0/24 with gateway 203.0.113.1

  This network requires a gateway to provide Internet access to instances in your OpenStack environment.

You can modify these ranges and gateways to work with your particular network infrastructure.

Network interface names vary by distribution. Traditionally, interfaces use `eth` followed by a sequential number. To cover all variations, this guide refers to the first interface as the interface with the lowest number and the second interface as the interface with the highest number.

Unless you intend to use the exact configuration provided in this example architecture, you must modify the networks in this procedure to match your environment. Each node must resolve the other nodes by name in addition to IP address. For example, the `controller` name must resolve to `10.0.0.11`, the IP address of the management interface on the controller node.


Network Layout
Controller Node 1
Interface 1
10.0.0.11/24
Interface 2
(unnumbered)
Compute Node 1
Interface 1
10.0.0.31/24
Interface 2
(unnumbered)
NAT
Internet
Block Storage Node 1
Interface 1
10.0.0.41/24
Object Storage Node 1
Interface 1
10.0.0.51/24
Object Storage Node 2
Interface 1
10.0.0.52/24
Management network
10.0.0.0/24
Provider network
203.0.113.0/24
Core component
Optional component

Warning: Reconfiguring network interfaces will interrupt network connectivity. We recommend using a local terminal session for these procedures.

Note: Your distribution does not enable a restrictive firewall by default. For more information about securing your environment, refer to the OpenStack Security Guide.

6.1.1 Controller node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.11

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace `INTERFACE_NAME` with the actual interface name. For example, *eth1* or *ens224*.

- Edit the `/etc/network/interfaces` file to contain the following:

```
# The provider network interface
auto INTERFACE_NAME
iface INTERFACE_NAME inet manual
up ip link set dev $IFACE up
down ip link set dev $IFACE down
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `controller`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

```

(continues on next page)

(continued from previous page)

```
# object2
10.0.0.52       object2
```

Warning: Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

Note: This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

6.1.2 Compute node

Configure network interfaces

1. Configure the first interface as the management interface:

   IP address: 10.0.0.31

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

   Note: Additional compute nodes should use 10.0.0.32, 10.0.0.33, and so on.

2. The provider interface uses a special configuration without an IP address assigned to it. Configure the second interface as the provider interface:

   Replace `INTERFACE_NAME` with the actual interface name. For example, *eth1* or *ens224*.

- Edit the `/etc/network/interfaces` file to contain the following:

```
# The provider network interface
auto INTERFACE_NAME
iface  INTERFACE_NAME inet manual
up ip link set dev $IFACE up
down ip link set dev $IFACE down
```

1. Reboot the system to activate the changes.

Configure name resolution

1. Set the hostname of the node to `compute1`.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1

# block1
10.0.0.41       block1

# object1
10.0.0.51       object1

# object2
10.0.0.52       object2
```

Warning: Some distributions add an extraneous entry in the `/etc/hosts` file that resolves the actual hostname to another loopback IP address such as `127.0.1.1`. You must comment out or remove this entry to prevent name resolution problems. **Do not remove the 127.0.0.1 entry.**

Note: This guide includes host entries for optional services in order to reduce complexity should you choose to deploy them.

6.1.3 Verify connectivity

We recommend that you verify network connectivity to the Internet and among the nodes before proceeding further.

1. From the *controller* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. From the *controller* node, test access to the management interface on the *compute* node:

```
# ping -c 4 compute1

PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- compute1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. From the *compute* node, test access to the Internet:

```
# ping -c 4 openstack.org

PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

4. From the *compute* node, test access to the management interface on the *controller* node:

```
# ping -c 4 controller

PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

Note: Your distribution does not enable a restrictive firewall by default. For more information about securing your environment, refer to the OpenStack Security Guide.

6.2 Install and configure controller node

6.2.1 Prerequisites

Before you configure the OpenStack Networking (neutron) service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

```
$ mysql -u root -p
```

- Create the `neutron` database:

```
MariaDB [(none)] CREATE DATABASE neutron;
```

- Grant proper access to the `neutron` database, replacing `NEUTRON_DBPASS` with a suitable password:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@
↪'localhost' \
  IDENTIFIED BY 'NEUTRON_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'
↪%' \
  IDENTIFIED BY 'NEUTRON_DBPASS';
```

- Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the service credentials, complete these steps:

- Create the `neutron` user:

```
$ openstack user create --domain default --password-prompt neutron

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | fdb0f541e28141719b6a43c8944bf1fb |
| name                | neutron                          |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Add the `admin` role to the `neutron` user:

```
$ openstack role add --project service --user neutron admin
```

Note: This command provides no output.

- Create the `neutron` service entity:

```
$ openstack service create --name neutron \
  --description "OpenStack Networking" network

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

(continues on next page)

(continued from previous page)

```
+-------------+----------------------------------+
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | f71529314dab4a4d8eca427e701d209e |
| name        | neutron                          |
| type        | network                          |
+-------------+----------------------------------+
```

4. Create the Networking service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  network public http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 85d80a6d02fc4b7683f611d7fc1493a3 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network internal http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 09753b537ac74422a68d2d791cf3714f |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  network admin http://controller:9696

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 1ee14289c9374dffb5db92a5c112fc4e |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | f71529314dab4a4d8eca427e701d209e |
```

(continues on next page)

(continued from previous page)

```
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller:9696           |
+--------------+----------------------------------+
```

6.2.2 Configure networking options

You can deploy the Networking service using one of two architectures represented by options 1 and 2.

Option 1 deploys the simplest possible architecture that only supports attaching instances to provider (external) networks. No self-service (private) networks, routers, or floating IP addresses. Only the `admin` or other privileged user can manage provider networks.

Option 2 augments option 1 with layer-3 services that support attaching instances to self-service networks. The `demo` or other unprivileged user can manage self-service networks including routers that provide connectivity between self-service and provider networks. Additionally, floating IP addresses provide connectivity to instances using self-service networks from external networks such as the Internet.

Self-service networks typically use overlay networks. Overlay network protocols such as VXLAN include additional headers that increase overhead and decrease space available for the payload or user data. Without knowledge of the virtual network infrastructure, instances attempt to send packets using the default Ethernet maximum transmission unit (MTU) of 1500 bytes. The Networking service automatically provides the correct MTU value to instances via DHCP. However, some cloud images do not use DHCP or ignore the DHCP MTU option and require configuration using metadata or a script.

Note: Option 2 also supports attaching instances to provider networks.

Choose one of the following networking options to configure services specific to it. Afterwards, return here and proceed to *Configure the metadata agent*.

Networking Option 1: Provider networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# apt install neutron-server neutron-plugin-ml2 \
  neutron-linuxbridge-agent neutron-dhcp-agent \
  neutron-metadata-agent
```

Configure the server component

The Networking server component configuration includes the database, authentication mechanism, message queue, topology change notifications, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the /etc/neutron/neutron.conf file and complete the following actions:
  - In the [database] section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace NEUTRON_DBPASS with the password you chose for the database.

    Note: Comment out or remove any other connection options in the [database] section.

  - In the [DEFAULT] section, enable the Modular Layer 2 (ML2) plug-in and disable additional plug-ins:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins =
    ```

  - In the [DEFAULT] section, configure RabbitMQ message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

  - In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

    ```
    [DEFAULT]
    # ...
    auth_strategy = keystone

    [keystone_authtoken]
    # ...
    www_authenticate_uri = http://controller:5000
    auth_url = http://controller:5000
    memcached_servers = controller:11211
    auth_type = password
    ```

    (continues on next page)

(continued from previous page)

```
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] and [nova] sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file and complete the following actions:
  - In the [ml2] section, enable flat and VLAN networks:

```
[ml2]
# ...
type_drivers = flat,vlan
```

- In the [ml2] section, disable self-service networks:

```
[ml2]
# ...
tenant_network_types =
```

- In the [ml2] section, enable the Linux bridge mechanism:

```
[ml2]
# ...
mechanism_drivers = linuxbridge
```

Warning: After you configure the ML2 plug-in, removing values in the type_drivers option can lead to database inconsistency.

- In the [ml2] section, enable the port security extension driver:

```
[ml2]
# ...
extension_drivers = port_security
```

- In the [ml2_type_flat] section, configure the provider virtual network as a flat network:

```
[ml2_type_flat]
# ...
flat_networks = provider
```

- In the [securitygroup] section, enable ipset to increase efficiency of security group rules:

```
[securitygroup]
# ...
enable_ipset = true
```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

  Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the [vxlan] section, disable VXLAN overlay networks:

```
[vxlan]
enable_vxlan = false
```

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the /etc/neutron/dhcp_agent.ini file and complete the following actions:
  - In the [DEFAULT] section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Create the provider network

Follow this provider network document from the General Installation Guide.

Return to *Networking controller node configuration*.

Networking Option 2: Self-service networks

Install and configure the Networking components on the *controller* node.

Install the components

```
# apt install neutron-server neutron-plugin-ml2 \
  neutron-linuxbridge-agent neutron-l3-agent neutron-dhcp-agent \
  neutron-metadata-agent
```

Configure the server component

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

    ```
    [database]
    # ...
    connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/
    ↪neutron
    ```

    Replace `NEUTRON_DBPASS` with the password you chose for the database.

    Note: Comment out or remove any other `connection` options in the `[database]` section.

  - In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

    ```
    [DEFAULT]
    # ...
    core_plugin = ml2
    service_plugins = router
    allow_overlapping_ips = true
    ```

  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.

  - In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

    ```
    [DEFAULT]
    # ...
    auth_strategy = keystone
    ```

(continues on next page)

(continued from previous page)

```
[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

– In the `[DEFAULT]` and `[nova]` sections, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
# ...
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

Replace `NOVA_PASS` with the password you chose for the `nova` user in the Identity service.

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

- Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file and complete the following actions:
  - In the [ml2] section, enable flat, VLAN, and VXLAN networks:

```
[ml2]
# ...
type_drivers = flat,vlan,vxlan
```

  - In the [ml2] section, enable VXLAN self-service networks:

```
[ml2]
# ...
tenant_network_types = vxlan
```

  - In the [ml2] section, enable the Linux bridge and layer-2 population mechanisms:

```
[ml2]
# ...
mechanism_drivers = linuxbridge,l2population
```

Warning: After you configure the ML2 plug-in, removing values in the type_drivers option can lead to database inconsistency.

Note: The Linux bridge agent only supports VXLAN overlay networks.

  - In the [ml2] section, enable the port security extension driver:

```
[ml2]
# ...
extension_drivers = port_security
```

  - In the [ml2_type_flat] section, configure the provider virtual network as a flat network:

```
[ml2_type_flat]
# ...
flat_networks = provider
```

  - In the [ml2_type_vxlan] section, configure the VXLAN network identifier range for self-service networks:

```
[ml2_type_vxlan]
# ...
vni_ranges = 1:1000
```

– In the [securitygroup] section, enable ipset to increase efficiency of security group rules:

```
[securitygroup]
# ...
enable_ipset = true
```

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:

  – In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

  ```
  [linux_bridge]
  physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
  ```

  Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  – In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

  ```
  [vxlan]
  enable_vxlan = true
  local_ip = OVERLAY_INTERFACE_IP_ADDRESS
  l2_population = true
  ```

  Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the controller node. See *Host networking* for more information.

  – In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

  ```
  [securitygroup]
  # ...
  enable_security_group = true
  firewall_driver = neutron.agent.linux.iptables_firewall.
  ↪IptablesFirewallDriver
  ```

  – Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

  ```
  net.bridge.bridge-nf-call-iptables
  net.bridge.bridge-nf-call-ip6tables
  ```

  To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Configure the layer-3 agent

The Layer-3 (L3) agent provides routing and NAT services for self-service virtual networks.

- Edit the /etc/neutron/l3_agent.ini file and complete the following actions:
  - In the [DEFAULT] section, configure the Linux bridge interface driver:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
```

Configure the DHCP agent

The DHCP agent provides DHCP services for virtual networks.

- Edit the /etc/neutron/dhcp_agent.ini file and complete the following actions:
  - In the [DEFAULT] section, configure the Linux bridge interface driver, Dnsmasq DHCP driver, and enable isolated metadata so instances on provider networks can access metadata over the network:

```
[DEFAULT]
# ...
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Return to *Networking controller node configuration*.

6.2.3 Configure the metadata agent

The metadata agent provides configuration information such as credentials to instances.

- Edit the /etc/neutron/metadata_agent.ini file and complete the following actions:
  - In the [DEFAULT] section, configure the metadata host and shared secret:

```
[DEFAULT]
# ...
nova_metadata_host = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

    Replace METADATA_SECRET with a suitable secret for the metadata proxy.

6.2.4 Configure the Compute service to use the Networking service

Note: The Nova compute service must be installed to complete this step. For more details see the compute install guide found under the *Installation Guides* section of the docs website.

- Edit the `/etc/nova/nova.conf` file and perform the following actions:
  - In the `[neutron]` section, configure access parameters, enable the metadata proxy, and configure the secret:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET
```

Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

Replace `METADATA_SECRET` with the secret you chose for the metadata proxy.

See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

6.2.5 Finalize installation

1. Populate the database:

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/
↪neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head"␣
↪neutron
```

Note: Database population occurs later for Networking because the script requires complete server and plug-in configuration files.

2. Restart the Compute API service:

```
# service nova-api restart
```

3. Restart the Networking services.

   For both networking options:

```
# service neutron-server restart
# service neutron-linuxbridge-agent restart
# service neutron-dhcp-agent restart
# service neutron-metadata-agent restart
```

For networking option 2, also restart the layer-3 service:

```
# service neutron-l3-agent restart
```

6.3 Install and configure compute node

The compute node handles connectivity and security groups for instances.

6.3.1 Install the components

```
# apt install neutron-linuxbridge-agent
```

6.3.2 Configure the common component

The Networking common component configuration includes the authentication mechanism, message queue, and plug-in.

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, comment out any `connection` options because compute nodes do not directly access the database.
  - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

    ```
    [DEFAULT]
    # ...
    transport_url = rabbit://openstack:RABBIT_PASS@controller
    ```

    Replace `RABBIT_PASS` with the password you chose for the `openstack` account in RabbitMQ.
  - In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

    ```
    [DEFAULT]
    # ...
    auth_strategy = keystone

    [keystone_authtoken]
    # ...
    ```

(continues on next page)

(continued from previous page)

```
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace `NEUTRON_PASS` with the password you chose for the `neutron` user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

6.3.3 Configure networking options

Choose the same networking option that you chose for the controller node to configure services specific to it. Afterwards, return here and proceed to *Configure the Compute service to use the Networking service*.

Networking Option 1: Provider networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the `/etc/neutron/plugins/ml2/linuxbridge_agent.ini` file and complete the following actions:
  - In the `[linux_bridge]` section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

    Replace `PROVIDER_INTERFACE_NAME` with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the `[vxlan]` section, disable VXLAN overlay networks:

```
[vxlan]
enable_vxlan = false
```

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*

Networking Option 2: Self-service networks

Configure the Networking components on a *compute* node.

Configure the Linux bridge agent

The Linux bridge agent builds layer-2 (bridging and switching) virtual networking infrastructure for instances and handles security groups.

- Edit the /etc/neutron/plugins/ml2/linuxbridge_agent.ini file and complete the following actions:
  - In the [linux_bridge] section, map the provider virtual network to the provider physical network interface:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE_NAME
```

Replace PROVIDER_INTERFACE_NAME with the name of the underlying provider physical network interface. See *Host networking* for more information.

  - In the [vxlan] section, enable VXLAN overlay networks, configure the IP address of the physical network interface that handles overlay networks, and enable layer-2 population:

```
[vxlan]
enable_vxlan = true
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
l2_population = true
```

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the underlying physical network interface that handles overlay networks. The example architecture uses the management interface to tunnel traffic to the other nodes. Therefore, replace OVERLAY_INTERFACE_IP_ADDRESS with the management IP address of the compute node. See *Host networking* for more information.

- In the [securitygroup] section, enable security groups and configure the Linux bridge iptables firewall driver:

```
[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
↪IptablesFirewallDriver
```

- Ensure your Linux operating system kernel supports network bridge filters by verifying all the following sysctl values are set to 1:

```
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

To enable networking bridge support, typically the br_netfilter kernel module needs to be loaded. Check your operating systems documentation for additional details on enabling this module.

Return to *Networking compute node configuration*.

6.3.4 Configure the Compute service to use the Networking service

- Edit the /etc/nova/nova.conf file and complete the following actions:
  - In the [neutron] section, configure access parameters:

```
[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
```

Replace NEUTRON_PASS with the password you chose for the neutron user in the Identity service.

See the compute service configuration guide for the full set of options including overriding the service catalog endpoint URL if necessary.

6.3.5 Finalize installation

1. Restart the Compute service:

```
# service nova-compute restart
```

2. Restart the Linux bridge agent:

```
# service neutron-linuxbridge-agent restart
```

CHAPTER

SEVEN

OVN INSTALL DOCUMENTATION

7.1 Manual install & Configuration

This document discusses what is required for manual installation or integration into a production OpenStack deployment tool of conventional architectures that include the following types of nodes:

- Controller - Runs OpenStack control plane services such as REST APIs and databases.
- Network - Runs the layer-2, layer-3 (routing), DHCP, and metadata agents for the Networking service. Some agents optional. Usually provides connectivity between provider (public) and project (private) networks via NAT and floating IP addresses.

  Note: Some tools deploy these services on controller nodes.

- Compute - Runs the hypervisor and layer-2 agent for the Networking service.

7.1.1 Packaging

Open vSwitch (OVS) includes OVN beginning with version 2.5 and considers it experimental. From version 2.13 OVN has been released as separate project. The Networking service integration for OVN is now one of the in-tree Neutron drivers so should be delivered with `neutron` package, but older versions of this integration were delivered with independent package, typically `networking-ovn`.

Building OVS from source automatically installs OVN for releases older than 2.13. For newer releases it is required to build OVS and OVN separately. For deployment tools using distribution packages, the `openvswitch-ovn` package for RHEL/CentOS and compatible distributions automatically installs `openvswitch` as a dependency. Ubuntu/Debian includes `ovn-central`, `ovn-host`, `ovn-docker`, and `ovn-common` packages that pull in the appropriate Open vSwitch dependencies as needed.

A `python-networking-ovn` RPM may be obtained for Fedora or CentOS from the RDO project. Since Ussuri release OVN driver is shipped with `neutron` package. A package based on the older branch of `networking-ovn` can be found at https://trunk.rdoproject.org/.

Fedora and CentOS RPM builds of OVS and OVN from the `master` branch of `ovs` can be found in this COPR repository: https://copr.fedorainfracloud.org/coprs/leifmadsen/ovs-master/.

7.1.2 Controller nodes

Each controller node runs the OVS service (including dependent services such as `ovsdb-server`) and the `ovn-northd` service. However, only a single instance of the `ovsdb-server` and `ovn-northd` services can operate in a deployment. However, deployment tools can implement active/passive high-availability using a management tool that monitors service health and automatically starts these services on another node after failure of the primary node. See the *Frequently Asked Questions* for more information.

1. Install the `openvswitch-ovn` and `networking-ovn` packages.
2. Start the OVS service. The central OVS service starts the `ovsdb-server` service that manages OVN databases.

   Using the *systemd* unit:

   ```
   # systemctl start openvswitch
   ```

   Using the `ovs-ctl` script:

   ```
   # /usr/share/openvswitch/scripts/ovs-ctl start  --system-id="random"
   ```

3. Configure the `ovsdb-server` component. By default, the `ovsdb-server` service only permits local access to databases via Unix socket. However, OVN services on compute nodes require access to these databases.

   - Permit remote database access.

     ```
     # ovn-nbctl set-connection ptcp:6641:0.0.0.0 -- \
                 set connection . inactivity_probe=60000
     # ovn-sbctl set-connection ptcp:6642:0.0.0.0 -- \
                 set connection . inactivity_probe=60000
     # if using the VTEP functionality:
     #   ovs-appctl -t ovsdb-server ovsdb-server/add-remote␣
     ↪ptcp:6640:0.0.0.0
     ```

     Replace `0.0.0.0` with the IP address of the management network interface on the controller node to avoid listening on all interfaces.

     Note: Permit remote access to TCP ports: 6640 (OVS) to VTEPS (if you use vteps), 6642 (SBDB) to hosts running neutron-server, gateway nodes that run ovn-controller, and compute node services like ovn-controller an ovn-metadata-agent. 6641 (NBDB) to hosts running neutron-server.

4. Start the `ovn-northd` service.

   Using the *systemd* unit:

   ```
   # systemctl start ovn-northd
   ```

   Using the `ovn-ctl` script:

   ```
   # /usr/share/openvswitch/scripts/ovn-ctl start_northd
   ```

   Options for *start_northd*:

```
# /usr/share/openvswitch/scripts/ovn-ctl start_northd --help
# ...
# DB_NB_SOCK="/usr/local/etc/openvswitch/nb_db.sock"
# DB_NB_PID="/usr/local/etc/openvswitch/ovnnb_db.pid"
# DB_SB_SOCK="usr/local/etc/openvswitch/sb_db.sock"
# DB_SB_PID="/usr/local/etc/openvswitch/ovnsb_db.pid"
# ...
```

5. Configure the Networking server component. The Networking service implements OVN as an ML2 driver. Edit the `/etc/neutron/neutron.conf` file:

   - Enable the ML2 core plug-in.

   ```
   [DEFAULT]
   ...
   core_plugin = ml2
   ```

   - Enable the OVN layer-3 service.

   ```
   [DEFAULT]
   ...
   service_plugins = ovn-router
   ```

6. Configure the ML2 plug-in. Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file:

   - Configure the OVN mechanism driver, network type drivers, self-service (tenant) network types, and enable the port security extension.

   ```
   [ml2]
   ...
   mechanism_drivers = ovn
   type_drivers = local,flat,vlan,geneve
   tenant_network_types = geneve
   extension_drivers = port_security
   overlay_ip_version = 4
   ```

   Note: To enable VLAN self-service networks, make sure that OVN version 2.11 (or higher) is used, then add `vlan` to the `tenant_network_types` option. The first network type in the list becomes the default self-service network type.

   To use IPv6 for all overlay (tunnel) network endpoints, set the `overlay_ip_version` option to `6`.

   - Configure the Geneve ID range and maximum header size. The IP version overhead (20 bytes for IPv4 (default) or 40 bytes for IPv6) is added to the maximum header size based on the ML2 `overlay_ip_version` option.

   ```
   [ml2_type_geneve]
   ...
   vni_ranges = 1:65536
   max_header_size = 38
   ```

   Note: The Networking service uses the `vni_ranges` option to allocate network segments. However, OVN ignores the actual values. Thus, the ID range only determines the quantity of Geneve networks in the environment. For example, a range of `5001:6000` defines a maximum of 1000 Geneve networks.

- Optionally, enable support for VLAN provider and self-service networks on one or more physical networks. If you specify only the physical network, only administrative (privileged) users can manage VLAN networks. Additionally specifying a VLAN ID range for a physical network enables regular (non-privileged) users to manage VLAN networks. The Networking service allocates the VLAN ID for each self-service network using the VLAN ID range for the physical network.

```
[ml2_type_vlan]
...
network_vlan_ranges = PHYSICAL_NETWORK:MIN_VLAN_ID:MAX_VLAN_ID
```

  Replace `PHYSICAL_NETWORK` with the physical network name and optionally define the minimum and maximum VLAN IDs. Use a comma to separate each physical network.

  For example, to enable support for administrative VLAN networks on the `physnet1` network and self-service VLAN networks on the `physnet2` network using VLAN IDs 1001 to 2000:

```
network_vlan_ranges = physnet1,physnet2:1001:2000
```

- Enable security groups.

```
[securitygroup]
...
enable_security_group = true
```

  Note: The `firewall_driver` option under `[securitygroup]` is ignored since the OVN ML2 driver itself handles security groups.

- Configure OVS database access and L3 scheduler

```
[ovn]
...
ovn_nb_connection = tcp:IP_ADDRESS:6641
ovn_sb_connection = tcp:IP_ADDRESS:6642
ovn_l3_scheduler = OVN_L3_SCHEDULER
```

  Note: Replace `IP_ADDRESS` with the IP address of the controller node that runs the `ovsdb-server` service. Replace `OVN_L3_SCHEDULER` with `leastloaded` if you want the scheduler to select a compute node with the least number of gateway ports or `chance` if you want the scheduler to randomly select a compute node from the available list of compute nodes.

- Set ovn-cms-options with enable-chassis-as-gw in Open_vSwitch tables external_ids column. Then if this chassis has proper bridge mappings, it will be selected for scheduling gateway routers.

```
# ovs-vsctl set open . external-ids:ovn-cms-options=enable-
↪chassis-as-gw
```

7. Start the `neutron-server` service.

7.1.3 Network nodes

Deployments using OVN native layer-3 and DHCP services do not require conventional network nodes because connectivity to external networks (including VTEP gateways) and routing occurs on compute nodes.

7.1.4 Compute nodes

Each compute node runs the OVS and `ovn-controller` services. The `ovn-controller` service replaces the conventional OVS layer-2 agent.

1. Install the `openvswitch-ovn` and `networking-ovn` packages.
2. Start the OVS service.

   Using the *systemd* unit:

   ```
   # systemctl start openvswitch
   ```

   Using the `ovs-ctl` script:

   ```
   # /usr/share/openvswitch/scripts/ovs-ctl start --system-id="random"
   ```

3. Configure the OVS service.
   - Use OVS databases on the controller node.

     ```
     # ovs-vsctl set open . external-ids:ovn-remote=tcp:IP_ADDRESS:6642
     ```

     Replace `IP_ADDRESS` with the IP address of the controller node that runs the `ovsdb-server` service.

   - Enable one or more overlay network protocols. At a minimum, OVN requires enabling the `geneve` protocol. Deployments using VTEP gateways should also enable the `vxlan` protocol.

     ```
     # ovs-vsctl set open . external-ids:ovn-encap-type=geneve,vxlan
     ```

     Note: Deployments without VTEP gateways can safely enable both protocols.

   - Configure the overlay network local endpoint IP address.

     ```
     # ovs-vsctl set open . external-ids:ovn-encap-ip=IP_ADDRESS
     ```

     Replace `IP_ADDRESS` with the IP address of the overlay network interface on the compute node.

4. Start the `ovn-controller` service.

   Using the *systemd* unit:

   ```
   # systemctl start ovn-controller
   ```

   Using the `ovn-ctl` script:

   ```
   # /usr/share/openvswitch/scripts/ovn-ctl start_controller
   ```

7.1.5 Verify operation

1. Each compute node should contain an `ovn-controller` instance.

   ```
   # ovn-sbctl show
     <output>
   ```

7.2 TripleO/RDO based deployments

TripleO is a project aimed at installing, upgrading and operating OpenStack clouds using OpenStacks own cloud facilities as the foundation.

RDO is the OpenStack distribution that runs on top of CentOS, and can be deployed via TripleO.

TripleO Quickstart is an easy way to try out TripleO in a libvirt virtualized environment.

In this document we will stick to the details of installing a 3 controller + 1 compute in high availability through TripleO Quickstart, but the non-quickstart details in this document also work with TripleO.

Note: This deployment requires 32GB for the VMs, so your host may have >32GB of RAM at least. If you have 32GB I recommend to trim down the compute node memory in config/nodes/3ctlr_1comp.yml to 2GB and controller nodes to 5GB.

7.2.1 Deployment steps

1. Download the quickstart.sh script with curl:

   ```
   $ curl -O https://raw.githubusercontent.com/openstack/tripleo-
   ↪quickstart/master/quickstart.sh
   ```

2. Install the necessary dependencies by running:

   ```
   $ bash quickstart.sh --install-deps
   ```

3. Clone the tripleo-quickstart and neutron repositories:

   ```
   $ git clone https://opendev.org/openstack/tripleo-quickstart
   $ git clone https://opendev.org/openstack/neutron
   ```

4. Once youre done, run quickstart as follows (3 controller HA + 1 compute):

```
# Exporting the tags is a workaround until the bug
# https://bugs.launchpad.net/tripleo/+bug/1737602 is resolved

$ export ansible_tags="untagged,provision,environment,libvirt,\
undercloud-scripts,undercloud-inventory,overcloud-scripts,\
undercloud-setup,undercloud-install,undercloud-post-install,\
overcloud-prep-config"

$ bash ./quickstart.sh --tags $ansible_tags --teardown all \
--release master-tripleo-ci \
--nodes tripleo-quickstart/config/nodes/3ctlr_1comp.yml  \
--config neutron/tools/tripleo/ovn.yml \
$VIRTHOST
```

Note: When deploying directly on `localhost` use the loopback address 127.0.0.2 as your $VIRTHOST. The loopback address 127.0.0.1 is reserved by ansible. Also make sure that 127.0.0.2 is accessible via public keys:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Note: You can adjust RAM/VCPUs if you want by editing *config/nodes/3ctlr_1comp.yml* before running the above command. If you have enough memory stick to the defaults. We recommend using 8GB of RAM for the controller nodes.

5. When quickstart has finished you will have 5 VMs ready to be used, 1 for the undercloud (TripleOs node to deploy your openstack from), 3 VMs for controller nodes and 1 VM for the compute node.
6. Log in into the undercloud:

```
$ ssh -F ~/.quickstart/ssh.config.ansible undercloud
```

7. Prepare overcloud container images:

```
[stack@undercloud ~]$ ./overcloud-prep-containers.sh
```

8. Run inside the undercloud:

```
[stack@undercloud ~]$ ./overcloud-deploy.sh
```

9. Grab a coffee, that may take around 1 hour (depending on your hardware).
10. If anything goes wrong, go to IRC on freenode, and ask on #oooq

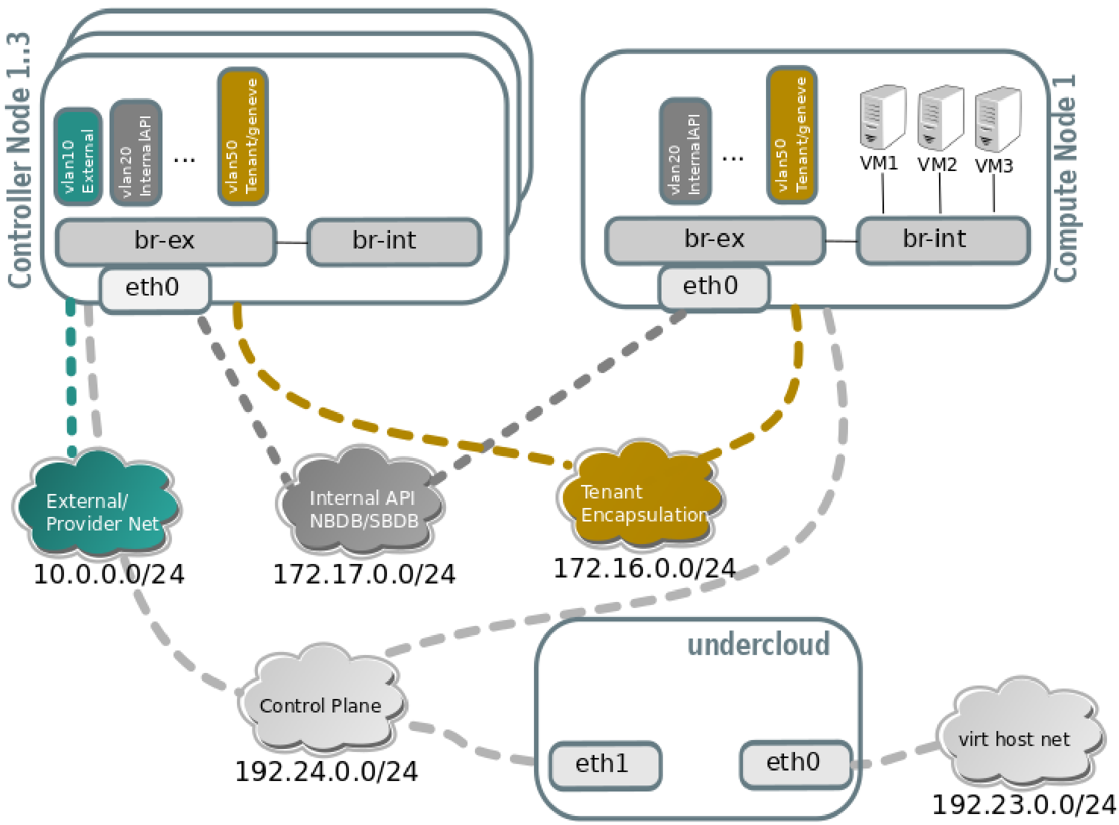
7.2.2 Description of the environment

Once deployed, inside the undercloud root directory two files are present: stackrc and overcloudrc, which will let you connect to the APIs of the undercloud (managing the openstack node), and to the overcloud (where your instances would live).

We can find out the existing controller/computes this way:

```
[stack@undercloud ~]$ source stackrc
(undercloud) [stack@undercloud ~]$ openstack server list -c Name -c␣
↪Networks -c Flavor
+-------------------------+------------------------+--------------+
| Name                    | Networks               | Flavor       |
+-------------------------+------------------------+--------------+
| overcloud-controller-1  | ctlplane=192.168.24.16 | oooq_control |
| overcloud-controller-0  | ctlplane=192.168.24.14 | oooq_control |
| overcloud-controller-2  | ctlplane=192.168.24.12 | oooq_control |
| overcloud-novacompute-0 | ctlplane=192.168.24.13 | oooq_compute |
+-------------------------+------------------------+--------------+
```

Network architecture of the environment



Connecting to one of the nodes via ssh

We can connect to the IP address in the *openstack server list* we showed before.

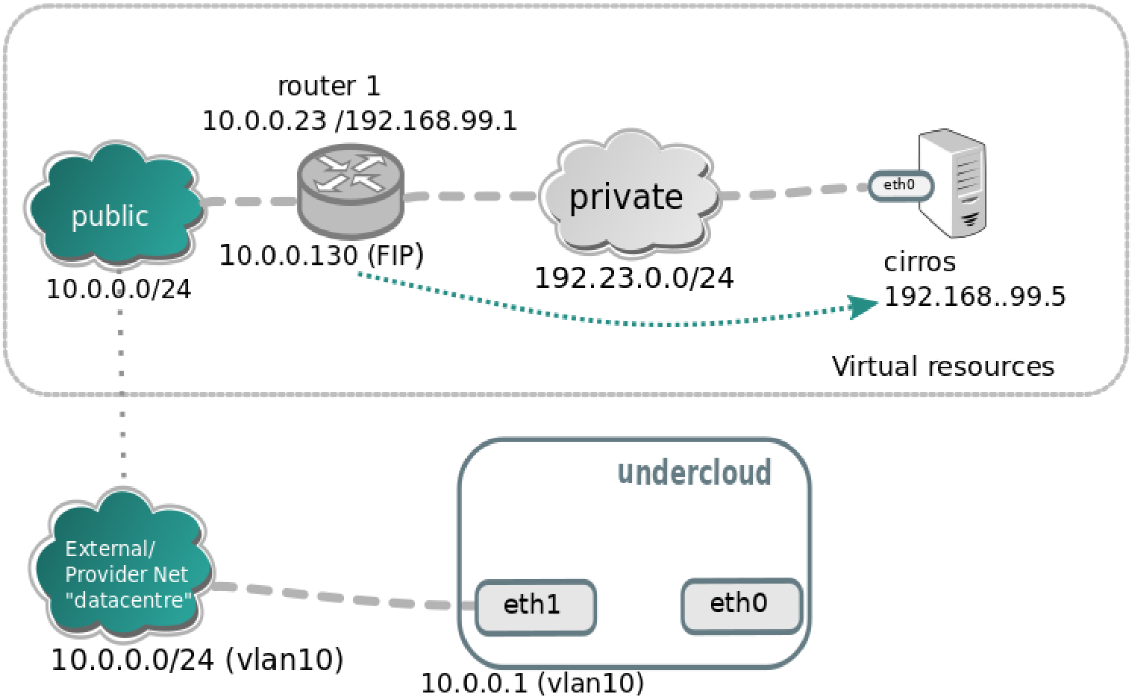
```
(undercloud) [stack@undercloud ~]$ ssh heat-admin@192.168.24.16
Last login: Wed Feb 21 14:11:40 2018 from 192.168.24.1

[heat-admin@overcloud-controller-1 ~]$ ps fax | grep ovn-controller
 20422 ?        S<s   30:40 ovn-controller unix:/var/run/openvswitch/db.
↪sock -vconsole:emer -vsyslog:err -vfile:info --no-chdir --log-file=/var/
↪log/openvswitch/ovn-controller.log --pidfile=/var/run/openvswitch/ovn-
↪controller.pid --detach

[heat-admin@overcloud-controller-1 ~]$ sudo ovs-vsctl show
bb413f44-b74f-4678-8d68-a2c6de725c73
Bridge br-ex
  fail_mode: standalone
  ...
  Port "patch-provnet-84d63c87-aad1-43d0-bdc9-dca5145b6fe6-to-br-int"
      Interface "patch-provnet-84d63c87-aad1-43d0-bdc9-dca5145b6fe6-to-br-
↪int"
          type: patch
          options: {peer="patch-br-int-to-provnet-84d63c87-aad1-43d0-bdc9-
↪dca5145b6fe6"}
  Port "eth0"
      Interface "eth0"
  ...
Bridge br-int
  fail_mode: secure
   Port "ovn-c8b85a-0"
   Interface "ovn-c8b85a-0"
       type: geneve
       options: {csum="true", key=flow, remote_ip="172.16.0.17"}
   Port "ovn-b5643d-0"
      Interface "ovn-b5643d-0"
          type: geneve
          options: {csum="true", key=flow, remote_ip="172.16.0.14"}
   Port "ovn-14d60a-0"
      Interface "ovn-14d60a-0"
          type: geneve
          options: {csum="true", key=flow, remote_ip="172.16.0.12"}
   Port "patch-br-int-to-provnet-84d63c87-aad1-43d0-bdc9-dca5145b6fe6"
      Interface "patch-br-int-to-provnet-84d63c87-aad1-43d0-bdc9-
↪dca5145b6fe6"
          type: patch
          options: {peer="patch-provnet-84d63c87-aad1-43d0-bdc9-
↪dca5145b6fe6-to-br-int"}
   Port br-int
      Interface br-int
          type: internal
```

7.2.3 Initial resource creation

Well, now you have a virtual cloud with 3 controllers in HA, and one compute node, but no instances or routers running. We can give it a try and create a few resources:



You can use the following script to create the resources.

```
ssh -F ~ /.quickstart/ssh.config.ansible undercloud

source ~/overcloudrc

curl http://download.cirros-cloud.net/0.5.0/cirros-0.5.1-x86_64-disk.img \
  > cirros-0.5.1-x86_64-disk.img
openstack image create "cirros" --file cirros-0.5.1-x86_64-disk.img   \
               --disk-format qcow2 --container-format bare --public

openstack network create public --provider-physical-network datacentre \
                                --provider-network-type vlan \
                                --provider-segment 10 \
                                --external --share

openstack subnet create --network public public --subnet-range 10.0.0.0/24␣
↪\
                         --allocation-pool start=10.0.0.20,end=10.0.0.250 \
                         --dns-nameserver 8.8.8.8 --gateway 10.0.0.1 \
                         --no-dhcp

openstack network create private
openstack subnet create --network private private \
                        --subnet-range 192.168.99.0/24
```

(continues on next page)

(continued from previous page)

```
openstack router create router1

openstack router set --external-gateway public router1
openstack router add subnet router1 private

openstack security group create test
openstack security group rule create --ingress --protocol tcp \
                                     --dst-port 22 test
openstack security group rule create --ingress --protocol icmp test
openstack security group rule create --egress test

openstack flavor create m1.tiny --disk 1 --vcpus 1 --ram 64

PRIV_NET=$(openstack network show private -c id -f value)

openstack server create --flavor m1.tiny --image cirros \
                        --nic net-id=$PRIV_NET --security-group test \
                        --wait cirros

openstack floating ip create --floating-ip-address 10.0.0.130 public
openstack server add floating ip cirros 10.0.0.130
```

Note: You can now log in into the instance if you want. In a CirrOS >0.4.0 image, the login account is cirros. The password is *gocubsgo*.

```
(overcloud) [stack@undercloud ~]$ ssh cirros@10.0.0.130
cirros@10.0.0.130's password:

$ ip a | grep eth0 -A 10
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1442 qdisc pfifo_fast qlen␣
↪1000
    link/ether fa:16:3e:85:b4:66 brd ff:ff:ff:ff:ff:ff
    inet 192.168.99.5/24 brd 192.168.99.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe85:b466/64 scope link
       valid_lft forever preferred_lft forever

$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1): 56 data bytes
64 bytes from 10.0.0.1: seq=0 ttl=63 time=2.145 ms
64 bytes from 10.0.0.1: seq=1 ttl=63 time=1.025 ms
64 bytes from 10.0.0.1: seq=2 ttl=63 time=0.836 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.836/1.335/2.145 ms

$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=52 time=3.943 ms
64 bytes from 8.8.8.8: seq=1 ttl=52 time=4.519 ms
64 bytes from 8.8.8.8: seq=2 ttl=52 time=3.778 ms
```

(continues on next page)

(continued from previous page)

```
$ curl http://169.254.169.254/2009-04-04/meta-data/instance-id
i-00000002
```

This chapter explains how to install and configure the Networking service (neutron) using the *provider networks* or *self-service networks* option.

For more information about the Networking service including virtual networking components, layout, and traffic flows, see the *OpenStack Networking Guide*.

CHAPTER

EIGHT

OPENSTACK NETWORKING GUIDE

This guide targets OpenStack administrators seeking to deploy and manage OpenStack Networking (neutron).

8.1 Introduction

The OpenStack Networking service (neutron) provides an API that allows users to set up and define network connectivity and addressing in the cloud. The project code-name for Networking services is neutron. OpenStack Networking handles the creation and management of a virtual networking infrastructure, including networks, switches, subnets, and routers for devices managed by the OpenStack Compute service (nova). Advanced services such as firewalls or virtual private network (VPN) can also be used.

OpenStack Networking consists of the neutron-server, a database for persistent storage, and any number of plug-in agents, which provide other services such as interfacing with native Linux networking mechanisms, external devices, or SDN controllers.

OpenStack Networking is entirely standalone and can be deployed to a dedicated host. If your deployment uses a controller host to run centralized Compute components, you can deploy the Networking server to that specific host instead.

OpenStack Networking integrates with various OpenStack components:

- OpenStack Identity service (keystone) is used for authentication and authorization of API requests.
- OpenStack Compute service (nova) is used to plug each virtual NIC on the VM into a particular network.
- OpenStack Dashboard (horizon) is used by administrators and project users to create and manage network services through a web-based graphical interface.

Note: The network address ranges used in this guide are chosen in accordance with RFC 5737 and RFC 3849, and as such are restricted to the following:

IPv4:

- 192.0.2.0/24
- 198.51.100.0/24
- 203.0.113.0/24

IPv6:

- 2001:DB8::/32

The network address ranges in the examples of this guide should not be used for any purpose other than documentation.

Note: To reduce clutter, this guide removes command output without relevance to the particular action.

8.1.1 Basic networking

Ethernet

Ethernet is a networking protocol, specified by the IEEE 802.3 standard. Most wired network interface cards (NICs) communicate using Ethernet.

In the OSI model of networking protocols, Ethernet occupies the second layer, which is known as the data link layer. When discussing Ethernet, you will often hear terms such as *local network*, *layer 2*, *L2*, *link layer* and *data link layer*.

In an Ethernet network, the hosts connected to the network communicate by exchanging *frames*. Every host on an Ethernet network is uniquely identified by an address called the media access control (MAC) address. In particular, every virtual machine instance in an OpenStack environment has a unique MAC address, which is different from the MAC address of the compute host. A MAC address has 48 bits and is typically represented as a hexadecimal string, such as `08:00:27:b9:88:74`. The MAC address is hard-coded into the NIC by the manufacturer, although modern NICs allow you to change the MAC address programmatically. In Linux, you can retrieve the MAC address of a NIC using the **ip** command:

```
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:b9:88:74 brd ff:ff:ff:ff:ff:ff
```

Conceptually, you can think of an Ethernet network as a single bus that each of the network hosts connects to. In early implementations, an Ethernet network consisted of a single coaxial cable that hosts would tap into to connect to the network. However, network hosts in modern Ethernet networks connect directly to a network device called a *switch*. Still, this conceptual model is useful, and in network diagrams (including those generated by the OpenStack dashboard) an Ethernet network is often depicted as if it was a single bus. Youll sometimes hear an Ethernet network referred to as a *layer 2 segment*.

In an Ethernet network, every host on the network can send a frame directly to every other host. An Ethernet network also supports broadcasts so that one host can send a frame to every host on the network by sending to the special MAC address `ff:ff:ff:ff:ff:ff`. *ARP* and *DHCP* are two notable protocols that use Ethernet broadcasts. Because Ethernet networks support broadcasts, you will sometimes hear an Ethernet network referred to as a *broadcast domain*.

When a NIC receives an Ethernet frame, by default the NIC checks to see if the destination MAC address matches the address of the NIC (or the broadcast address), and the Ethernet frame is discarded if the MAC address does not match. For a compute host, this behavior is undesirable because the frame may be intended for one of the instances. NICs can be configured for *promiscuous mode*, where they pass all Ethernet frames to the operating system, even if the MAC address does not match. Compute hosts should always have the appropriate NICs configured for promiscuous mode.

As mentioned earlier, modern Ethernet networks use switches to interconnect the network hosts. A switch is a box of networking hardware with a large number of ports that forward Ethernet frames from one connected host to another. When hosts first send frames over the switch, the switch doesnt know which MAC address is associated with which port. If an Ethernet frame is destined for an unknown MAC address, the switch broadcasts the frame to all ports. The switch learns which MAC addresses are at which ports by observing the traffic. Once it knows which MAC address is associated with a port, it can send Ethernet frames to the correct port instead of broadcasting. The switch maintains the mappings of MAC addresses to switch ports in a table called a *forwarding table* or *forwarding information base* (FIB). Switches can be daisy-chained together, and the resulting connection of switches and hosts behaves like a single network.

VLANs

VLAN is a networking technology that enables a single switch to act as if it was multiple independent switches. Specifically, two hosts that are connected to the same switch but on different VLANs do not see each others traffic. OpenStack is able to take advantage of VLANs to isolate the traffic of different projects, even if the projects happen to have instances running on the same compute host. Each VLAN has an associated numerical ID, between 1 and 4095. We say VLAN 15 to refer to the VLAN with a numerical ID of 15.

To understand how VLANs work, lets consider VLAN applications in a traditional IT environment, where physical hosts are attached to a physical switch, and no virtualization is involved. Imagine a scenario where you want three isolated networks but you only have a single physical switch. The network administrator would choose three VLAN IDs, for example, 10, 11, and 12, and would configure the switch to associate switchports with VLAN IDs. For example, switchport 2 might be associated with VLAN 10, switchport 3 might be associated with VLAN 11, and so forth. When a switchport is configured for a specific VLAN, it is called an *access port*. The switch is responsible for ensuring that the network traffic is isolated across the VLANs.

Now consider the scenario that all of the switchports in the first switch become occupied, and so the organization buys a second switch and connects it to the first switch to expand the available number of switchports. The second switch is also configured to support VLAN IDs 10, 11, and 12. Now imagine host A connected to switch 1 on a port configured for VLAN ID 10 sends an Ethernet frame intended for host B connected to switch 2 on a port configured for VLAN ID 10. When switch 1 forwards the Ethernet frame to switch 2, it must communicate that the frame is associated with VLAN ID 10.

If two switches are to be connected together, and the switches are configured for VLANs, then the switchports used for cross-connecting the switches must be configured to allow Ethernet frames from any VLAN to be forwarded to the other switch. In addition, the sending switch must tag each Ethernet frame with the VLAN ID so that the receiving switch can ensure that only hosts on the matching VLAN are eligible to receive the frame.

A switchport that is configured to pass frames from all VLANs and tag them with the VLAN IDs is called a *trunk port*. IEEE 802.1Q is the network standard that describes how VLAN tags are encoded in Ethernet frames when trunking is being used.

Note that if you are using VLANs on your physical switches to implement project isolation in your OpenStack cloud, you must ensure that all of your switchports are configured as trunk ports.

It is important that you select a VLAN range not being used by your current network infrastructure. For example, if you estimate that your cloud must support a maximum of 100 projects, pick a VLAN range outside of that value, such as VLAN 200299. OpenStack, and all physical network infrastructure that handles project networks, must then support this VLAN range.

Trunking is used to connect between different switches. Each trunk uses a tag to identify which VLAN is in use. This ensures that switches on the same VLAN can communicate.

Subnets and ARP

While NICs use MAC addresses to address network hosts, TCP/IP applications use IP addresses. The Address Resolution Protocol (ARP) bridges the gap between Ethernet and IP by translating IP addresses into MAC addresses.

IP addresses are broken up into two parts: a *network number* and a *host identifier*. Two hosts are on the same *subnet* if they have the same network number. Recall that two hosts can only communicate directly over Ethernet if they are on the same local network. ARP assumes that all machines that are in the same subnet are on the same local network. Network administrators must take care when assigning IP addresses and netmasks to hosts so that any two hosts that are in the same subnet are on the same local network, otherwise ARP does not work properly.

To calculate the network number of an IP address, you must know the *netmask* associated with the address. A netmask indicates how many of the bits in the 32-bit IP address make up the network number.

There are two syntaxes for expressing a netmask:

- dotted quad
- classless inter-domain routing (CIDR)

Consider an IP address of 192.0.2.5, where the first 24 bits of the address are the network number. In dotted quad notation, the netmask would be written as `255.255.255.0`. CIDR notation includes both the IP address and netmask, and this example would be written as `192.0.2.5/24`.

Note: Creating CIDR subnets including a multicast address or a loopback address cannot be used in an OpenStack environment. For example, creating a subnet using `224.0.0.0/16` or `127.0.1.0/24` is not supported.

Sometimes we want to refer to a subnet, but not any particular IP address on the subnet. A common convention is to set the host identifier to all zeros to make reference to a subnet. For example, if a hosts IP address is `192.0.2.24/24`, then we would say the subnet is `192.0.2.0/24`.

To understand how ARP translates IP addresses to MAC addresses, consider the following example. Assume host *A* has an IP address of `192.0.2.5/24` and a MAC address of `fc:99:47:49:d4:a0`, and wants to send a packet to host *B* with an IP address of `192.0.2.7`. Note that the network number is the same for both hosts, so host *A* is able to send frames directly to host *B*.

The first time host *A* attempts to communicate with host *B*, the destination MAC address is not known. Host *A* makes an ARP request to the local network. The request is a broadcast with a message like this:

To: everybody (ff:ff:ff:ff:ff:ff). I am looking for the computer who has IP address 192.0.2.7. Signed: MAC address fc:99:47:49:d4:a0.

Host *B* responds with a response like this:

To: fc:99:47:49:d4:a0. I have IP address 192.0.2.7. Signed: MAC address 54:78:1a:86:00:a5.

Host *A* then sends Ethernet frames to host *B*.

You can initiate an ARP request manually using the **`arping`** command. For example, to send an ARP request to IP address `192.0.2.132`:

```
$ arping -I eth0 192.0.2.132
ARPING 192.0.2.132 from 192.0.2.131 eth0
Unicast reply from 192.0.2.132 [54:78:1A:86:1C:0B]  0.670ms
Unicast reply from 192.0.2.132 [54:78:1A:86:1C:0B]  0.722ms
Unicast reply from 192.0.2.132 [54:78:1A:86:1C:0B]  0.723ms
Sent 3 probes (1 broadcast(s))
Received 3 response(s)
```

To reduce the number of ARP requests, operating systems maintain an ARP cache that contains the mappings of IP addresses to MAC address. On a Linux machine, you can view the contents of the ARP cache by using the **arp** command:

```
$ arp -n
Address                  HWtype  HWaddress           Flags Mask            ␣
↪Iface
192.0.2.3                ether   52:54:00:12:35:03   C                     ␣
↪eth0
192.0.2.2                ether   52:54:00:12:35:02   C                     ␣
↪eth0
```

DHCP

Hosts connected to a network use the Dynamic Host Configuration Protocol (DHCP) to dynamically obtain IP addresses. A DHCP server hands out the IP addresses to network hosts, which are the DHCP clients.

DHCP clients locate the DHCP server by sending a *UDP* packet from port 68 to address `255.255.255.255` on port 67. Address `255.255.255.255` is the local network broadcast address: all hosts on the local network see the UDP packets sent to this address. However, such packets are not forwarded to other networks. Consequently, the DHCP server must be on the same local network as the client, or the server will not receive the broadcast. The DHCP server responds by sending a UDP packet from port 67 to port 68 on the client. The exchange looks like this:

1. The client sends a discover (Im a client at MAC address `08:00:27:b9:88:74`, I need an IP address)
2. The server sends an offer (OK `08:00:27:b9:88:74`, Im offering IP address `192.0.2.112`)
3. The client sends a request (Server `192.0.2.131`, I would like to have IP `192.0.2.112`)
4. The server sends an acknowledgement (OK `08:00:27:b9:88:74`, IP `192.0.2.112` is yours)

OpenStack uses a third-party program called dnsmasq to implement the DHCP server. Dnsmasq writes to the syslog, where you can observe the DHCP request and replies:

```
Apr 23 15:53:46 c100-1 dhcpd: DHCPDISCOVER from 08:00:27:b9:88:74 via eth2
Apr 23 15:53:46 c100-1 dhcpd: DHCPOFFER on 192.0.2.112 to␣
↪08:00:27:b9:88:74 via eth2
Apr 23 15:53:48 c100-1 dhcpd: DHCPREQUEST for 192.0.2.112 (192.0.2.131)␣
↪from 08:00:27:b9:88:74 via eth2
Apr 23 15:53:48 c100-1 dhcpd: DHCPACK on 192.0.2.112 to 08:00:27:b9:88:74␣
↪via eth2
```

When troubleshooting an instance that is not reachable over the network, it can be helpful to examine this log to verify that all four steps of the DHCP protocol were carried out for the instance in question.

IP

The Internet Protocol (IP) specifies how to route packets between hosts that are connected to different local networks. IP relies on special network hosts called *routers* or *gateways*. A router is a host that is connected to at least two local networks and can forward IP packets from one local network to another. A router has multiple IP addresses: one for each of the networks it is connected to.

In the OSI model of networking protocols IP occupies the third layer, known as the network layer. When discussing IP, you will often hear terms such as *layer 3*, *L3*, and *network layer*.

A host sending a packet to an IP address consults its *routing table* to determine which machine on the local network(s) the packet should be sent to. The routing table maintains a list of the subnets associated with each local network that the host is directly connected to, as well as a list of routers that are on these local networks.

On a Linux machine, any of the following commands displays the routing table:

```
$ ip route show
$ route -n
$ netstat -rn
```

Here is an example of output from **ip route show**:

```
$ ip route show
default via 192.0.2.2 dev eth0
192.0.2.0/24 dev eth0  proto kernel  scope link  src 192.0.2.15
198.51.100.0/25 dev eth1  proto kernel  scope link  src 198.51.100.100
198.51.100.192/26 dev virbr0  proto kernel  scope link  src 198.51.100.193
```

Line 1 of the output specifies the location of the default route, which is the effective routing rule if none of the other rules match. The router associated with the default route (`192.0.2.2` in the example above) is sometimes referred to as the *default gateway*. A *DHCP* server typically transmits the IP address of the default gateway to the DHCP client along with the clients IP address and a netmask.

Line 2 of the output specifies that IPs in the `192.0.2.0/24` subnet are on the local network associated with the network interface eth0.

Line 3 of the output specifies that IPs in the `198.51.100.0/25` subnet are on the local network associated with the network interface eth1.

Line 4 of the output specifies that IPs in the `198.51.100.192/26` subnet are on the local network associated with the network interface virbr0.

The output of the **route -n** and **netstat -rn** commands are formatted in a slightly different way. This example shows how the same routes would be formatted using these commands:

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt␣
↪Iface
0.0.0.0         192.0.2.2       0.0.0.0         UG        0 0          0␣
↪eth0
192.0.2.0       0.0.0.0         255.255.255.0   U         0 0          0␣
↪eth0
198.51.100.0    0.0.0.0         255.255.255.128 U         0 0          0␣
↪eth1
198.51.100.192  0.0.0.0         255.255.255.192 U         0 0          0␣
↪virbr0
```

(continues on next page)

(continued from previous page)

The **ip route get** command outputs the route for a destination IP address. From the below example, destination IP address `192.0.2.14` is on the local network of eth0 and would be sent directly:

```
$ ip route get 192.0.2.14
192.0.2.14 dev eth0  src 192.0.2.15
```

The destination IP address `203.0.113.34` is not on any of the connected local networks and would be forwarded to the default gateway at `192.0.2.2`:

```
$ ip route get 203.0.113.34
203.0.113.34 via 192.0.2.2 dev eth0  src 192.0.2.15
```

It is common for a packet to hop across multiple routers to reach its final destination. On a Linux machine, the `traceroute` and more recent `mtr` programs prints out the IP address of each router that an IP packet traverses along its path to its destination.

TCP/UDP/ICMP

For networked software applications to communicate over an IP network, they must use a protocol layered atop IP. These protocols occupy the fourth layer of the OSI model known as the *transport layer* or *layer 4*. See the Protocol Numbers web page maintained by the Internet Assigned Numbers Authority (IANA) for a list of protocols that layer atop IP and their associated numbers.

The *Transmission Control Protocol* (TCP) is the most commonly used layer 4 protocol in networked applications. TCP is a *connection-oriented* protocol: it uses a client-server model where a client connects to a server, where *server* refers to the application that receives connections. The typical interaction in a TCP-based application proceeds as follows:

1. Client connects to server.
2. Client and server exchange data.
3. Client or server disconnects.

Because a network host may have multiple TCP-based applications running, TCP uses an addressing scheme called *ports* to uniquely identify TCP-based applications. A TCP port is associated with a number in the range 1-65535, and only one application on a host can be associated with a TCP port at a time, a restriction that is enforced by the operating system.

A TCP server is said to *listen* on a port. For example, an SSH server typically listens on port 22. For a client to connect to a server using TCP, the client must know both the IP address of a servers host and the servers TCP port.

The operating system of the TCP client application automatically assigns a port number to the client. The client owns this port number until the TCP connection is terminated, after which the operating system reclaims the port number. These types of ports are referred to as *ephemeral ports*.

IANA maintains a registry of port numbers for many TCP-based services, as well as services that use other layer 4 protocols that employ ports. Registering a TCP port number is not required, but registering a port number is helpful to avoid collisions with other services. See firewalls and default ports in OpenStack Installation Guide for the default TCP ports used by various services involved in an OpenStack deployment.

The most common application programming interface (API) for writing TCP-based applications is called *Berkeley sockets*, also known as *BSD sockets* or, simply, *sockets*. The sockets API exposes a *stream oriented* interface for writing TCP applications. From the perspective of a programmer, sending data over a TCP connection is similar to writing a stream of bytes to a file. It is the responsibility of the operating systems TCP/IP implementation to break up the stream of data into IP packets. The operating system is also responsible for automatically retransmitting dropped packets, and for handling flow control to ensure that transmitted data does not overrun the senders data buffers, receivers data buffers, and network capacity. Finally, the operating system is responsible for re-assembling the packets in the correct order into a stream of data on the receivers side. Because TCP detects and retransmits lost packets, it is said to be a *reliable* protocol.

The *User Datagram Protocol* (UDP) is another layer 4 protocol that is the basis of several well-known networking protocols. UDP is a *connectionless* protocol: two applications that communicate over UDP do not need to establish a connection before exchanging data. UDP is also an *unreliable* protocol. The operating system does not attempt to retransmit or even detect lost UDP packets. The operating system also does not provide any guarantee that the receiving application sees the UDP packets in the same order that they were sent in.

UDP, like TCP, uses the notion of ports to distinguish between different applications running on the same system. Note, however, that operating systems treat UDP ports separately from TCP ports. For example, it is possible for one application to be associated with TCP port 16543 and a separate application to be associated with UDP port 16543.

Like TCP, the sockets API is the most common API for writing UDP-based applications. The sockets API provides a *message-oriented* interface for writing UDP applications: a programmer sends data over UDP by transmitting a fixed-sized message. If an application requires retransmissions of lost packets or a well-defined ordering of received packets, the programmer is responsible for implementing this functionality in the application code.

DHCP, the Domain Name System (DNS), the Network Time Protocol (NTP), and *Virtual extensible local area network (VXLAN)* are examples of UDP-based protocols used in OpenStack deployments.

UDP has support for one-to-many communication: sending a single packet to multiple hosts. An application can broadcast a UDP packet to all of the network hosts on a local network by setting the receiver IP address as the special IP broadcast address `255.255.255.255`. An application can also send a UDP packet to a set of receivers using *IP multicast*. The intended receiver applications join a multicast group by binding a UDP socket to a special IP address that is one of the valid multicast group addresses. The receiving hosts do not have to be on the same local network as the sender, but the intervening routers must be configured to support IP multicast routing. VXLAN is an example of a UDP-based protocol that uses IP multicast.

The *Internet Control Message Protocol* (ICMP) is a protocol used for sending control messages over an IP network. For example, a router that receives an IP packet may send an ICMP packet back to the source if there is no route in the routers routing table that corresponds to the destination address (ICMP code 1, destination host unreachable) or if the IP packet is too large for the router to handle (ICMP code 4, fragmentation required and dont fragment flag is set).

The **ping** and **mtr** Linux command-line tools are two examples of network utilities that use ICMP.

8.1.2 Network components

Switches

Switches are Multi-Input Multi-Output (MIMO) devices that enable packets to travel from one node to another. Switches connect hosts that belong to the same layer-2 network. Switches enable forwarding of the packet received on one port (input) to another port (output) so that they reach the desired destination node. Switches operate at layer-2 in the networking model. They forward the traffic based on the destination Ethernet address in the packet header.

Routers

Routers are special devices that enable packets to travel from one layer-3 network to another. Routers enable communication between two nodes on different layer-3 networks that are not directly connected to each other. Routers operate at layer-3 in the networking model. They route the traffic based on the destination IP address in the packet header.

Firewalls

Firewalls are used to regulate traffic to and from a host or a network. A firewall can be either a specialized device connecting two networks or a software-based filtering mechanism implemented on an operating system. Firewalls are used to restrict traffic to a host based on the rules defined on the host. They can filter packets based on several criteria such as source IP address, destination IP address, port numbers, connection state, and so on. It is primarily used to protect the hosts from unauthorized access and malicious attacks. Linux-based operating systems implement firewalls through `iptables`.

Load balancers

Load balancers can be software-based or hardware-based devices that allow traffic to evenly be distributed across several servers. By distributing the traffic across multiple servers, it avoids overload of a single server thereby preventing a single point of failure in the product. This further improves the performance, network throughput, and response time of the servers. Load balancers are typically used in a 3-tier architecture. In this model, a load balancer receives a request from the front-end web server, which then forwards the request to one of the available back-end database servers for processing. The response from the database server is passed back to the web server for further processing.

8.1.3 Overlay (tunnel) protocols

Tunneling is a mechanism that makes transfer of payloads feasible over an incompatible delivery network. It allows the network user to gain access to denied or insecure networks. Data encryption may be employed to transport the payload, ensuring that the encapsulated user network data appears as public even though it is private and can easily pass the conflicting network.

Generic routing encapsulation (GRE)

Generic routing encapsulation (GRE) is a protocol that runs over IP and is employed when delivery and payload protocols are compatible but payload addresses are incompatible. For instance, a payload might think it is running on a datalink layer but it is actually running over a transport layer using datagram protocol over IP. GRE creates a private point-to-point connection and works by encapsulating a payload. GRE is a foundation protocol for other tunnel protocols but the GRE tunnels provide only weak authentication.

Virtual extensible local area network (VXLAN)

The purpose of VXLAN is to provide scalable network isolation. VXLAN is a Layer 2 overlay scheme on a Layer 3 network. It allows an overlay layer-2 network to spread across multiple underlay layer-3 network domains. Each overlay is termed a VXLAN segment. Only VMs within the same VXLAN segment can communicate.

Generic Network Virtualization Encapsulation (GENEVE)

Geneve is designed to recognize and accommodate 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.

8.1.4 Network namespaces

A namespace is a way of scoping a particular set of identifiers. Using a namespace, you can use the same identifier multiple times in different namespaces. You can also restrict an identifier set visible to particular processes.

For example, Linux provides namespaces for networking and processes, among other things. If a process is running within a process namespace, it can only see and communicate with other processes in the same namespace. So, if a shell in a particular process namespace ran **`ps waux`**, it would only show the other processes in the same namespace.

Linux network namespaces

In a network namespace, the scoped identifiers are network devices; so a given network device, such as `eth0`, exists in a particular namespace. Linux starts up with a default network namespace, so if your operating system does not do anything special, that is where all the network devices will be located. But it is also possible to create further non-default namespaces, and create new devices in those namespaces, or to move an existing device from one namespace to another.

Each network namespace also has its own routing table, and in fact this is the main reason for namespaces to exist. A routing table is keyed by destination IP address, so network namespaces are what you need if you want the same destination IP address to mean different things at different times - which is something that OpenStack Networking requires for its feature of providing overlapping IP addresses in different virtual networks.

Each network namespace also has its own set of iptables (for both IPv4 and IPv6). So, you can apply different security to flows with the same IP addressing in different namespaces, as well as different routing.

Any given Linux process runs in a particular network namespace. By default this is inherited from its parent process, but a process with the right capabilities can switch itself into a different namespace; in practice this is mostly done using the **`ip netns exec NETNS COMMAND...`** invocation, which starts `COMMAND` running in the namespace named `NETNS`. Suppose such a process sends out a message to IP address A.B.C.D, the effect of the namespace is that A.B.C.D will be looked up in that namespaces routing table, and that will determine the network device that the message is transmitted through.

Virtual routing and forwarding (VRF)

Virtual routing and forwarding is an IP technology that allows multiple instances of a routing table to coexist on the same router at the same time. It is another name for the network namespace functionality described above.

8.1.5 Network address translation

Network Address Translation (NAT) is a process for modifying the source or destination addresses in the headers of an IP packet while the packet is in transit. In general, the sender and receiver applications are not aware that the IP packets are being manipulated.

NAT is often implemented by routers, and so we will refer to the host performing NAT as a *NAT router*. However, in OpenStack deployments it is typically Linux servers that implement the NAT functionality, not hardware routers. These servers use the iptables software package to implement the NAT functionality.

There are multiple variations of NAT, and here we describe three kinds commonly found in OpenStack deployments.

SNAT

In *Source Network Address Translation* (SNAT), the NAT router modifies the IP address of the sender in IP packets. SNAT is commonly used to enable hosts with *private addresses* to communicate with servers on the public Internet.

RFC 1918 reserves the following three subnets as private addresses:

- `10.0.0.0/8`
- `172.16.0.0/12`
- `192.168.0.0/16`

These IP addresses are not publicly routable, meaning that a host on the public Internet can not send an IP packet to any of these addresses. Private IP addresses are widely used in both residential and corporate environments.

Often, an application running on a host with a private IP address will need to connect to a server on the public Internet. An example is a user who wants to access a public website such as www.openstack.org. If the IP packets reach the web server at www.openstack.org with a private IP address as the source, then the web server cannot send packets back to the sender.

SNAT solves this problem by modifying the source IP address to an IP address that is routable on the public Internet. There are different variations of SNAT; in the form that OpenStack deployments use, a NAT router on the path between the sender and receiver replaces the packets source IP address with the routers public IP address. The router also modifies the source TCP or UDP port to another value, and the router maintains a record of the senders true IP address and port, as well as the modified IP address and port.

When the router receives a packet with the matching IP address and port, it translates these back to the private IP address and port, and forwards the packet along.

Because the NAT router modifies ports as well as IP addresses, this form of SNAT is sometimes referred to as *Port Address Translation* (PAT). It is also sometimes referred to as *NAT overload*.

OpenStack uses SNAT to enable applications running inside of instances to connect out to the public Internet.

DNAT

In *Destination Network Address Translation* (DNAT), the NAT router modifies the IP address of the destination in IP packet headers.

OpenStack uses DNAT to route packets from instances to the OpenStack metadata service. Applications running inside of instances access the OpenStack metadata service by making HTTP GET requests to a web server with IP address 169.254.169.254. In an OpenStack deployment, there is no host with this IP address. Instead, OpenStack uses DNAT to change the destination IP of these packets so they reach the network interface that a metadata service is listening on.

One-to-one NAT

In *one-to-one NAT*, the NAT router maintains a one-to-one mapping between private IP addresses and public IP addresses. OpenStack uses one-to-one NAT to implement floating IP addresses.

8.1.6 OpenStack Networking

OpenStack Networking allows you to create and manage network objects, such as networks, subnets, and ports, which other OpenStack services can use. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

The Networking service, code-named neutron, provides an API that lets you define network connectivity and addressing in the cloud. The Networking service enables operators to leverage different networking technologies to power their cloud networking. The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and Network Address Translation (NAT) to perimeter firewalls, and virtual private networks.

It includes the following components:

API server The OpenStack 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. OpenStack 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.

OpenStack Networking plug-in and agents Plugs and unplugs ports, creates networks or subnets, and provides IP addressing. The chosen plug-in and agents differ depending on the vendor and technologies used in the particular cloud. It is important to mention that only one plug-in can be used at a time.

Messaging queue Accepts and routes RPC requests between agents to complete API operations. Message queue is used in the ML2 plug-in for RPC between the neutron server and neutron agents that run on each hypervisor, in the ML2 mechanism drivers for Open vSwitch and Linux bridge.

Concepts

To configure rich network topologies, you can create and configure networks and subnets and instruct other OpenStack services like Compute to attach virtual devices to ports on these networks. OpenStack Compute is a prominent consumer of OpenStack Networking to provide connectivity for its instances. In particular, OpenStack Networking supports each project having multiple private networks and enables projects to choose their own IP addressing scheme, even if those IP addresses overlap with those that other projects use. There are two types of network, project and provider networks. It is possible to share any of these types of networks among projects as part of the network creation process.

Provider networks

Provider networks offer layer-2 connectivity to instances with optional support for DHCP and metadata services. These networks connect, or map, to existing layer-2 networks in the data center, typically using VLAN (802.1q) tagging to identify and separate them.

Provider networks generally offer simplicity, performance, and reliability at the cost of flexibility. By default only administrators can create or update provider networks because they require configuration of physical network infrastructure. It is possible to change the user who is allowed to create or update provider networks with the following parameters of `policy.json`:

- `create_network:provider:physical_network`
- `update_network:provider:physical_network`

> **Warning:** The creation and modification of provider networks enables use of physical network resources, such as VLAN-s. Enable these changes only for trusted projects.

Also, provider networks only handle layer-2 connectivity for instances, thus lacking support for features such as routers and floating IP addresses.

In many cases, operators who are already familiar with virtual networking architectures that rely on physical network infrastructure for layer-2, layer-3, or other services can seamlessly deploy the OpenStack Networking service. In particular, provider networks appeal to operators looking to migrate from the Compute networking service (nova-network) to the OpenStack Networking service. Over time, operators can build on this minimal architecture to enable more cloud networking features.

In general, the OpenStack Networking software components that handle layer-3 operations impact performance and reliability the most. To improve performance and reliability, provider networks move layer-3 operations to the physical network infrastructure.

In one particular use case, the OpenStack deployment resides in a mixed environment with conventional virtualization and bare-metal hosts that use a sizable physical network infrastructure. Applications that run inside the OpenStack deployment might require direct layer-2 access, typically using VLANs, to applications outside of the deployment.

Routed provider networks

Routed provider networks offer layer-3 connectivity to instances. These networks map to existing layer-3 networks in the data center. More specifically, the network maps to multiple layer-2 segments, each of which is essentially a provider network. Each has a router gateway attached to it which routes traffic between them and externally. The Networking service does not provide the routing.

Routed provider networks offer performance at scale that is difficult to achieve with a plain provider network at the expense of guaranteed layer-2 connectivity.

Neutron port could be associated with only one network segment, but there is an exception for OVN distributed services like OVN Metadata.

See *Routed provider networks* for more information.

Self-service networks

Self-service 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 provider and external networks such as the Internet. Self-service networks also usually provide DHCP and metadata services to instances.

In most cases, self-service networks use overlay protocols such as VXLAN or GRE because they can support many more networks than layer-2 segmentation using VLAN tagging (802.1q). Furthermore, VLANs typically require additional configuration of physical network infrastructure.

IPv4 self-service networks typically use private IP address ranges (RFC1918) and interact with provider networks via source NAT on virtual routers. Floating IP addresses enable access to instances from provider networks via destination NAT on virtual routers. IPv6 self-service networks always use public IP address ranges and interact with provider networks via virtual routers with static routes.

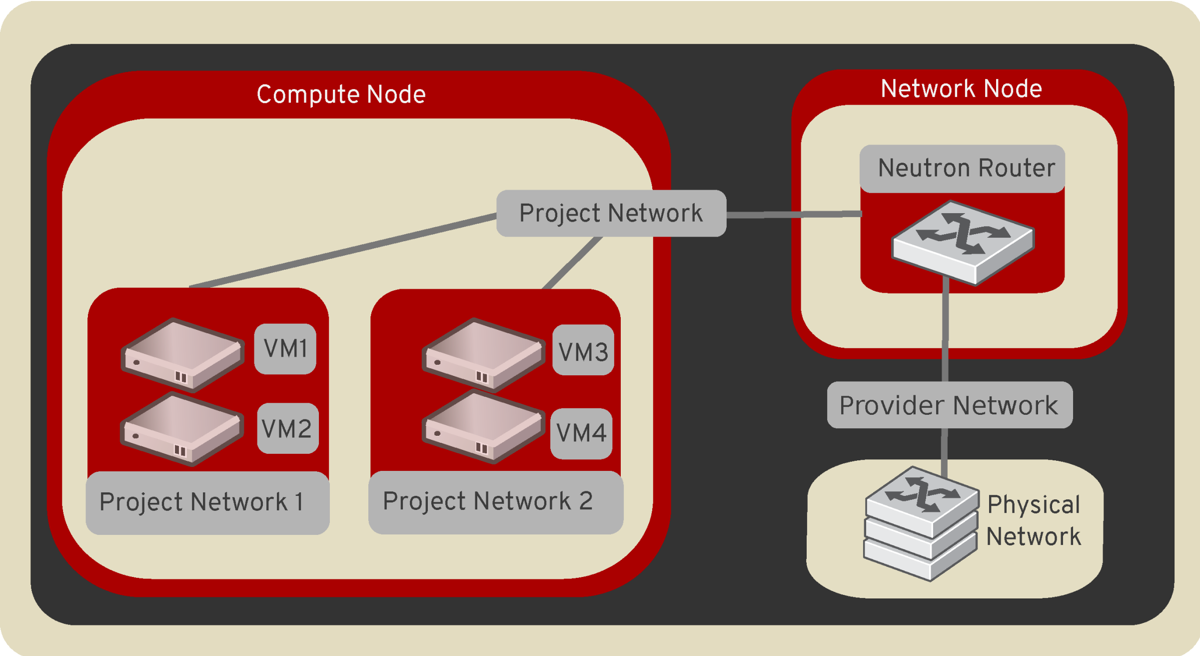
The Networking service implements routers using a layer-3 agent that typically resides at least one network node. Contrary to provider networks that connect instances to the physical network infrastructure at layer-2, self-service networks must traverse a layer-3 agent. Thus, oversubscription or failure of a layer-3 agent or network node can impact a significant quantity of self-service networks and instances using them. Consider implementing one or more high-availability features to increase redundancy and performance of self-service networks.

Users create project networks for connectivity within projects. By default, they are fully isolated and are not shared with other projects. OpenStack Networking supports the following types of network isolation and overlay technologies.

Flat All instances reside on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation takes place.

VLAN Networking allows users to 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, and other networking infrastructure on the same layer 2 VLAN.

GRE and VXLAN VXLAN and GRE are encapsulation protocols that create overlay networks to activate and control communication between compute instances. A Networking router is required to allow traffic to flow 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.



Subnets

A block of IP addresses and associated configuration state. This is also known as the native IPAM (IP Address Management) provided by the networking service for both project and provider networks. Subnets are used to allocate IP addresses when new ports are created on a network.

Subnet pools

End users normally can create subnets with any valid IP addresses without other restrictions. However, in some cases, it is nice for the admin or the project to pre-define a pool of addresses from which to create subnets with automatic allocation.

Using subnet pools constrains what addresses can be used by requiring that every subnet be within the defined pool. It also prevents address reuse or overlap by two subnets from the same pool.

See *Subnet pools* for more information.

Ports

A port is a connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. The port also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

Routers

Routers provide virtual layer-3 services such as routing and NAT between self-service and provider networks or among self-service networks belonging to a project. The Networking service uses a layer-3 agent to manage routers via namespaces.

Security groups

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`.

Each project contains a `default` security group that allows all egress traffic and denies all ingress traffic. You can change the rules in the `default` security group. If you launch an instance without specifying a security group, the `default` security group automatically applies to it. Similarly, if you create a port without specifying a security group, the `default` security group automatically applies to it.

Note: If you use the metadata service, removing the default egress rules denies access to TCP port 80 on 169.254.169.254, thus preventing instances from retrieving metadata.

Security group rules are stateful. Thus, allowing ingress TCP port 22 for secure shell automatically creates rules that allow return egress traffic and ICMP error messages involving those TCP connections.

By default, all security groups contain a series of basic (sanity) and anti-spoofing rules that perform the following actions:

- Allow egress traffic only if it uses the source MAC and IP addresses of the port for the instance, source MAC and IP combination in `allowed-address-pairs`, or valid MAC address (port or `allowed-address-pairs`) and associated EUI64 link-local IPv6 address.
- Allow egress DHCP discovery and request messages that use the source MAC address of the port for the instance and the unspecified IPv4 address (0.0.0.0).
- Allow ingress DHCP and DHCPv6 responses from the DHCP server on the subnet so instances can acquire IP addresses.
- Deny egress DHCP and DHCPv6 responses to prevent instances from acting as DHCP(v6) servers.
- Allow ingress/egress ICMPv6 MLD, neighbor solicitation, and neighbor discovery messages so instances can discover neighbors and join multicast groups.
- Deny egress ICMPv6 router advertisements to prevent instances from acting as IPv6 routers and forwarding IPv6 traffic for other instances.

- Allow egress ICMPv6 MLD reports (v1 and v2) and neighbor solicitation messages that use the source MAC address of a particular instance and the unspecified IPv6 address (::). Duplicate address detection (DAD) relies on these messages.
- Allow egress non-IP traffic from the MAC address of the port for the instance and any additional MAC addresses in `allowed-address-pairs` on the port for the instance.

Although non-IP traffic, security groups do not implicitly allow all ARP traffic. Separate ARP filtering rules prevent instances from using ARP to intercept traffic for another instance. You cannot disable or remove these rules.

You can disable security groups including basic and anti-spoofing rules by setting the port attribute `port_security_enabled` to `False`.

Extensions

The OpenStack Networking service 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.

DHCP

The optional DHCP service manages IP addresses for instances on provider and self-service networks. The Networking service implements the DHCP service using an agent that manages `qdhcp` namespaces and the `dnsmasq` service.

Metadata

The optional metadata service provides an API for instances to obtain metadata such as SSH keys.

Service and component hierarchy

Server

- Provides API, manages database, etc.

Plug-ins

- Manages agents

Agents

- Provides layer 2/3 connectivity to instances
- Handles physical-virtual network transition
- Handles metadata, etc.

Layer 2 (Ethernet and Switching)

- Linux Bridge
- OVS

Layer 3 (IP and Routing)

- L3
- DHCP

Miscellaneous

- Metadata

Services

Routing services

VPNaaS

The Virtual Private Network-as-a-Service (VPNaaS) is a neutron extension that introduces the VPN feature set.

LBaaS

The Load-Balancer-as-a-Service (LBaaS) API provisions and configures load balancers. The reference implementation is based on the HAProxy software load balancer. See the Octavia project for more information.

FWaaS

The Firewall-as-a-Service (FWaaS) API allows to apply firewalls to OpenStack objects such as projects, routers, and router ports.

8.1.7 Firewall-as-a-Service (FWaaS)

The Firewall-as-a-Service (FWaaS) plug-in applies firewalls to OpenStack objects such as projects, routers, and router ports.

The central concepts with OpenStack firewalls are the notions of a firewall policy and a firewall rule. A policy is an ordered collection of rules. A rule specifies a collection of attributes (such as port ranges, protocol, and IP addresses) that constitute match criteria and an action to take (allow or deny) on matched traffic. A policy can be made public, so it can be shared across projects.

Firewalls are implemented in various ways, depending on the driver used. For example, an iptables driver implements firewalls using iptable rules. An OpenVSwitch driver implements firewall rules using flow entries in flow tables. A Cisco firewall driver manipulates NSX devices.

FWaaS v2

The newer FWaaS implementation, v2, provides a much more granular service. The notion of a firewall has been replaced with firewall group to indicate that a firewall consists of two policies: an ingress policy and an egress policy. A firewall group is applied not at the router level (all ports on a router) but at the port level. Currently, router ports can be specified. For Ocata, VM ports can also be specified.

FWaaS v1

FWaaS v1 was deprecated in the Newton cycle and removed entirely in the Stein cycle.

FWaaS Feature Matrix

The following table shows FWaaS v2 features.

Feature	Supported
Supports L3 firewalling for routers	NO*
Supports L3 firewalling for router ports	YES
Supports L2 firewalling (VM ports)	YES
CLI support	YES
Horizon support	NO

* A firewall group can be applied to all ports on a given router in order to effect this.

For further information, see the FWaaS v2 configuration guide.

8.2 Configuration

8.2.1 Services and agents

A usual neutron setup consists of multiple services and agents running on one or multiple nodes (though some setups may not need any agents). Each of these services provide some of the networking or API services. Among those of special interest are:

1. The neutron-server that provides API endpoints and serves as a single point of access to the database. It usually runs on the controller nodes.
2. Layer2 agent that can utilize Open vSwitch, Linux Bridge or other vendor-specific technology to provide network segmentation and isolation for project networks. The L2 agent should run on every node where it is deemed responsible for wiring and securing virtual interfaces (usually both compute and network nodes).
3. Layer3 agent that runs on network node and provides east-west and north-south routing plus some advanced services such as FWaaS or VPNaaS.

Configuration options

The neutron configuration options are segregated between neutron-server and agents. Both services and agents may load the main `neutron.conf` since this file should contain the oslo.messaging configuration for internal neutron RPCs and may contain host specific configuration, such as file paths. The `neutron.conf` contains the database, keystone, nova credentials, and endpoints strictly for neutron-server to use.

In addition, neutron-server may load a plugin-specific configuration file, yet the agents should not. As the plugin configuration is primarily site wide options and the plugin provides the persistence layer for neutron, agents should be instructed to act upon these values through RPC.

Each individual agent may have its own configuration file. This file should be loaded after the main `neutron.conf` file, so the agent configuration takes precedence. The agent-specific configuration may contain configurations which vary between hosts in a neutron deployment such as the `local_ip` for an L2 agent. If any agent requires access to additional external services beyond the neutron RPC, those endpoints should be defined in the agent-specific configuration file (for example, nova metadata for metadata agent).

External processes run by agents

Some neutron agents, like DHCP, Metadata or L3, often run external processes to provide some of their functionalities. It may be keepalived, dnsmasq, haproxy or some other process. Neutron agents are responsible for spawning and killing such processes when necessary. By default, to kill such processes, agents use a simple `kill` command, but in some cases, like for example when those additional services are running inside containers, it may be not a good solution. To address this problem, operators should use the `AGENT` config group option `kill_scripts_path` to configure a path to where `kill scripts` for such processes live. By default, it is set to `/etc/neutron/kill_scripts/`. If option `kill_scripts_path` is changed in the config to the different location, `exec_dirs` in `/etc/rootwrap.conf` should be changed accordingly. If `kill_scripts_path` is set, every time neutron has to kill a process, for example `dnsmasq`, it will look in this directory for a file with the name `<process_name>-kill`. So for `dnsmasq` process it will look for a `dnsmasq-kill` script. If such a file exists there, it will be called instead of using the `kill` command.

Kill scripts are called with two parameters:

```
<process>-kill <sig> <pid>
```

where: `<sig>` is the signal, same as with the `kill` command, for example `9` or `SIGKILL`; and `<pid>` is pid of the process to kill.

This external script should then handle killing of the given process as neutron will not call the `kill` command for it anymore.

8.2.2 ML2 plug-in

Architecture

The Modular Layer 2 (ML2) neutron plug-in is a framework allowing OpenStack Networking to simultaneously use the variety of layer 2 networking technologies found in complex real-world data centers. The ML2 framework distinguishes between the two kinds of drivers that can be configured:

- Type drivers

  Define how an OpenStack network is technically realized. Example: VXLAN

  Each available network type is managed by an ML2 type driver. Type drivers maintain any needed type-specific network state. They validate the type specific information for provider networks and are responsible for the allocation of a free segment in project networks.

- Mechanism drivers

  Define the mechanism to access an OpenStack network of a certain type. Example: Open vSwitch mechanism driver.

  The mechanism driver is responsible for taking the information established by the type driver and ensuring that it is properly applied given the specific networking mechanisms that have been enabled.

  Mechanism drivers can utilize L2 agents (via RPC) and/or interact directly with external devices or controllers.

Multiple mechanism and type drivers can be used simultaneously to access different ports of the same virtual network.

Todo: Picture showing relationships

ML2 driver support matrix

Table 1: Mechanism drivers and L2 agents

type driver / mech driver	Flat	VLAN	VXLAN	GRE
Open vSwitch	yes	yes	yes	yes
Linux bridge	yes	yes	yes	no
SRIOV	yes	yes	no	no
MacVTap	yes	yes	no	no
L2 population	no	no	yes	yes

Note: L2 population is a special mechanism driver that optimizes BUM (Broadcast, unknown destination address, multicast) traffic in the overlay networks VXLAN and GRE. It needs to be used in conjunction with either the Linux bridge or the Open vSwitch mechanism driver and cannot be used as standalone mechanism driver. For more information, see the *Mechanism drivers* section below.

Configuration

Network type drivers

To enable type drivers in the ML2 plug-in. Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file:

```
[ml2]
type_drivers = flat,vlan,vxlan,gre
```

Note: For more detailssee the Bug 1567792.

For more details, see the Networking configuration options of Configuration Reference.

The following type drivers are available

- Flat
- VLAN
- GRE
- VXLAN

Provider network types

Provider networks provide connectivity like project networks. But only administrative (privileged) users can manage those networks because they interface with the physical network infrastructure. More information about provider networks see *OpenStack Networking*.

- Flat

  The administrator needs to configure a list of physical network names that can be used for provider networks. For more details, see the related section in the Configuration Reference.

- VLAN

  The administrator needs to configure a list of physical network names that can be used for provider networks. For more details, see the related section in the Configuration Reference.

- GRE

  No additional configuration required.

- VXLAN

  The administrator can configure the VXLAN multicast group that should be used.

Note: VXLAN multicast group configuration is not applicable for the Open vSwitch agent.

As of today it is not used in the Linux bridge agent. The Linux bridge agent has its own agent specific configuration option. For more details, see the Bug 1523614.

Project network types

Project networks provide connectivity to instances for a particular project. Regular (non-privileged) users can manage project networks within the allocation that an administrator or operator defines for them. More information about project and provider networks see *OpenStack Networking*.

Project network configurations are made in the `/etc/neutron/plugins/ml2/ml2_conf.ini` configuration file on the neutron server:

- VLAN

  The administrator needs to configure the range of VLAN IDs that can be used for project network allocation. For more details, see the related section in the Configuration Reference.

- GRE

  The administrator needs to configure the range of tunnel IDs that can be used for project network allocation. For more details, see the related section in the Configuration Reference.

- VXLAN

  The administrator needs to configure the range of VXLAN IDs that can be used for project network allocation. For more details, see the related section in the Configuration Reference.

Note: Flat networks for project allocation are not supported. They only can exist as a provider network.

Mechanism drivers

To enable mechanism drivers in the ML2 plug-in, edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file on the neutron server:

```
[ml2]
mechanism_drivers = ovs,l2pop
```

Note: For more details, see the Bug 1567792.

For more details, see the Configuration Reference.

- Linux bridge

  No additional configurations required for the mechanism driver. Additional agent configuration is required. For details, see the related *L2 agent* section below.

- Open vSwitch

No additional configurations required for the mechanism driver. Additional agent configuration is required. For details, see the related *L2 agent* section below.

- SRIOV

  The SRIOV driver accepts all PCI vendor devices.

- MacVTap

  No additional configurations required for the mechanism driver. Additional agent configuration is required. Please see the related section.

- L2 population

  The administrator can configure some optional configuration options. For more details, see the related section in the Configuration Reference.

- Specialized

  – Open source

    External open source mechanism drivers exist as well as the neutron integrated reference implementations. Configuration of those drivers is not part of this document. For example:

    * OpenDaylight
    * OpenContrail

  – Proprietary (vendor)

    External mechanism drivers from various vendors exist as well as the neutron integrated reference implementations.

    Configuration of those drivers is not part of this document.

Supported VNIC types

The `vnic_type_prohibit_list` option is used to remove values from the mechanism drivers `supported_vnic_types` list.

Table 2: Mechanism drivers and supported VNIC types

mech driver / supported_vnic_types	supported VNIC types	prohibiting available
Linux bridge	normal	no
MacVTap	macvtap	no
Open vSwitch	normal, direct	yes (ovs_driver vnic_type_prohibit_list, see: Configuration Reference)
SRIOV	direct, macvtap, direct_physical	yes (sriov_driver vnic_type_prohibit_list, see: Configuration Reference)

Extension Drivers

The ML2 plug-in also supports extension drivers that allows other pluggable drivers to extend the core resources implemented in the ML2 plug-in (`networks`, `ports`, etc.). Examples of extension drivers include support for QoS, port security, etc. For more details see the `extension_drivers` configuration option in the Configuration Reference.

Agents

L2 agent

An L2 agent serves layer 2 (Ethernet) network connectivity to OpenStack resources. It typically runs on each Network Node and on each Compute Node.

- Open vSwitch agent

  The Open vSwitch agent configures the Open vSwitch to realize L2 networks for OpenStack resources.

  Configuration for the Open vSwitch agent is typically done in the `openvswitch_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

  For a detailed list of configuration options, see the related section in the Configuration Reference.

- Linux bridge agent

  The Linux bridge agent configures Linux bridges to realize L2 networks for OpenStack resources.

  Configuration for the Linux bridge agent is typically done in the `linuxbridge_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

  For a detailed list of configuration options, see the related section in the Configuration Reference.

- SRIOV Nic Switch agent

  The sriov nic switch agent configures PCI virtual functions to realize L2 networks for OpenStack instances. Network attachments for other resources like routers, DHCP, and so on are not supported.

  Configuration for the SRIOV nic switch agent is typically done in the `sriov_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

  For a detailed list of configuration options, see the related section in the Configuration Reference.

- MacVTap agent

  The MacVTap agent uses kernel MacVTap devices for realizing L2 networks for OpenStack instances. Network attachments for other resources like routers, DHCP, and so on are not supported.

  Configuration for the MacVTap agent is typically done in the `macvtap_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

  For a detailed list of configuration options, see the related section in the Configuration Reference.

L3 agent

The L3 agent offers advanced layer 3 services, like virtual Routers and Floating IPs. It requires an L2 agent running in parallel.

Configuration for the L3 agent is typically done in the `l3_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the Configuration Reference.

DHCP agent

The DHCP agent is responsible for DHCP (Dynamic Host Configuration Protocol) and RADVD (Router Advertisement Daemon) services. It requires a running L2 agent on the same node.

Configuration for the DHCP agent is typically done in the `dhcp_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the Configuration Reference.

Metadata agent

The Metadata agent allows instances to access cloud-init meta data and user data via the network. It requires a running L2 agent on the same node.

Configuration for the Metadata agent is typically done in the `metadata_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the Configuration Reference.

L3 metering agent

The L3 metering agent enables layer3 traffic metering. It requires a running L3 agent on the same node.

Configuration for the L3 metering agent is typically done in the `metering_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the Configuration Reference.

Security

L2 agents support some important security configurations.

- Security Groups

  For more details, see the related section in the Configuration Reference.

- Arp Spoofing Prevention

  Configured in the *L2 agent* configuration.

Reference implementations

Overview

In this section, the combination of a mechanism driver and an L2 agent is called reference implementation. The following table lists these implementations:

Table 3: Mechanism drivers and L2 agents

Mechanism Driver	L2 agent
Open vSwitch	Open vSwitch agent
Linux bridge	Linux bridge agent
SRIOV	SRIOV nic switch agent
MacVTap	MacVTap agent
L2 population	Open vSwitch agent, Linux bridge agent

The following tables shows which reference implementations support which non-L2 neutron agents:

Table 4: Reference implementations and other agents

Reference Implementation	L3 agent	DHCP agent	Metadata agent	L3 Metering agent
Open vSwitch & Open vSwitch agent	yes	yes	yes	yes
Linux bridge & Linux bridge agent	yes	yes	yes	yes
SRIOV & SRIOV nic switch agent	no	no	no	no
MacVTap & MacVTap agent	no	no	no	no

Note: L2 population is not listed here, as it is not a standalone mechanism. If other agents are supported depends on the conjunctive mechanism driver that is used for binding a port.

More information about L2 population see the OpenStack Manuals.

Buying guide

This guide characterizes the L2 reference implementations that currently exist.

- Open vSwitch mechanism and Open vSwitch agent

  Can be used for instance network attachments as well as for attachments of other network resources like routers, DHCP, and so on.

- Linux bridge mechanism and Linux bridge agent

  Can be used for instance network attachments as well as for attachments of other network resources like routers, DHCP, and so on.

- SRIOV mechanism driver and SRIOV NIC switch agent

  Can only be used for instance network attachments (device_owner = compute).

Is deployed besides an other mechanism driver and L2 agent such as OVS or Linux bridge. It offers instances direct access to the network adapter through a PCI Virtual Function (VF). This gives an instance direct access to hardware capabilities and high performance networking.

The cloud consumer can decide via the neutron APIs VNIC_TYPE attribute, if an instance gets a normal OVS port or an SRIOV port.

Due to direct connection, some features are not available when using SRIOV. For example, DVR, security groups, migration.

For more information see the *SR-IOV*.

- MacVTap mechanism driver and MacVTap agent

  Can only be used for instance network attachments (device_owner = compute) and not for attachment of other resources like routers, DHCP, and so on.

  It is positioned as alternative to Open vSwitch or Linux bridge support on the compute node for internal deployments.

  MacVTap offers a direct connection with very little overhead between instances and down to the adapter. You can use MacVTap agent on the compute node when you require a network connection that is performance critical. It does not require specific hardware (like with SRIOV).

  Due to the direct connection, some features are not available when using it on the compute node. For example, DVR, security groups and arp-spoofing protection.

8.2.3 Address scopes

Address scopes build from subnet pools. While subnet pools provide a mechanism for controlling the allocation of addresses to subnets, address scopes show where addresses can be routed between networks, preventing the use of overlapping addresses in any two subnets. Because all addresses allocated in the address scope do not overlap, neutron routers do not NAT between your projects network and your external network. As long as the addresses within an address scope match, the Networking service performs simple routing between networks.

Accessing address scopes

Anyone with access to the Networking service can create their own address scopes. However, network administrators can create shared address scopes, allowing other projects to create networks within that address scope.

Access to addresses in a scope are managed through subnet pools. Subnet pools can either be created in an address scope, or updated to belong to an address scope.

With subnet pools, all addresses in use within the address scope are unique from the point of view of the address scope owner. Therefore, add more than one subnet pool to an address scope if the pools have different owners, allowing for delegation of parts of the address scope. Delegation prevents address overlap across the whole scope. Otherwise, you receive an error if two pools have the same address ranges.

Each router interface is associated with an address scope by looking at subnets connected to the network. When a router connects to an external network with matching address scopes, network traffic routes between without Network address translation (NAT). The router marks all traffic connections originating from each interface with its corresponding address scope. If traffic leaves an interface in the wrong scope, the router blocks the traffic.

Backwards compatibility

Networks created before the Mitaka release do not contain explicitly named address scopes, unless the network contains subnets from a subnet pool that belongs to a created or updated address scope. The Networking service preserves backwards compatibility with pre-Mitaka networks through special address scope properties so that these networks can perform advanced routing:

1. Unlimited address overlap is allowed.
2. Neutron routers, by default, will NAT traffic from internal networks to external networks.
3. Pre-Mitaka address scopes are not visible through the API. You cannot list address scopes or show details. Scopes exist implicitly as a catch-all for addresses that are not explicitly scoped.

Create shared address scopes as an administrative user

This section shows how to set up shared address scopes to allow simple routing for project networks with the same subnet pools.

Note: Irrelevant fields have been trimmed from the output of these commands for brevity.

---

1. Create IPv6 and IPv4 address scopes:

```
$ openstack address scope create --share --ip-version 6 address-scope-
↪ip6

+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| headers    |                                      |
| id         | 28424dfc-9abd-481b-afa3-1da97a8fead7 |
| ip_version | 6                                    |
| name       | address-scope-ip6                    |
| project_id | 098429d072d34d3596c88b7dbf7e91b6     |
| shared     | True                                 |
+------------+--------------------------------------+
```

```
$ openstack address scope create --share --ip-version 4 address-scope-
↪ip4

+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| headers    |                                      |
| id         | 3193bd62-11b5-44dc-acf8-53180f21e9f2 |
| ip_version | 4                                    |
| name       | address-scope-ip4                    |
| project_id | 098429d072d34d3596c88b7dbf7e91b6     |
| shared     | True                                 |
+------------+--------------------------------------+
```

2. Create subnet pools specifying the name (or UUID) of the address scope that the subnet pool belongs to. If you have existing subnet pools, use the **openstack subnet pool set** command to put them in a new address scope:

```
$ openstack subnet pool create --address-scope address-scope-ip6 \
--share --pool-prefix 2001:db8:a583::/48 --default-prefix-length 64 \
subnet-pool-ip6
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | 28424dfc-9abd-481b-afa3-1da97a8fead7 |
| created_at        | 2016-12-13T22:53:30Z                 |
| default_prefixlen | 64                                   |
| default_quota     | None                                 |
| description       |                                      |
| id                | a59ff52b-0367-41ff-9781-6318b927dd0e |
| ip_version        | 6                                    |
| is_default        | False                                |
| max_prefixlen     | 128                                  |
| min_prefixlen     | 64                                   |
| name              | subnet-pool-ip6                      |
| prefixes          | 2001:db8:a583::/48                   |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 1                                    |
| shared            | True                                 |
| tags              | []                                   |
| updated_at        | 2016-12-13T22:53:30Z                 |
+-------------------+--------------------------------------+
```

```
$ openstack subnet pool create --address-scope address-scope-ip4 \
--share --pool-prefix 203.0.113.0/24 --default-prefix-length 26 \
subnet-pool-ip4
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | 3193bd62-11b5-44dc-acf8-53180f21e9f2 |
| created_at        | 2016-12-13T22:55:09Z                 |
| default_prefixlen | 26                                   |
| default_quota     | None                                 |
| description       |                                      |
| id                | d02af70b-d622-426f-8e60-ed9df2a8301f |
| ip_version        | 4                                    |
| is_default        | False                                |
| max_prefixlen     | 32                                   |
| min_prefixlen     | 8                                    |
| name              | subnet-pool-ip4                      |
| prefixes          | 203.0.113.0/24                       |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 1                                    |
| shared            | True                                 |
| tags              | []                                   |
| updated_at        | 2016-12-13T22:55:09Z                 |
+-------------------+--------------------------------------+
```

3. Make sure that subnets on an external network are created from the subnet pools created above:

```
$ openstack subnet show ipv6-public-subnet
+-------------------+------------------------------------------+
| Field             | Value                                    |
+-------------------+------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| allocation_pools  | 2001:db8:a583::2-2001:db8:a583:0:ffff:ff |
|                   | ff:ffff:ffff                             |
| cidr              | 2001:db8:a583::/64                       |
| created_at        | 2016-12-10T21:36:04Z                     |
| description       |                                          |
| dns_nameservers   |                                          |
| enable_dhcp       | False                                    |
| gateway_ip        | 2001:db8:a583::1                         |
| host_routes       |                                          |
| id                | b333bf5a-758c-4b3f-97ec-5f12d9bfceb7     |
| ip_version        | 6                                        |
| ipv6_address_mode | None                                     |
| ipv6_ra_mode      | None                                     |
| name              | ipv6-public-subnet                       |
| network_id        | 05a8d31e-330b-4d96-a3fa-884b04abfa4c     |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6         |
| revision_number   | 2                                        |
| segment_id        | None                                     |
| service_types     |                                          |
| subnetpool_id     | a59ff52b-0367-41ff-9781-6318b927dd0e     |
| tags              | []                                       |
| updated_at        | 2016-12-10T21:36:04Z                     |
+-------------------+------------------------------------------+
```

```
$ openstack subnet show public-subnet
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.2-203.0.113.62             |
| cidr              | 203.0.113.0/26                       |
| created_at        | 2016-12-10T21:35:52Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | False                                |
| gateway_ip        | 203.0.113.1                          |
| host_routes       |                                      |
| id                | 7fd48240-3acc-4724-bc82-16c62857edec |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | public-subnet                        |
| network_id        | 05a8d31e-330b-4d96-a3fa-884b04abfa4c |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 2                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | d02af70b-d622-426f-8e60-ed9df2a8301f |
| tags              | []                                   |
| updated_at        | 2016-12-10T21:35:52Z                 |
+-------------------+--------------------------------------+
```

Routing with address scopes for non-privileged users

This section shows how non-privileged users can use address scopes to route straight to an external network without NAT.

1. Create a couple of networks to host subnets:

```
$ openstack network create network1
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-13T23:21:01Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | 1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | network1                             |
| port_security_enabled     | True                                 |
| project_id                | 098429d072d34d3596c88b7dbf7e91b6     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 94                                   |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2016-12-13T23:21:01Z                 |
+---------------------------+--------------------------------------+
```

```
$ openstack network create network2
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-13T23:21:45Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | 6c583603-c097-4141-9c5c-288b0e49c59f |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | network2                             |
| port_security_enabled     | True                                 |
| project_id                | 098429d072d34d3596c88b7dbf7e91b6     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 81                                   |
| revision_number           | 3                                    |
```

(continues on next page)

(continued from previous page)

```
| router:external            | Internal                             |
| shared                     | False                                |
| status                     | ACTIVE                               |
| subnets                    |                                      |
| tags                       | []                                   |
| updated_at                 | 2016-12-13T23:21:45Z                 |
+----------------------------+--------------------------------------+
```

2. Create a subnet not associated with a subnet pool or an address scope:

```
$ openstack subnet create --network network1 --subnet-range \
198.51.100.0/26 subnet-ip4-1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 198.51.100.2-198.51.100.62           |
| cidr              | 198.51.100.0/26                      |
| created_at        | 2016-12-13T23:24:16Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 198.51.100.1                         |
| headers           |                                      |
| host_routes       |                                      |
| id                | 66874039-d31b-4a27-85d7-14c89341bbb7 |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | subnet-ip4-1                         |
| network_id        | 1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0 |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| tags              | []                                   |
| updated_at        | 2016-12-13T23:24:16Z                 |
+-------------------+--------------------------------------+
```

```
$ openstack subnet create --network network1 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --ip-version 6 --subnet-range \
2001:db8:80d2:c4d3::/64 subnet-ip6-1
+-------------------+-----------------------------------------+
| Field             | Value                                   |
+-------------------+-----------------------------------------+
| allocation_pools  | 2001:db8:80d2:c4d3::2-2001:db8:80d2:c4d |
|                   | 3:ffff:ffff:ffff:ffff                   |
| cidr              | 2001:db8:80d2:c4d3::/64                 |
| created_at        | 2016-12-13T23:28:28Z                    |
| description       |                                         |
| dns_nameservers   |                                         |
| enable_dhcp       | True                                    |
| gateway_ip        | 2001:db8:80d2:c4d3::1                   |
| headers           |                                         |
| host_routes       |                                         |
| id                | a7551b23-2271-4a88-9c41-c84b048e0722    |
```

(continues on next page)

(continued from previous page)

```
| ip_version        | 6                                    |
| ipv6_address_mode | slaac                                |
| ipv6_ra_mode      | slaac                                |
| name              | subnet-ip6-1                         |
| network_id        | 1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0 |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| tags              | []                                   |
| updated_at        | 2016-12-13T23:28:28Z                 |
+-------------------+--------------------------------------+
```

3. Create a subnet using a subnet pool associated with an address scope from an external network:

```
$ openstack subnet create --subnet-pool subnet-pool-ip4 \
--network network2 subnet-ip4-2
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.2-203.0.113.62             |
| cidr              | 203.0.113.0/26                       |
| created_at        | 2016-12-13T23:32:12Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 203.0.113.1                          |
| headers           |                                      |
| host_routes       |                                      |
| id                | 12be8e8f-5871-4091-9e9e-4e0651b9677e |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | subnet-ip4-2                         |
| network_id        | 6c583603-c097-4141-9c5c-288b0e49c59f |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | d02af70b-d622-426f-8e60-ed9df2a8301f |
| tags              | []                                   |
| updated_at        | 2016-12-13T23:32:12Z                 |
+-------------------+--------------------------------------+
```

```
$ openstack subnet create --ip-version 6 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --subnet-pool subnet-pool-ip6 \
--network network2 subnet-ip6-2
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 2001:db8:a583::2-2001:db8:a583:0:fff |
|                   | f:ffff:ffff:ffff                     |
| cidr              | 2001:db8:a583::/64                   |
| created_at        | 2016-12-13T23:31:17Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
```

(continues on next page)

(continued from previous page)

```
| enable_dhcp       | True                                 |
| gateway_ip        | 2001:db8:a583::1                     |
| headers           |                                      |
| host_routes       |                                      |
| id                | b599c2be-e3cd-449c-ba39-3cfcc744c4be |
| ip_version        | 6                                    |
| ipv6_address_mode | slaac                                |
| ipv6_ra_mode      | slaac                                |
| name              | subnet-ip6-2                         |
| network_id        | 6c583603-c097-4141-9c5c-288b0e49c59f |
| project_id        | 098429d072d34d3596c88b7dbf7e91b6     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | a59ff52b-0367-41ff-9781-6318b927dd0e |
| tags              | []                                   |
| updated_at        | 2016-12-13T23:31:17Z                 |
+-------------------+--------------------------------------+
```

By creating subnets from scoped subnet pools, the network is associated with the address scope.

```
$ openstack network show network2
+---------------------------+------------------------------+
| Field                     | Value                        |
+---------------------------+------------------------------+
| admin_state_up            | UP                           |
| availability_zone_hints   |                              |
| availability_zones        | nova                         |
| created_at                | 2016-12-13T23:21:45Z         |
| description               |                              |
| id                        | 6c583603-c097-4141-9c5c-     |
|                           | 288b0e49c59f                 |
| ipv4_address_scope        | 3193bd62-11b5-44dc-          |
|                           | acf8-53180f21e9f2            |
| ipv6_address_scope        | 28424dfc-9abd-481b-          |
|                           | afa3-1da97a8fead7            |
| mtu                       | 1450                         |
| name                      | network2                     |
| port_security_enabled     | True                         |
| project_id                | 098429d072d34d3596c88b7dbf7e |
|                           | 91b6                         |
| provider:network_type     | vxlan                        |
| provider:physical_network | None                         |
| provider:segmentation_id  | 81                           |
| revision_number           | 10                           |
| router:external           | Internal                     |
| shared                    | False                        |
| status                    | ACTIVE                       |
| subnets                   | 12be8e8f-5871-4091-9e9e-     |
|                           | 4e0651b9677e, b599c2be-e3cd- |
|                           | 449c-ba39-3cfcc744c4be       |
| tags                      | []                           |
| updated_at                | 2016-12-13T23:32:12Z         |
+---------------------------+------------------------------+
```

4. Connect a router to each of the project subnets that have been created, for example, using a router called `router1`:

```
$ openstack router add subnet router1 subnet-ip4-1
$ openstack router add subnet router1 subnet-ip4-2
$ openstack router add subnet router1 subnet-ip6-1
$ openstack router add subnet router1 subnet-ip6-2
```

Checking connectivity

This example shows how to check the connectivity between networks with address scopes.

1. Launch two instances, `instance1` on `network1` and `instance2` on `network2`. Associate a floating IP address to both instances.
2. Adjust security groups to allow pings and SSH (both IPv4 and IPv6):

```
$ openstack server list
+--------------+------------+-----------------------------------------
↪---------------------------------+--------+----------+
| ID           | Name       | Networks                                ␣
↪                                 | Image  | Flavor   |
+--------------+------------+-----------------------------------------
↪---------------------------------+--------+----------+
| 97e49c8e-... | instance1 |␣
↪network1=2001:db8:80d2:c4d3:f816:3eff:fe52:b69f, 198.51.100.3, 203.
↪0.113.3| cirros | m1.tiny  |
| ceba9638-... | instance2 | network2=203.0.113.3,␣
↪2001:db8:a583:0:f816:3eff:fe42:1eeb, 203.0.113.4    | centos | m1.
↪small |
+--------------+------------+-----------------------------------------
↪---------------------------------+--------+----------+
```

Regardless of address scopes, the floating IPs can be pinged from the external network:

```
$ ping -c 1 203.0.113.3
1 packets transmitted, 1 received, 0% packet loss, time 0ms
$ ping -c 1 203.0.113.4
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

You can now ping `instance2` directly because `instance2` shares the same address scope as the external network:

Note: BGP routing can be used to automatically set up a static route for your instances.

```
# ip route add 203.0.113.0/26 via 203.0.113.2
$ ping -c 1 203.0.113.3
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
# ip route add 2001:db8:a583::/64 via 2001:db8::1
$ ping6 -c 1 2001:db8:a583:0:f816:3eff:fe42:1eeb
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

You cannot ping `instance1` directly because the address scopes do not match:

```
# ip route add 198.51.100.0/26 via 203.0.113.2
$ ping -c 1 198.51.100.3
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
# ip route add 2001:db8:80d2:c4d3::/64 via 2001:db8::1
$ ping6 -c 1 2001:db8:80d2:c4d3:f816:3eff:fe52:b69f
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

If the address scopes match between networks then pings and other traffic route directly through. If the scopes do not match between networks, the router either drops the traffic or applies NAT to cross scope boundaries.

8.2.4 Automatic allocation of network topologies

The auto-allocation feature introduced in Mitaka simplifies the procedure of setting up an external connectivity for end-users, and is also known as **Get Me A Network**.

Previously, a user had to configure a range of networking resources to boot a server and get access to the Internet. For example, the following steps are required:

- Create a network
- Create a subnet
- Create a router
- Uplink the router on an external network
- Downlink the router on the previously created subnet

These steps need to be performed on each logical segment that a VM needs to be connected to, and may require networking knowledge the user might not have.

This feature is designed to automate the basic networking provisioning for projects. The steps to provision a basic network are run during instance boot, making the networking setup hands-free.

To make this possible, provide a default external network and default subnetpools (one for IPv4, or one for IPv6, or one of each) so that the Networking service can choose what to do in lieu of input. Once these are in place, users can boot their VMs without specifying any networking details. The Compute service will then use this feature automatically to wire user VMs.

Enabling the deployment for auto-allocation

To use this feature, the neutron service must have the following extensions enabled:

- `auto-allocated-topology`
- `subnet_allocation`
- `external-net`
- `router`

Before the end-user can use the auto-allocation feature, the operator must create the resources that will be used for the auto-allocated network topology creation. To perform this task, proceed with the following steps:

1. Set up a default external network

   Setting up an external network is described in OpenStack Networking Guide. Assuming the external network to be used for the auto-allocation feature is named `public`, make it the `default` external network with the following command:

   ```
   $ openstack network set public --default
   ```

   Note: The flag `--default` (and `--no-default` flag) is only effective with external networks and has no effects on regular (or internal) networks.

2. Create default subnetpools

   The auto-allocation feature requires at least one default subnetpool. One for IPv4, or one for IPv6, or one of each.

   ```
   $ openstack subnet pool create --share --default \
     --pool-prefix 192.0.2.0/24 --default-prefix-length 26 \
     shared-default

     +-------------------+--------------------------------------+
     | Field             | Value                                |
     +-------------------+--------------------------------------+
     | address_scope_id  | None                                 |
     | created_at        | 2017-01-12T15:10:34Z                 |
     | default_prefixlen | 26                                   |
     | default_quota     | None                                 |
     | description       |                                      |
     | headers           |                                      |
     | id                | b41b7b9c-de57-4c19-b1c5-731985bceb7f |
     | ip_version        | 4                                    |
     | is_default        | True                                 |
     | max_prefixlen     | 32                                   |
     | min_prefixlen     | 8                                    |
     | name              | shared-default                       |
     | prefixes          | 192.0.2.0/24                         |
     | project_id        | 86acdbd1d72745fd8e8320edd7543400     |
     | revision_number   | 1                                    |
     | shared            | True                                 |
     | tags              | []                                   |
     | updated_at        | 2017-01-12T15:10:34Z                 |
     +-------------------+--------------------------------------+

   $ openstack subnet pool create --share --default \
     --pool-prefix 2001:db8:8000::/48 --default-prefix-length 64 \
     default-v6

   +-------------------+--------------------------------------+
   | Field             | Value                                |
   +-------------------+--------------------------------------+
   | address_scope_id  | None                                 |
   | created_at        | 2017-01-12T15:14:35Z                 |
   | default_prefixlen | 64                                   |
   | default_quota     | None                                 |
   | description       |                                      |
   ```

   (continues on next page)

(continued from previous page)

```
| headers           |                                      |
| id                | 6f387016-17f0-4564-96ad-e34775b6ea14 |
| ip_version        | 6                                    |
| is_default        | True                                 |
| max_prefixlen     | 128                                  |
| min_prefixlen     | 64                                   |
| name              | default-v6                           |
| prefixes          | 2001:db8:8000::/48                   |
| project_id        | 86acdbd1d72745fd8e8320edd7543400     |
| revision_number   | 1                                    |
| shared            | True                                 |
| tags              | []                                   |
| updated_at        | 2017-01-12T15:14:35Z                 |
+-------------------+--------------------------------------+
```

Get Me A Network

In a deployment where the operator has set up the resources as described above, they can get their auto-allocated network topology as follows:

```
$ openstack network auto allocated topology create --or-show
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| id         | a380c780-d6cd-4510-a4c0-1a6ec9b85a29 |
| name       | None                                 |
| project_id | cfd1889ac7d64ad891d4f20aef9f8d7c     |
+------------+--------------------------------------+
```

Note: When the `--or-show` option is used the command returns the topology information if it already exists.

Operators (and users with admin role) can get the auto-allocated topology for a project by specifying the project ID:

```
$ openstack network auto allocated topology create --project \
  cfd1889ac7d64ad891d4f20aef9f8d7c --or-show
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| id         | a380c780-d6cd-4510-a4c0-1a6ec9b85a29 |
| name       | None                                 |
| project_id | cfd1889ac7d64ad891d4f20aef9f8d7c     |
+------------+--------------------------------------+
```

The ID returned by this command is a network which can be used for booting a VM.

```
$ openstack server create --flavor m1.small --image \
  cirros-0.3.5-x86_64-uec --nic \
  net-id=8b835bfb-cae2-4acc-b53f-c16bb5f9a7d0 vm1
```

The auto-allocated topology for a user never changes. In practice, when a user boots a server omitting the `--nic` option, and there is more than one network available, the Compute service will invoke the API behind `auto allocated topology create`, fetch the network UUID, and pass it on during the boot process.

Validating the requirements for auto-allocation

To validate that the required resources are correctly set up for auto-allocation, without actually provisioning anything, use the `--check-resources` option:

```
$ openstack network auto allocated topology create --check-resources
Deployment error: No default router:external network.

$ openstack network set public --default

$ openstack network auto allocated topology create --check-resources
Deployment error: No default subnetpools defined.

$ openstack subnet pool set shared-default --default

$ openstack network auto allocated topology create --check-resources
+---------+-------+
| Field   | Value |
+---------+-------+
| dry-run | pass  |
+---------+-------+
```

The validation option behaves identically for all users. However, it is considered primarily an admin or service utility since it is the operator who must set up the requirements.

Project resources created by auto-allocation

The auto-allocation feature creates one network topology in every project where it is used. The auto-allocated network topology for a project contains the following resources:

Resource	Name
network	`auto_allocated_network`
subnet (IPv4)	`auto_allocated_subnet_v4`
subnet (IPv6)	`auto_allocated_subnet_v6`
router	`auto_allocated_router`

Compatibility notes

Nova uses the `auto allocated topology` feature with API micro version 2.37 or later. This is because, unlike the neutron feature which was implemented in the Mitaka release, the integration for nova was completed during the Newton release cycle. Note that the CLI option `--nic` can be omitted regardless of the microversion used as long as there is no more than one network available to the project, in which case nova fails with a 400 error because it does not know which network to use. Furthermore, nova does not start using the feature, regardless of whether or not a user requests micro version 2.37 or later, unless all of the `nova-compute` services are running Newton-level code.

8.2.5 Availability zones

An availability zone groups network nodes that run services like DHCP, L3, FW, and others. It is defined as an agents attribute on the network node. This allows users to associate an availability zone with their resources so that the resources get high availability.

Use case

An availability zone is used to make network resources highly available. The operators group the nodes that are attached to different power sources under separate availability zones and configure scheduling for resources with high availability so that they are scheduled on different availability zones.

Required extensions

The core plug-in must support the `availability_zone` extension. The core plug-in also must support the `network_availability_zone` extension to schedule a network according to availability zones. The `Ml2Plugin` supports it. The router service plug-in must support the `router_availability_zone` extension to schedule a router according to the availability zones. The `L3RouterPlugin` supports it.

```
$ openstack extension list --network -c Alias -c Name
+---------------------------+---------------------------+
| Name                      | Alias                     |
+---------------------------+---------------------------+
...
| Network Availability Zone | network_availability_zone |
...
| Availability Zone         | availability_zone         |
...
| Router Availability Zone  | router_availability_zone  |
...
+---------------------------+---------------------------+
```

Availability zone of agents

The `availability_zone` attribute can be defined in `dhcp-agent` and `l3-agent`. To define an availability zone for each agent, set the value into `[AGENT]` section of `/etc/neutron/dhcp_agent.ini` or `/etc/neutron/l3_agent.ini`:

```
[AGENT]
availability_zone = zone-1
```

To confirm the agents availability zone:

```
$ openstack network agent show 116cc128-4398-49af-a4ed-3e95494cd5fc
+---------------------+---------------------------------------------------+
| Field               | Value                                             |
+---------------------+---------------------------------------------------+
| admin_state_up      | UP                                                |
| agent_type          | DHCP agent                                        |
| alive               | True                                              |
```

(continues on next page)

(continued from previous page)

```
| availability_zone   | zone-1                                          |
| binary              | neutron-dhcp-agent                              |
| configurations      | dhcp_driver='neutron.agent.linux.dhcp.Dnsmasq', |
|                     | dhcp_lease_duration='86400',                    |
|                     | log_agent_heartbeats='False', networks='2',     |
|                     | notifies_port_ready='True', ports='6', subnets='4 |
| created_at          | 2016-12-14 00:25:54                             |
| description         | None                                            |
| heartbeat_timestamp | 2016-12-14 06:20:24                             |
| host                | ankur-desktop                                   |
| id                  | 116cc128-4398-49af-a4ed-3e95494cd5fc            |
| started_at          | 2016-12-14 00:25:54                             |
| topic               | dhcp_agent                                      |
+---------------------+-------------------------------------------------+

$ openstack network agent show 9632309a-2aa4-4304-8603-c4de02c4a55f
+---------------------+-----------------------------------------------+
| Field               | Value                                         |
+---------------------+-----------------------------------------------+
| admin_state_up      | UP                                            |
| agent_type          | L3 agent                                      |
| alive               | True                                          |
| availability_zone   | zone-1                                        |
| binary              | neutron-l3-agent                              |
| configurations      | agent_mode='legacy', ex_gw_ports='2',         |
|                     | floating_ips='0',                             |
|                     | handle_internal_only_routers='True',          |
|                     | interface_driver='openvswitch', interfaces='4', |
|                     | log_agent_heartbeats='False', routers='2'     |
| created_at          | 2016-12-14 00:25:58                           |
| description         | None                                          |
| heartbeat_timestamp | 2016-12-14 06:20:28                           |
| host                | ankur-desktop                                 |
| id                  | 9632309a-2aa4-4304-8603-c4de02c4a55f          |
| started_at          | 2016-12-14 00:25:58                           |
| topic               | l3_agent                                      |
+---------------------+-----------------------------------------------+
```

Availability zone related attributes

The following attributes are added into network and router:

Attribute name	Access	Required	Input type	Description
availability_zone_hints	RW(POST only)	No	list of string	availability zone candidates for the resource
availability_zones	RO	N/A	list of string	availability zones for the resource

Use `availability_zone_hints` to specify the zone in which the resource is hosted:

```
$ openstack network create --availability-zone-hint zone-1 \
--availability-zone-hint zone-2 net1
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   | zone-1                               |
|                           | zone-2                               |
| availability_zones        |                                      |
| created_at                | 2016-12-14T06:23:36Z                 |
| 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                | 2016-12-14T06:23:37Z                 |
+---------------------------+--------------------------------------+
```

```
$ openstack router create --ha --availability-zone-hint zone-1 \
--availability-zone-hint zone-2 router1
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints | zone-1                               |
|                         | zone-2                               |
| availability_zones      |                                      |
| created_at              | 2016-12-14T06:25:40Z                 |
| 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              | 2016-12-14T06:25:40Z                 |
+-------------------------+--------------------------------------+
```

Availability zone is selected from `default_availability_zones` in `/etc/neutron/neutron.conf` if a resource is created without `availability_zone_hints`:

```
default_availability_zones = zone-1,zone-2
```

To confirm the availability zone defined by the system:

```
$ openstack availability zone list
+-----------+-------------+
| Zone Name | Zone Status |
+-----------+-------------+
| zone-1    | available   |
| zone-2    | available   |
| zone-1    | available   |
| zone-2    | available   |
+-----------+-------------+
```

Look at the `availability_zones` attribute of each resource to confirm in which zone the resource is hosted:

```
$ openstack network show net1
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   | zone-1                               |
|                           | zone-2                               |
| availability_zones        | zone-1                               |
|                           | zone-2                               |
| created_at                | 2016-12-14T06:23:36Z                 |
| 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                | 2016-12-14T06:23:37Z                 |
+---------------------------+--------------------------------------+
```

```
$ openstack router show router1
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
```

(continues on next page)

(continued from previous page)

```
| availability_zone_hints | zone-1                               |
|                         | zone-2                               |
| availability_zones      | zone-1                               |
|                         | zone-2                               |
| created_at              | 2016-12-14T06:25:40Z                 |
| 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              | 2016-12-14T06:25:40Z                 |
+-------------------------+--------------------------------------+
```

Note: The `availability_zones` attribute does not have a value until the resource is scheduled. Once the Networking service schedules the resource to zones according to `availability_zone_hints`, `availability_zones` shows in which zone the resource is hosted practically. The `availability_zones` may not match `availability_zone_hints`. For example, even if you specify a zone with `availability_zone_hints`, all agents of the zone may be dead before the resource is scheduled. In general, they should match, unless there are failures or there is no capacity left in the zone requested.

Availability zone aware scheduler

Network scheduler

Set `AZAwareWeightScheduler` to `network_scheduler_driver` in `/etc/neutron/neutron.conf` so that the Networking service schedules a network according to the availability zone:

```
network_scheduler_driver = neutron.scheduler.dhcp_agent_scheduler.
↪AZAwareWeightScheduler
dhcp_load_type = networks
```

The Networking service schedules a network to one of the agents within the selected zone as with `WeightScheduler`. In this case, scheduler refers to `dhcp_load_type` as well.

Router scheduler

Set `AZLeastRoutersScheduler` to `router_scheduler_driver` in file `/etc/neutron/neutron.conf` so that the Networking service schedules a router according to the availability zone:

```
router_scheduler_driver = neutron.scheduler.l3_agent_scheduler.
↪AZLeastRoutersScheduler
```

The Networking service schedules a router to one of the agents within the selected zone as with `LeastRouterScheduler`.

Achieving high availability with availability zone

Although, the Networking service provides high availability for routers and high availability and fault tolerance for networks DHCP services, availability zones provide an extra layer of protection by segmenting a Networking service deployment in isolated failure domains. By deploying HA nodes across different availability zones, it is guaranteed that network services remain available in face of zone-wide failures that affect the deployment.

This section explains how to get high availability with the availability zone for L3 and DHCP. You should naturally set above configuration options for the availability zone.

L3 high availability

Set the following configuration options in file `/etc/neutron/neutron.conf` so that you get L3 high availability.

```
l3_ha = True
max_l3_agents_per_router = 3
```

HA routers are created on availability zones you selected when creating the router.

DHCP high availability

Set the following configuration options in file `/etc/neutron/neutron.conf` so that you get DHCP high availability.

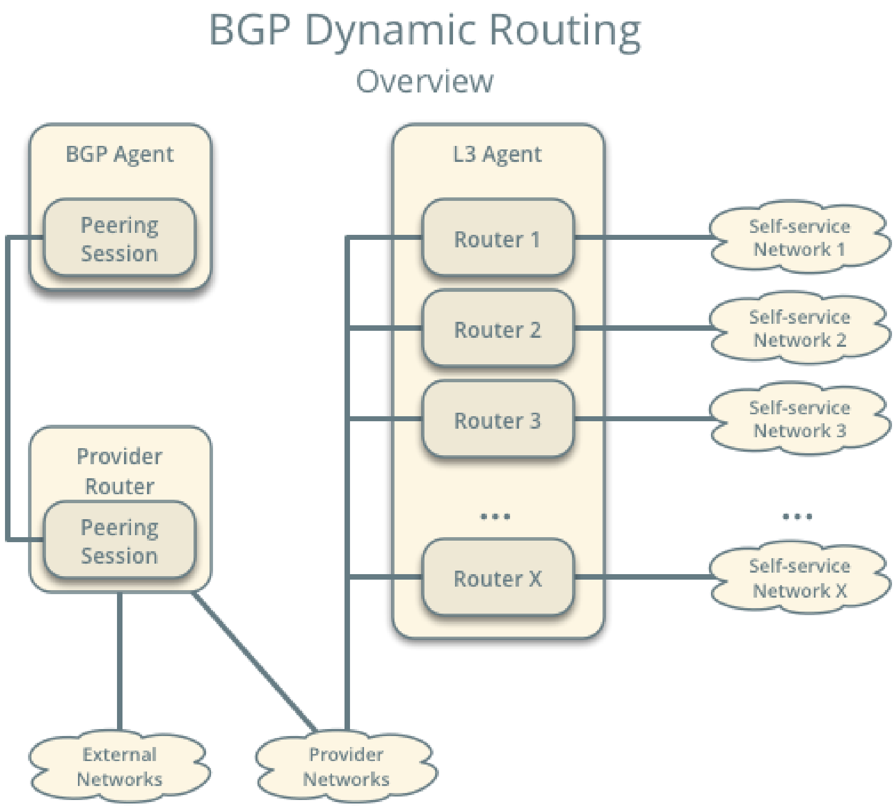
```
dhcp_agents_per_network = 2
```

DHCP services are created on availability zones you selected when creating the network.

8.2.6 BGP dynamic routing

BGP dynamic routing enables advertisement of self-service (private) network prefixes to physical network devices that support BGP such as routers, thus removing the conventional dependency on static routes. The feature relies on *address scopes* and requires knowledge of their operation for proper deployment.

BGP dynamic routing consists of a service plug-in and an agent. The service plug-in implements the Networking service extension and the agent manages BGP peering sessions. A cloud administrator creates and configures a BGP speaker using the CLI or API and manually schedules it to one or more hosts running the agent. Agents can reside on hosts with or without other Networking service agents. Prefix advertisement depends on the binding of external networks to a BGP speaker and the address scope of external and internal IP address ranges or subnets.



Note: Although self-service networks generally use private IP address ranges (RFC1918) for IPv4 subnets, BGP dynamic routing can advertise any IPv4 address ranges.

Example configuration

The example configuration involves the following components:

- One BGP agent.
- One address scope containing IP address range 203.0.113.0/24 for provider networks, and IP address ranges 192.0.2.0/25 and 192.0.2.128/25 for self-service networks.
- One provider network using IP address range 203.0.113.0/24.
- Three self-service networks.
  - Self-service networks 1 and 2 use IP address ranges inside of the address scope.
  - Self-service network 3 uses a unique IP address range 198.51.100.0/24 to demonstrate that the BGP speaker does not advertise prefixes outside of address scopes.
- Three routers. Each router connects one self-service network to the provider network.
  - Router 1 contains IP addresses 203.0.113.11 and 192.0.2.1

- Router 2 contains IP addresses 203.0.113.12 and 192.0.2.129
- Router 3 contains IP addresses 203.0.113.13 and 198.51.100.1

Note: The example configuration assumes sufficient knowledge about the Networking service, routing, and BGP. For basic deployment of the Networking service, consult one of the *Deployment examples*. For more information on BGP, see RFC 4271.

Controller node

- In the `neutron.conf` file, enable the conventional layer-3 and BGP dynamic routing service plug-ins:

```
[DEFAULT]
service_plugins = neutron_dynamic_routing.services.bgp.bgp_plugin.
↪BgpPlugin,neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
```

Agent nodes

- In the `bgp_dragent.ini` file:
  - Configure the driver.

```
[BGP]
bgp_speaker_driver = neutron_dynamic_routing.services.bgp.agent.
↪driver.os_ken.driver.OsKenBgpDriver
```

Note: The agent currently only supports the os-ken BGP driver.

  - Configure the router ID.

```
[BGP]
bgp_router_id = ROUTER_ID
```

Replace `ROUTER_ID` with a suitable unique 32-bit number, typically an IPv4 address on the host running the agent. For example, 192.0.2.2.

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of each BGP dynamic routing agent.

```
$ openstack network agent list --agent-type bgp
+--------------------------------------+---------------------------+--
↪----------+--------------------+-------+-------+---------------------
↪+
| ID                                   | Agent Type                |␣
↪Host       | Availability Zone | Alive | State | Binary               ␣
↪ |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+---------------------------+--
↪----------+-------------------+-------+-------+---------------------
↪+
| 37729181-2224-48d8-89ef-16eca8e2f77e | BGP dynamic routing agent |␣
↪controller | None              | :-)   | UP    | neutron-bgp-
↪dragent |
+--------------------------------------+---------------------------+--
↪----------+-------------------+-------+-------+---------------------
↪+
```

Create the address scope and subnet pools

1. Create an address scope. The provider (external) and self-service networks must belong to the same address scope for the agent to advertise those self-service network prefixes.

```
$ openstack address scope create --share --ip-version 4 bgp

+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| headers    |                                      |
| id         | f71c958f-dbe8-49a2-8fb9-19c5f52a37f1 |
| ip_version | 4                                    |
| name       | bgp                                  |
| project_id | 86acdbd1d72745fd8e8320edd7543400     |
| shared     | True                                 |
+------------+--------------------------------------+
```

2. Create subnet pools. The provider and self-service networks use different pools.

   - Create the provider network pool.

```
$ openstack subnet pool create --pool-prefix 203.0.113.0/24 \
  --address-scope bgp provider

+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | f71c958f-dbe8-49a2-8fb9-19c5f52a37f1 |
| created_at        | 2017-01-12T14:58:57Z                 |
| default_prefixlen | 8                                    |
| default_quota     | None                                 |
| description       |                                      |
| headers           |                                      |
| id                | 63532225-b9a0-445a-9935-20a15f9f68d1 |
| ip_version        | 4                                    |
| is_default        | False                                |
| max_prefixlen     | 32                                   |
| min_prefixlen     | 8                                    |
| name              | provider                             |
| prefixes          | 203.0.113.0/24                       |
| project_id        | 86acdbd1d72745fd8e8320edd7543400     |
| revision_number   | 1                                    |
| shared            | False                                |
```

(continues on next page)

(continued from previous page)

```
| tags              | []                                   |
| updated_at        | 2017-01-12T14:58:57Z                 |
+-------------------+--------------------------------------+
```

- Create the self-service network pool.

```
$ openstack subnet pool create --pool-prefix 192.0.2.0/25 \
  --pool-prefix 192.0.2.128/25 --address-scope bgp \
  --share selfservice

+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | f71c958f-dbe8-49a2-8fb9-19c5f52a37f1 |
| created_at        | 2017-01-12T15:02:31Z                 |
| default_prefixlen | 8                                    |
| default_quota     | None                                 |
| description       |                                      |
| headers           |                                      |
| id                | 8d8270b1-b194-4b7e-914c-9c741dcbd49b |
| ip_version        | 4                                    |
| is_default        | False                                |
| max_prefixlen     | 32                                   |
| min_prefixlen     | 8                                    |
| name              | selfservice                          |
| prefixes          | 192.0.2.0/25, 192.0.2.128/25         |
| project_id        | 86acdbd1d72745fd8e8320edd7543400     |
| revision_number   | 1                                    |
| shared            | True                                 |
| tags              | []                                   |
| updated_at        | 2017-01-12T15:02:31Z                 |
+-------------------+--------------------------------------+
```

Create the provider and self-service networks

1. Create the provider network.

```
$ openstack network create provider --external --provider-physical-
↪network \
  provider --provider-network-type flat
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-21T08:47:41Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | 190ca651-2ee3-4a4b-891f-dedda47974fe |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
```

(continues on next page)

(continued from previous page)

```
| mtu                       | 1450                                 |
| name                      | provider                             |
| port_security_enabled     | True                                 |
| project_id                | c961a8f6d3654657885226378ade8220     |
| provider:network_type     | flat                                 |
| provider:physical_network | provider                             |
| provider:segmentation_id  | 66                                   |
| revision_number           | 3                                    |
| router:external           | External                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2016-12-21T08:47:41Z                 |
+---------------------------+--------------------------------------+
```

2. Create a subnet on the provider network using an IP address range from the provider subnet pool.

```
$ openstack subnet create --subnet-pool provider \
  --prefix-length 24 --gateway 203.0.113.1 --network provider \
  --allocation-pool start=203.0.113.11,end=203.0.113.254 provider
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.11-203.0.113.254           |
| cidr              | 203.0.113.0/24                       |
| created_at        | 2016-03-17T23:17:16                  |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 203.0.113.1                          |
| host_routes       |                                      |
| id                | 8ed65d41-2b2a-4f3a-9f92-45adb266e01a |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | provider                             |
| network_id        | 68ec148c-181f-4656-8334-8f4eb148689d |
| project_id        | b3ac05ef10bf441fbf4aa17f16ae1e6d     |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | 3771c0e7-7096-46d3-a3bd-699c58e70259 |
| tags              |                                      |
| updated_at        | 2016-03-17T23:17:16                  |
+-------------------+--------------------------------------+
```

Note: The IP address allocation pool starting at `.11` improves clarity of the diagrams. You can safely omit it.

3. Create the self-service networks.

```
$ openstack network create selfservice1
Created a new network:
+---------------------------+--------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-21T08:49:38Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | 9d842606-ef3d-4160-9ed9-e03fa63aed96 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | selfservice1                         |
| port_security_enabled     | True                                 |
| project_id                | c961a8f6d3654657885226378ade8220     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 106                                  |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2016-12-21T08:49:38Z                 |
+---------------------------+--------------------------------------+

$ openstack network create selfservice2
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-21T08:50:05Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | f85639e1-d23f-438e-b2b1-f40570d86b1c |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | selfservice2                         |
| port_security_enabled     | True                                 |
| project_id                | c961a8f6d3654657885226378ade8220     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 21                                   |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2016-12-21T08:50:05Z                 |
+---------------------------+--------------------------------------+
```

(continues on next page)

(continued from previous page)

```
$ openstack network create selfservice3
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2016-12-21T08:50:35Z                 |
| description               |                                      |
| headers                   |                                      |
| id                        | eeccdb82-5cf4-4999-8ab3-e7dc99e7d43b |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | selfservice3                         |
| port_security_enabled     | True                                 |
| project_id                | c961a8f6d3654657885226378ade8220     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 86                                   |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2016-12-21T08:50:35Z                 |
+---------------------------+--------------------------------------+
```

4. Create a subnet on the first two self-service networks using an IP address range from the self-service subnet pool.

```
$ openstack subnet create --network selfservice1 --subnet-pool␣
↪selfservice \
  --prefix-length 25 selfservice1
+-------------------+---------------------------------------------------
↪---+
| Field             | Value                                             ␣
↪   |
+-------------------+---------------------------------------------------
↪---+
| allocation_pools  | 192.0.2.2-192.0.2.127                             ␣
↪   |
| cidr              | 192.0.2.0/25                                      ␣
↪   |
| created_at        | 2016-03-17T23:20:20                               ␣
↪   |
| description       |                                                   ␣
↪   |
| dns_nameservers   |                                                   ␣
↪   |
| enable_dhcp       | True                                              ␣
↪   |
| gateway_ip        | 198.51.100.1                                      ␣
↪   |
```

(continues on next page)

(continued from previous page)

```
| host_routes       |                                                 ␣
↪   |
| id                | 8edd3dc2-df40-4d71-816e-a4586d61c809            ␣
↪   |
| ip_version        | 4                                               ␣
↪   |
| ipv6_address_mode |                                                 ␣
↪   |
| ipv6_ra_mode      |                                                 ␣
↪   |
| name              | selfservice1                                    ␣
↪   |
| network_id        | be79de1e-5f56-11e6-9dfb-233e41cec48c            ␣
↪   |
| project_id        | b3ac05ef10bf441fbf4aa17f16ae1e6d                ␣
↪   |
| revision_number   | 1                                               ␣
↪   |
| subnetpool_id     | c7e9737a-cfd3-45b5-a861-d1cee1135a92            ␣
↪   |
| tags              | []                                              ␣
↪   |
| tenant_id         | b3ac05ef10bf441fbf4aa17f16ae1e6d                ␣
↪   |
| updated_at        | 2016-03-17T23:20:20                             ␣
↪   |
+-------------------+-------------------------------------------------
↪---+

$ openstack subnet create --network selfservice2 --subnet-pool␣
↪selfservice \
  --prefix-length 25 selfservice2
+-------------------+----------------------------------------------------+
| Field             | Value                                              |
+-------------------+----------------------------------------------------+
| allocation_pools  | 192.0.2.130-192.0.2.254                            |
| cidr              | 192.0.2.128/25                                     |
| created_at        | 2016-03-17T23:20:20                                |
| description       |                                                    |
| dns_nameservers   |                                                    |
| enable_dhcp       | True                                               |
| gateway_ip        | 192.0.2.129                                        |
| host_routes       |                                                    |
| id                | 8edd3dc2-df40-4d71-816e-a4586d61c809               |
| ip_version        | 4                                                  |
| ipv6_address_mode |                                                    |
| ipv6_ra_mode      |                                                    |
| name              | selfservice2                                       |
| network_id        | c1fd9846-5f56-11e6-a8ac-0f998d9cc0a2               |
| project_id        | b3ac05ef10bf441fbf4aa17f16ae1e6d                   |
| revision_number   | 1                                                  |
| subnetpool_id     | c7e9737a-cfd3-45b5-a861-d1cee1135a92               |
| tags              | []                                                 |
| tenant_id         | b3ac05ef10bf441fbf4aa17f16ae1e6d                   |
| updated_at        | 2016-03-17T23:20:20                                |
+-------------------+----------------------------------------------------+
```

5. Create a subnet on the last self-service network using an IP address range outside of the address scope.

```
$ openstack subnet create --network selfservice3 --prefix 198.51.100.
↪0/24 subnet3
+-------------------+---------------------------------------------------
↪---+
| Field             | Value                                             ␣
↪   |
+-------------------+---------------------------------------------------
↪---+
| allocation_pools  | 198.51.100.2-198.51.100.254                       ␣
↪   |
| cidr              | 198.51.100.0/24                                   ␣
↪   |
| created_at        | 2016-03-17T23:20:20                               ␣
↪   |
| description       |                                                   ␣
↪   |
| dns_nameservers   |                                                   ␣
↪   |
| enable_dhcp       | True                                              ␣
↪   |
| gateway_ip        | 198.51.100.1                                      ␣
↪   |
| host_routes       |                                                   ␣
↪   |
| id                | cd9f9156-5f59-11e6-aeec-172ec7ee939a              ␣
↪   |
| ip_version        | 4                                                 ␣
↪   |
| ipv6_address_mode |                                                   ␣
↪   |
| ipv6_ra_mode      |                                                   ␣
↪   |
| name              | selfservice3                                      ␣
↪   |
| network_id        | c283dc1c-5f56-11e6-bfb6-efc30e1eb73b              ␣
↪   |
| project_id        | b3ac05ef10bf441fbf4aa17f16ae1e6d                  ␣
↪   |
| revision_number   | 1                                                 ␣
↪   |
| subnetpool_id     |                                                   ␣
↪   |
| tags              | []                                                ␣
↪   |
| tenant_id         | b3ac05ef10bf441fbf4aa17f16ae1e6d                  ␣
↪   |
| updated_at        | 2016-03-17T23:20:20                               ␣
↪   |
+-------------------+---------------------------------------------------
↪---+
```

Create and configure the routers

1. Create the routers.

```
$ openstack router create router1
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2017-01-10T13:15:19Z                 |
| description             |                                      |
| distributed             | False                                |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| headers                 |                                      |
| id                      | 3f6f4ef8-63be-11e6-bbb3-2fbcef363ab8 |
| name                    | router1                              |
| project_id              | b3ac05ef10bf441fbf4aa17f16ae1e6d     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    | []                                   |
| updated_at              | 2017-01-10T13:15:19Z                 |
+-------------------------+--------------------------------------+

$ openstack router create router2
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2017-01-10T13:15:19Z                 |
| description             |                                      |
| distributed             | False                                |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| headers                 |                                      |
| id                      | 3fd21a60-63be-11e6-9c95-5714c208c499 |
| name                    | router2                              |
| project_id              | b3ac05ef10bf441fbf4aa17f16ae1e6d     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    | []                                   |
| updated_at              | 2017-01-10T13:15:19Z                 |
+-------------------------+--------------------------------------+

$ openstack router create router3
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
```

(continues on next page)

(continued from previous page)

```
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2017-01-10T13:15:19Z                 |
| description             |                                      |
| distributed             | False                                |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| headers                 |                                      |
| id                      | 40069a4c-63be-11e6-9ecc-e37c1eaa7e84 |
| name                    | router3                              |
| project_id              | b3ac05ef10bf441fbf4aa17f16ae1e6d     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    | []                                   |
| updated_at              | 2017-01-10T13:15:19Z                 |
+-------------------------+--------------------------------------+
```

2. For each router, add one self-service subnet as an interface on the router.

```
$ openstack router add subnet router1 selfservice1

$ openstack router add subnet router2 selfservice2

$ openstack router add subnet router3 selfservice3
```

3. Add the provider network as a gateway on each router.

```
$ openstack router set --external-gateway provider router1

$ openstack router set --external-gateway provider router2

$ openstack router set --external-gateway provider router3
```

Create and configure the BGP speaker

The BGP speaker advertises the next-hop IP address for eligible self-service networks and floating IP addresses for instances using those networks.

1. Create the BGP speaker.

```
$ openstack bgp speaker create --ip-version 4 \
  --local-as LOCAL_AS bgpspeaker
Created a new bgp_speaker:
+-----------------------------------+--------------------------------
↪------+
| Field                             | Value                          ␣
↪      |
+-----------------------------------+--------------------------------
↪------+
| advertise_floating_ip_host_routes | True                           ␣
↪      |
| advertise_tenant_networks         | True                           ␣
↪      |
```

(continues on next page)

(continued from previous page)

```
| id                                | 5f227f14-4f46-4eca-9524-
↪fc5a1eabc358 |
| ip_version                        | 4                                ␣
↪       |
| local_as                          | 1234                             ␣
↪       |
| name                              | bgpspeaker                       ␣
↪       |
| networks                          |                                  ␣
↪       |
| peers                             |                                  ␣
↪       |
| tenant_id                         |␣
↪b3ac05ef10bf441fbf4aa17f16ae1e6d     |
+-----------------------------------+---------------------------------
↪-----+
```

Replace LOCAL_AS with an appropriate local autonomous system number. The example configuration uses AS 1234.

2. A BGP speaker requires association with a provider network to determine eligible prefixes. The association builds a list of all virtual routers with gateways on provider and self-service networks in the same address scope so the BGP speaker can advertise self-service network prefixes with the corresponding router as the next-hop IP address. Associate the BGP speaker with the provider network.

```
$ openstack bgp speaker add network bgpspeaker provider
Added network provider to BGP speaker bgpspeaker.
```

3. Verify association of the provider network with the BGP speaker.

```
$ openstack bgp speaker show bgpspeaker
+-----------------------------------+---------------------------------
↪-----+
| Field                             | Value                            ␣
↪       |
+-----------------------------------+---------------------------------
↪-----+
| advertise_floating_ip_host_routes | True                             ␣
↪       |
| advertise_tenant_networks         | True                             ␣
↪       |
| id                                | 5f227f14-4f46-4eca-9524-
↪fc5a1eabc358 |
| ip_version                        | 4                                ␣
↪       |
| local_as                          | 1234                             ␣
↪       |
| name                              | bgpspeaker                       ␣
↪       |
| networks                          | 68ec148c-181f-4656-8334-
↪8f4eb148689d |
| peers                             |                                  ␣
↪       |
| tenant_id                         |␣
↪b3ac05ef10bf441fbf4aa17f16ae1e6d     |
```

(continues on next page)

(continued from previous page)

```
+-----------------------------------+----------------------------------
↪-----+
```

4. Verify the prefixes and next-hop IP addresses that the BGP speaker advertises.

```
$ openstack bgp speaker list advertised routes bgpspeaker
+-----------------+--------------+
| Destination     | Nexthop      |
+-----------------+--------------+
| 192.0.2.0/25    | 203.0.113.11 |
| 192.0.2.128/25  | 203.0.113.12 |
+-----------------+--------------+
```

5. Create a BGP peer.

```
$ openstack bgp peer create --peer-ip 192.0.2.1 \
  --remote-as REMOTE_AS bgppeer
Created a new bgp_peer:
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| auth_type  | none                                 |
| id         | 35c89ca0-ac5a-4298-a815-0b073c2362e9 |
| name       | bgppeer                              |
| peer_ip    | 192.0.2.1                            |
| remote_as  | 4321                                 |
| tenant_id  | b3ac05ef10bf441fbf4aa17f16ae1e6d     |
+------------+--------------------------------------+
```

Replace REMOTE_AS with an appropriate remote autonomous system number. The example configuration uses AS 4321 which triggers EBGP peering.

Note: The host containing the BGP agent must have layer-3 connectivity to the provider router.

6. Add a BGP peer to the BGP speaker.

```
$ openstack bgp speaker add peer bgpspeaker bgppeer
Added BGP peer bgppeer to BGP speaker bgpspeaker.
```

7. Verify addition of the BGP peer to the BGP speaker.

```
$ openstack bgp speaker show bgpspeaker
+-----------------------------------+----------------------------------
↪-----+
| Field                             | Value                            ␣
↪     |
+-----------------------------------+----------------------------------
↪-----+
| advertise_floating_ip_host_routes | True                             ␣
↪     |
| advertise_tenant_networks         | True                             ␣
↪     |
| id                                | 5f227f14-4f46-4eca-9524-
↪fc5a1eabc358 |
```

(continues on next page)

(continued from previous page)

```
| ip_version                        | 4                                    ␣
↪     |
| local_as                          | 1234                                 ␣
↪     |
| name                              | bgpspeaker                           ␣
↪     |
| networks                          | 68ec148c-181f-4656-8334-
↪8f4eb148689d |
| peers                             | 35c89ca0-ac5a-4298-a815-
↪0b073c2362e9 |
| tenant_id                         |␣
↪b3ac05ef10bf441fbf4aa17f16ae1e6d    |
+-----------------------------------+----------------------------------
↪-----+
```

Note: After creating a peering session, you cannot change the local or remote autonomous system numbers.

Schedule the BGP speaker to an agent

1. Unlike most agents, BGP speakers require manual scheduling to an agent. BGP speakers only form peering sessions and begin prefix advertisement after scheduling to an agent. Schedule the BGP speaker to agent `37729181-2224-48d8-89ef-16eca8e2f77e`.

```
$ openstack bgp dragent add speaker 37729181-2224-48d8-89ef-
↪16eca8e2f77e bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.
```

2. Verify scheduling of the BGP speaker to the agent.

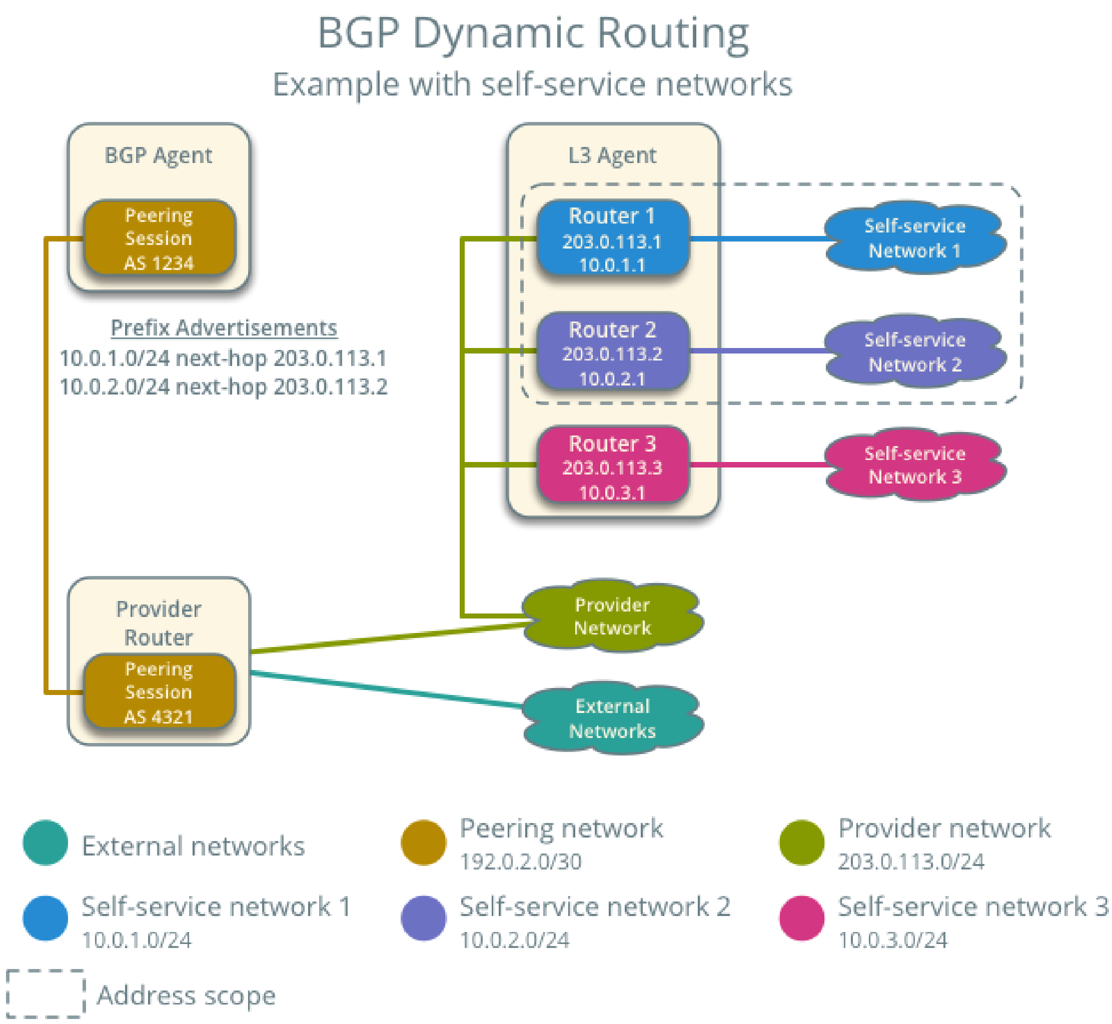
```
$ openstack bgp speaker show dragents bgpspeaker
+--------------------------------------+------------+-------+-------+
| ID                                   | Host       | State | Alive |
+--------------------------------------+------------+-------+-------+
| 37729181-2224-48d8-89ef-16eca8e2f77e | controller | True  | :-)   |
+--------------------------------------+------------+-------+-------+
```

Prefix advertisement

BGP dynamic routing advertises prefixes for self-service networks and host routes for floating IP addresses.

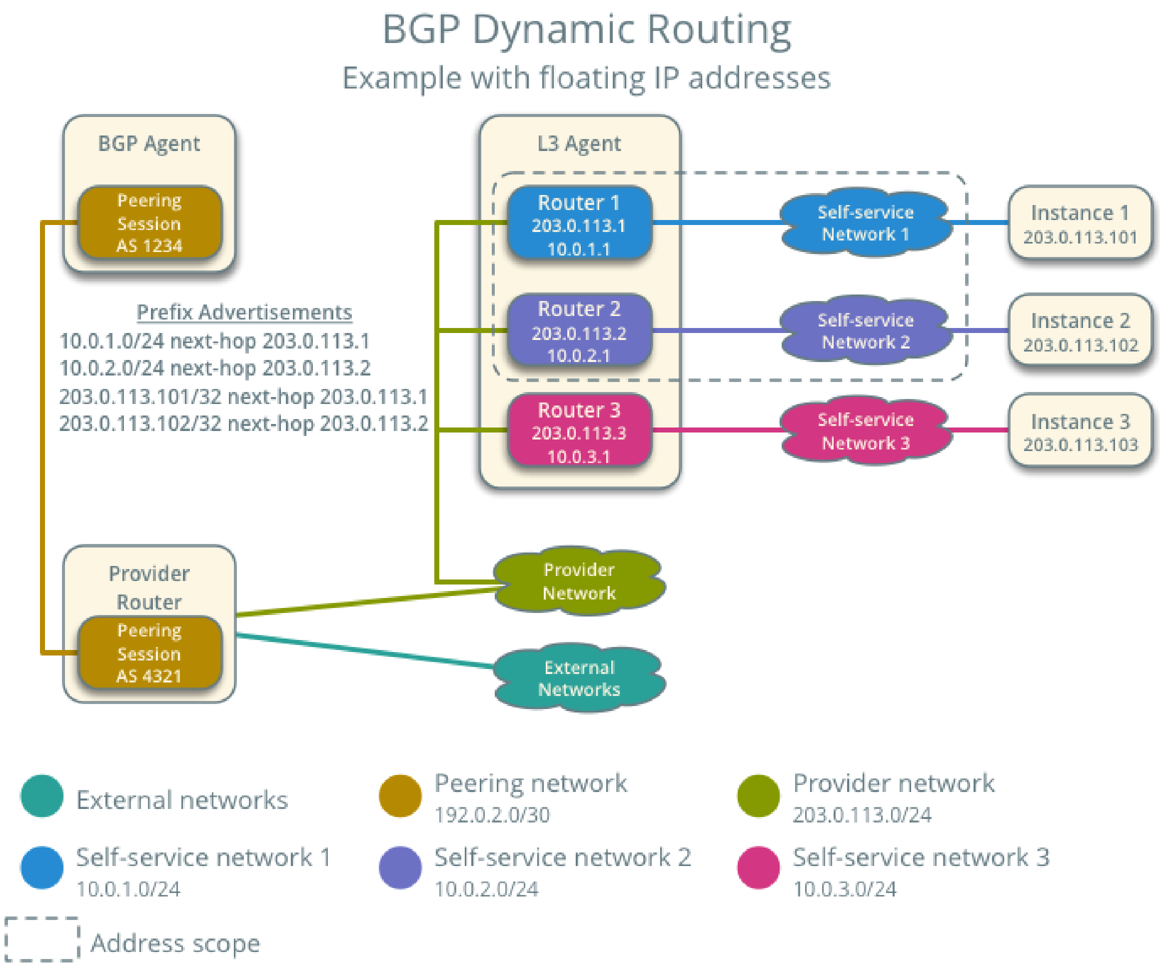
Advertisement of a self-service network requires satisfying the following conditions:

- The external and self-service network reside in the same address scope.
- The router contains an interface on the self-service subnet and a gateway on the external network.
- The BGP speaker associates with the external network that provides a gateway on the router.
- The BGP speaker has the `advertise_tenant_networks` attribute set to `True`.



Advertisement of a floating IP address requires satisfying the following conditions:

- The router with the floating IP address binding contains a gateway on an external network with the BGP speaker association.
- The BGP speaker has the `advertise_floating_ip_host_routes` attribute set to `True`.



Operation with Distributed Virtual Routers (DVR)

For both floating IP and IPv4 fixed IP addresses, the BGP speaker advertises the floating IP agent gateway on the corresponding compute node as the next-hop IP address. When using IPv6 fixed IP addresses, the BGP speaker advertises the DVR SNAT node as the next-hop IP address.

For example, consider the following components:

1. A provider network using IP address range 203.0.113.0/24, and supporting floating IP addresses 203.0.113.101, 203.0.113.102, and 203.0.113.103.
2. A self-service network using IP address range 198.51.100.0/24.
3. Instances with fixed IPs 198.51.100.11, 198.51.100.12, and 198.51.100.13
4. The SNAT gateway resides on 203.0.113.11.
5. The floating IP agent gateways (one per compute node) reside on 203.0.113.12, 203.0.113.13, and 203.0.113.14.
6. Three instances, one per compute node, each with a floating IP address.
7. `advertise_tenant_networks` is set to `False` on the BGP speaker

```
$ openstack bgp speaker list advertised routes bgpspeaker
+------------------+--------------+
| Destination      | Nexthop      |
+------------------+--------------+
| 198.51.100.0/24  | 203.0.113.11 |
| 203.0.113.101/32 | 203.0.113.12 |
| 203.0.113.102/32 | 203.0.113.13 |
| 203.0.113.103/32 | 203.0.113.14 |
+------------------+--------------+
```

When floating IPs are disassociated and `advertise_tenant_networks` is set to `True`, the following routes will be advertised:

```
$ openstack bgp speaker list advertised routes bgpspeaker
+------------------+--------------+
| Destination      | Nexthop      |
+------------------+--------------+
| 198.51.100.0/24  | 203.0.113.11 |
| 198.51.100.11/32 | 203.0.113.12 |
| 198.51.100.12/32 | 203.0.113.13 |
| 198.51.100.13/32 | 203.0.113.14 |
+------------------+--------------+
```

You can also identify floating IP agent gateways in your environment to assist with verifying operation of the BGP speaker.

```
$ openstack port list --device-owner network:floatingip_agent_gateway
+--------------------------------------+------+-------------------+--------
↪-----------------------------------------------------------------------------
↪-----------------------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                                ␣
↪                          |
+--------------------------------------+------+-------------------+--------
↪-----------------------------------------------------------------------------
↪-----------------------+
| 87cf2970-4970-462e-939e-00e808295dfa |      | fa:16:3e:7c:68:e3 | ip_
↪address='203.0.113.12', subnet_id='8ed65d41-2b2a-4f3a-9f92-45adb266e01a'␣
↪                             |
| 8d218440-0d2e-49d0-8a7b-3266a6146dc1 |      | fa:16:3e:9d:78:cf | ip_
↪address='203.0.113.13', subnet_id='8ed65d41-2b2a-4f3a-9f92-45adb266e01a'␣
↪                             |
| 87cf2970-4970-462e-939e-00e802281dfa |      | fa:16:3e:6b:18:e0 | ip_
↪address='203.0.113.14', subnet_id='8ed65d41-2b2a-4f3a-9f92-45adb266e01a'␣
↪                             |
+--------------------------------------+------+-------------------+--------
↪-----------------------------------------------------------------------------
↪-----------------------+
```

IPv6

BGP dynamic routing supports peering via IPv6 and advertising IPv6 prefixes.

- To enable peering via IPv6, create a BGP peer and use an IPv6 address for `peer_ip`.
- To enable advertising IPv6 prefixes, create an address scope with `ip_version=6` and a BGP speaker with `ip_version=6`.

Note: DVR lacks support for routing directly to a fixed IPv6 address via the floating IP agent gateway port and thus prevents the BGP speaker from advertising /128 host routes.

High availability

BGP dynamic routing supports scheduling a BGP speaker to multiple agents which effectively multiplies prefix advertisements to the same peer. If an agent fails, the peer continues to receive advertisements from one or more operational agents.

1. Show available dynamic routing agents.

```
$ openstack network agent list --agent-type bgp
+--------------------------------------+---------------------------+--
↪----- --+-------------------+-------+-------+----------------------
↪----+
| ID                                   | Agent Type                |␣
↪Host      | Availability Zone | Alive | State | Binary              ␣
↪      |
+--------------------------------------+---------------------------+--
↪--------+-------------------+-------+-------+----------------------
↪----+
| 37729181-2224-48d8-89ef-16eca8e2f77e | BGP dynamic routing agent |␣
↪bgp-ha1   | None              | :-)   | UP    | neutron-bgp-dragent ␣
↪      |
| 1a2d33bb-9321-30a2-76ab-22eff3d2f56a | BGP dynamic routing agent |␣
↪bgp-ha2   | None              | :-)   | UP    | neutron-bgp-dragent ␣
↪      |
+--------------------------------------+---------------------------+--
↪--------+-------------------+-------+-------+----------------------
↪----+
```

2. Schedule BGP speaker to multiple agents.

```
$ openstack bgp dragent add speaker 37729181-2224-48d8-89ef-
↪16eca8e2f77e bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.

$ openstack bgp dragent add speaker 1a2d33bb-9321-30a2-76ab-
↪22eff3d2f56a bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.

$ openstack bgp speaker show dragents bgpspeaker
+--------------------------------------+---------+-------+-------+
| ID                                   | Host    | State | Alive |
+--------------------------------------+---------+-------+-------+
```

(continues on next page)

(continued from previous page)

```
| 37729181-2224-48d8-89ef-16eca8e2f77e | bgp-ha1 | True  | :-)   |
| 1a2d33bb-9321-30a2-76ab-22eff3d2f56a | bgp-ha2 | True  | :-)   |
+--------------------------------------+---------+-------+-------+
```

8.2.7 High-availability for DHCP

This section describes how to use the agent management (alias agent) and scheduler (alias agent_scheduler) extensions for DHCP agents scalability and HA.

Note: Use the **openstack extension list** command to check if these extensions are enabled. Check `agent` and `agent_scheduler` are included in the output.

```
$ openstack extension list --network -c Name -c Alias
+-------------------------------------------------------------+------------
↪---------------+
| Name                                                        | Alias      ␣
↪               |
+-------------------------------------------------------------+------------
↪---------------+
| Default Subnetpools                                         | default-
↪subnetpools      |
| Network IP Availability                                     | network-ip-
↪availability   |
| Network Availability Zone                                   | network_
↪availability_zone |
| Auto Allocated Topology Services                            | auto-
↪allocated-topology   |
| Neutron L3 Configurable external gateway mode               | ext-gw-
↪mode             |
| Port Binding                                                | binding    ␣
↪               |
| Neutron Metering                                            | metering   ␣
↪               |
| agent                                                       | agent      ␣
↪               |
| Subnet Allocation                                           | subnet_
↪allocation       |
| L3 Agent Scheduler                                          | l3_agent_
↪scheduler       |
| Neutron external network                                    | external-
↪net             |
| Neutron Service Flavors                                     | flavors    ␣
↪               |
| Network MTU                                                 | net-mtu    ␣
↪               |
| Availability Zone                                           |␣
↪availability_zone        |
| Quota management support                                    | quotas     ␣
↪               |
| HA Router extension                                         | l3-ha      ␣
↪               |
| Provider Network                                            | provider   ␣
↪               |
```

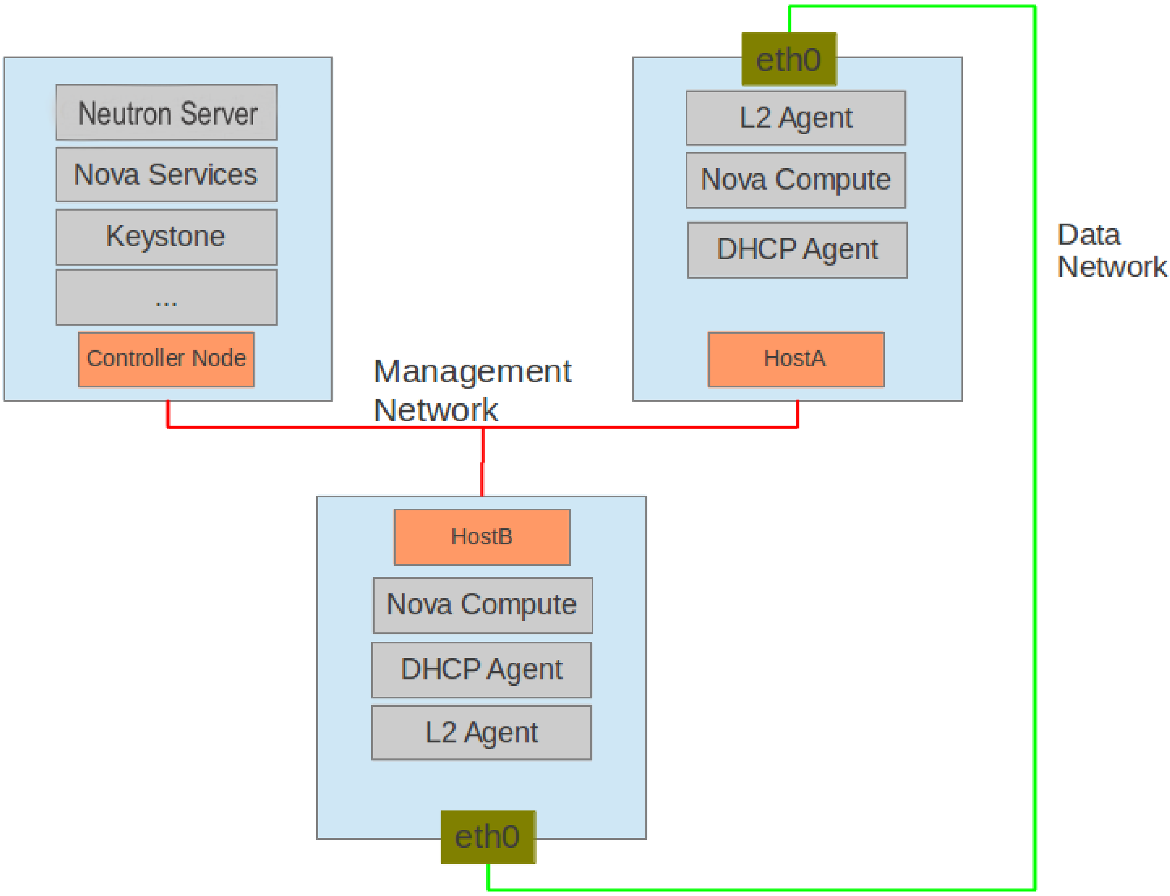
(continues on next page)

(continued from previous page)

```
| Multi Provider Network                                        | multi-
↪provider          |
| Address scope                                                 | address-
↪scope           |
| Neutron Extra Route                                           | extraroute␣
↪                |
| Subnet service types                                          | subnet-
↪service-types     |
| Resource timestamps                                           | standard-
↪attr-timestamp    |
| Neutron Service Type Management                               | service-
↪type            |
| Router Flavor Extension                                       | l3-flavors␣
↪                |
| Neutron Extra DHCP opts                                       | extra_dhcp_
↪opt             |
| Resource revision numbers                                     | standard-
↪attr-revisions    |
| Pagination support                                            | pagination␣
↪                |
| Sorting support                                               | sorting   ␣
↪                |
| security-group                                                | security-
↪group             |
| DHCP Agent Scheduler                                          | dhcp_agent_
↪scheduler       |
| Router Availability Zone                                      | router_
↪availability_zone  |
| RBAC Policies                                                 | rbac-
↪policies            |
| standard-attr-description                                     | standard-
↪attr-description |
| Neutron L3 Router                                             | router    ␣
↪                |
| Allowed Address Pairs                                         | allowed-
↪address-pairs     |
| project_id field enabled                                      | project-id␣
↪                |
| Distributed Virtual Router                                    | dvr       ␣
↪                |
+---------------------------------------------------------------+------------
↪---------------+
```

Demo setup

There will be three hosts in the setup.



Host	Description
OpenStack controller host - controlnode	Runs the Networking, Identity, and Compute services that are required to deploy VMs. The node must have at least one network interface that is connected to the Management Network. Note that `nova-network` should not be running because it is replaced by Neutron.
HostA	Runs `nova-compute`, the Neutron L2 agent and DHCP agent
HostB	Same as HostA

Configuration

controlnode: neutron server

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
core_plugin = linuxbridge
rabbit_host = controlnode
allow_overlapping_ips = True
host = controlnode
agent_down_time = 5
dhcp_agents_per_network = 1
```

Note: In the above configuration, we use `dhcp_agents_per_network = 1` for this demonstration. In usual deployments, we suggest setting `dhcp_agents_per_network` to more than one to match the number of DHCP agents in your deployment. See *Enabling DHCP high availability by default*.

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql+pymysql://root:root@127.0.0.1:3306/neutron_linux_
↪bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

HostA and HostB: L2 agent

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
rabbit_host = controlnode
rabbit_password = openstack
# host = HostB on hostb
host = HostA
```

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

3. Update the nova configuration file `/etc/nova/nova.conf`:

```
[DEFAULT]
use_neutron=True
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
admin_username=neutron
admin_password=servicepassword
admin_auth_url=http://controlnode:35357/v2.0/
auth_strategy=keystone
admin_tenant_name=servicetenant
url=http://203.0.113.10:9696/
```

HostA and HostB: DHCP agent

- Update the DHCP configuration file `/etc/neutron/dhcp_agent.ini`:

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

Prerequisites for demonstration

Admin role is required to use the agent management and scheduler extensions. Ensure you run the following commands under a project with an admin role.

To experiment, you need VMs and a neutron network:

```
$ openstack server list
+--------------------------------------+-----------+--------+--------------
↪--+--------+----------+
| ID                                   | Name      | Status | Networks     ␣
↪  | Image  | Flavor   |
+--------------------------------------+-----------+--------+--------------
↪--+--------+----------+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=192.0.2.
↪3 | cirros | m1.tiny  |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=192.0.2.
↪4 | ubuntu | m1.small |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=192.0.2.
↪5 | centos | m1.small |
+--------------------------------------+-----------+--------+--------------
↪--+--------+----------+

$ openstack network list
+--------------------------------------+------+----------------------------
↪----------+
| ID                                   | Name | Subnets                    ␣
↪          |
+--------------------------------------+------+----------------------------
↪----------+
| ad88e059-e7fa-4cf7-8857-6731a2a3a554 | net1 | 8086db87-3a7a-4cad-88c9-
↪7bab9bc69258 |
+--------------------------------------+------+----------------------------
↪----------+
```

Managing agents in neutron deployment

1. List all agents:

```
$ openstack network agent list
+--------------------------------------+--------------------+-------+-
↪-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host  |␣
↪Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------+-
↪-------------------+-------+-------+---------------------------+
| 22467163-01ea-4231-ba45-3bd316f425e6 | Linux bridge agent | HostA |␣
↪None              | True  | UP    | neutron-linuxbridge-agent |
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | DHCP agent         | HostA |␣
↪None              | True  | UP    | neutron-dhcp-agent        |
```

(continues on next page)

(continued from previous page)

```
| 3066d20c-9f8f-440c-ae7c-a40ffb4256b6 | Linux bridge agent | HostB |␣
↪nova              | True   | UP     | neutron-linuxbridge-agent |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | DHCP agent         | HostB |␣
↪nova              | True   | UP     | neutron-dhcp-agent        |
+--------------------------------------+--------------------+-------+-
↪------------------+-------+-------+---------------------------+
```

Every agent that supports these extensions will register itself with the neutron server when it starts up.

The output shows information for four agents. The `alive` field shows `True` if the agent reported its state within the period defined by the `agent_down_time` option in the `neutron.conf` file. Otherwise the `alive` is `False`.

2. List DHCP agents that host a specified network:

```
$ openstack network agent list --network net1
+--------------------------------------+---------------+---------------
↪--+-------+
| ID                                   | Host          | Admin State␣
↪Up | Alive |
+--------------------------------------+---------------+---------------
↪--+-------+
| 22467163-01ea-4231-ba45-3bd316f425e6 | HostA         | UP            ␣
↪   | True  |
+--------------------------------------+---------------+---------------
↪--+-------+
```

3. List the networks hosted by a given DHCP agent:

   This command is to show which networks a given dhcp agent is managing.

```
$ openstack network list --agent 22467163-01ea-4231-ba45-3bd316f425e6
+--------------------------------+------------------------+-----------
↪----------------------+
| ID                             | Name                   | Subnets   ␣
↪                     |
+--------------------------------+------------------------+-----------
↪----------------------+
| ad88e059-e7fa-                 | net1                   | 8086db87-
↪3a7a-4cad-                |
| 4cf7-8857-6731a2a3a554         |                        | 88c9-
↪7bab9bc69258                  |
+--------------------------------+------------------------+-----------
↪----------------------+
```

4. Show agent details.

   The **openstack network agent show** command shows details for a specified agent:

```
$ openstack network agent show 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b
+---------------------+---------------------------------------------------
↪---+
| Field               | Value                                            ␣
↪   |
+---------------------+---------------------------------------------------
↪---+
```

(continues on next page)

(continued from previous page)

```
| admin_state_up        | UP                                               ␣
↪     |
| agent_type            | DHCP agent                                       ␣
↪     |
| alive                 | True                                             ␣
↪     |
| availability_zone     | nova                                             ␣
↪     |
| binary                | neutron-dhcp-agent                               ␣
↪     |
| configurations        | dhcp_driver='neutron.agent.linux.dhcp.Dnsmasq
↪',   |
|                       | dhcp_lease_duration='86400',                     ␣
↪     |
|                       | log_agent_heartbeats='False', networks='1',      ␣
↪     |
|                       | notifies_port_ready='True', ports='3',           ␣
↪     |
|                       | subnets='1'                                      ␣
↪     |
| created_at            | 2016-12-14 00:25:54                              ␣
↪     |
| description           | None                                             ␣
↪     |
| last_heartbeat_at     | 2016-12-14 06:53:24                              ␣
↪     |
| host                  | HostA                                            ␣
↪     |
| id                    | 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b             ␣
↪     |
| started_at            | 2016-12-14 00:25:54                              ␣
↪     |
| topic                 | dhcp_agent                                       ␣
↪     |
+-----------------------+-------------------------------------------------
↪---+
```

In this output, `last_heartbeat_at` is the time on the neutron server. You do not need to synchronize all agents to this time for this extension to run correctly. `configurations` describes the static configuration for the agent or run time data. This agent is a DHCP agent and it hosts one network, one subnet, and three ports.

Different types of agents show different details. The following output shows information for a Linux bridge agent:

```
$ openstack network agent show 22467163-01ea-4231-ba45-3bd316f425e6
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | UP                                   |
| agent_type            | Linux bridge agent                   |
| alive                 | True                                 |
| availability_zone     | nova                                 |
| binary                | neutron-linuxbridge-agent            |
| configurations        | {                                    |
```

(continues on next page)

(continued from previous page)

```
|                       |       "physnet1": "eth0",                 |
|                       |       "devices": "4"                      |
|                       | }                                         |
| created_at            | 2016-12-14 00:26:54                       |
| description           | None                                      |
| last_heartbeat_at     | 2016-12-14 06:53:24                       |
| host                  | HostA                                     |
| id                    | 22467163-01ea-4231-ba45-3bd316f425e6 |
| started_at            | 2016-12-14T06:48:39.000000                |
| topic                 | N/A                                       |
+-----------------------+-------------------------------------------+
```

The output shows `bridge-mapping` and the number of virtual network devices on this L2 agent.

Managing assignment of networks to DHCP agent

A single network can be assigned to more than one DHCP agents and one DHCP agent can host more than one network. You can add a network to a DHCP agent and remove one from it.

1. Default scheduling.

   When you create a network with one port, the network will be scheduled to an active DHCP agent. If many active DHCP agents are running, select one randomly. You can design more sophisticated scheduling algorithms in the same way as nova-schedule later on.

```
$ openstack network create net2
$ openstack subnet create --network net2 --subnet-range 198.51.100.0/
↪24 subnet2
$ openstack port create port2 --network net2
$ openstack network agent list --network net2
+--------------------------------------+----------------+--------------
↪--+-------+
| ID                                   | Host           | Admin State␣
↪Up | Alive |
+--------------------------------------+----------------+--------------
↪--+-------+
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | HostA          | UP           ␣
↪  | True  |
+--------------------------------------+----------------+--------------
↪--+-------+
```

   It is allocated to DHCP agent on HostA. If you want to validate the behavior through the **dnsmasq** command, you must create a subnet for the network because the DHCP agent starts the dnsmasq service only if there is a DHCP.

2. Assign a network to a given DHCP agent.

   To add another DHCP agent to host the network, run this command:

```
$ openstack network agent add network --dhcp \
  55569f4e-6f31-41a6-be9d-526efce1f7fe net2
$ openstack network agent list --network net2
+--------------------------------------+--------+-----------------+-----
↪---+
```

(continues on next page)

(continued from previous page)

```
| ID                                   | Host  | Admin State Up |␣
↪Alive  |
+--------------------------------------+-------+----------------+-----
↪---+
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | HostA | UP             |␣
↪True   |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | HostB | UP             |␣
↪True   |
+--------------------------------------+-------+----------------+-----
↪---+
```

Both DHCP agents host the `net2` network.

3. Remove a network from a specified DHCP agent.

   This command is the sibling command for the previous one. Remove `net2` from the DHCP agent for HostA:

```
$ openstack network agent remove network --dhcp \
  2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b net2
$ openstack network agent list --network net2
+--------------------------------------+-------+----------------+-----
↪--+
| ID                                   | Host  | Admin State Up |␣
↪Alive |
+--------------------------------------+-------+----------------+-----
↪--+
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | HostB | UP             |␣
↪True  |
+--------------------------------------+-------+----------------+-----
↪--+
```

   You can see that only the DHCP agent for HostB is hosting the `net2` network.

HA of DHCP agents

Boot a VM on `net2`. Let both DHCP agents host `net2`. Fail the agents in turn to see if the VM can still get the desired IP.

1. Boot a VM on `net2`:

```
$ openstack network list
+--------------------------------------+------+-----------------------
↪---------------+
| ID                                   | Name | Subnets              ␣
↪               |
+--------------------------------------+------+-----------------------
↪---------------+
| ad88e059-e7fa-4cf7-8857-6731a2a3a554 | net1 | 8086db87-3a7a-4cad-
↪88c9-7bab9bc69258 |
| 9b96b14f-71b8-4918-90aa-c5d705606b1a | net2 | 6979b71a-0ae8-448c-
↪aa87-65f68eedcaaa |
+--------------------------------------+------+-----------------------
↪---------------+
$ openstack server create --image tty  --flavor 1 myserver4 \
```

(continues on next page)

(continued from previous page)

```
  --nic net-id=9b96b14f-71b8-4918-90aa-c5d705606b1a
...
$ openstack server list
+--------------------------------------+-----------+--------+---------
↪----------+---------+----------+
| ID                                   | Name      | Status |␣
↪Networks          | Image   | Flavor   |
+--------------------------------------+-----------+--------+---------
↪----------+---------+----------+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE |␣
↪net1=192.0.2.3    | cirros  | m1.tiny  |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE |␣
↪net1=192.0.2.4    | ubuntu  | m1.small |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE |␣
↪net1=192.0.2.5    | centos  | m1.small |
| f62f4731-5591-46b1-9d74-f0c901de567f | myserver4 | ACTIVE |␣
↪net2=198.51.100.2 | cirros1 | m1.tiny  |
+--------------------------------------+-----------+--------+---------
↪----------+---------+----------+
```

2. Make sure both DHCP agents hosting `net2`:

   Use the previous commands to assign the network to agents.

```
$ openstack network agent list --network net2
+--------------------------------------+-------+----------------+-----
↪--+
| ID                                   | Host  | Admin State Up |␣
↪Alive |
+--------------------------------------+-------+----------------+-----
↪--+
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | HostA | UP             |␣
↪True  |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | HostB | UP             |␣
↪True  |
+--------------------------------------+-------+----------------+-----
↪--+
```

To test the HA of DHCP agent:

1. Log in to the `myserver4` VM, and run `udhcpc`, `dhclient` or other DHCP client.
2. Stop the DHCP agent on HostA. Besides stopping the `neutron-dhcp-agent` binary, you must stop the `dnsmasq` processes.
3. Run a DHCP client in VM to see if it can get the wanted IP.
4. Stop the DHCP agent on HostB too.
5. Run `udhcpc` in the VM; it cannot get the wanted IP.
6. Start DHCP agent on HostB. The VM gets the wanted IP again.

No HA for metadata service on isolated networks

All Neutron backends using the DHCP agent can also provide metadata service in isolated networks (i.e. networks without a router). In this case the DHCP agent manages the metadata service (see config option enable_isolated_metadata).

Note however that the metadata service is only redundant for IPv4, and not IPv6, even when the DHCP service is configured to be highly available (config option dhcp_agents_per_network > 1). This is because the DHCP agent will insert a route to the well known metadata IPv4 address (*169.254.169.254*) via its own IP address, so it will be reachable as long as the DHCP service is available at that IP address. This also means that recovery after a failure is tied to the renewal of the DHCP lease, since that route will only change if the DHCP server for a VM changes.

With IPv6, the well known metadata IPv6 address (*fe80::a9fe:a9fe*) is used, but directly configured in the DHCP agent network namespace. Due to the enforcement of duplicate address detection (DAD), this address can only be configured in at most one DHCP network namespaces at any time. See RFC 4862 for details on the DAD process.

For this reason, even when you have multiple DHCP agents, an arbitrary one (where the metadata IPv6 address is not in *dadfailed* state) will serve all metadata requests over IPv6. When that metadata service instance becomes unreachable there is no failover and the service will become unreachable.

Disabling and removing an agent

An administrator might want to disable an agent if a system hardware or software upgrade is planned. Some agents that support scheduling also support disabling and enabling agents, such as L3 and DHCP agents. After the agent is disabled, the scheduler does not schedule new resources to the agent.

After the agent is disabled, you can safely remove the agent. Even after disabling the agent, resources on the agent are kept assigned. Ensure you remove the resources on the agent before you delete the agent.

Disable the DHCP agent on HostA before you stop it:

```
$ openstack network agent set 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b --
↪disable
$ openstack network agent list
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host  |␣
↪Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
| 22467163-01ea-4231-ba45-3bd316f425e6 | Linux bridge agent | HostA | None␣
↪                  | True  | UP    | neutron-linuxbridge-agent |
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | DHCP agent         | HostA | None␣
↪                  | True  | DOWN  | neutron-dhcp-agent        |
| 3066d20c-9f8f-440c-ae7c-a40ffb4256b6 | Linux bridge agent | HostB | nova␣
↪                  | True  | UP    | neutron-linuxbridge-agent |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | DHCP agent         | HostB | nova␣
↪                  | True  | UP    | neutron-dhcp-agent        |
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
```

After you stop the DHCP agent on HostA, you can delete it by the following command:

```
$ openstack network agent delete 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b
$ openstack network agent list
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host  |␣
↪Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
| 22467163-01ea-4231-ba45-3bd316f425e6 | Linux bridge agent | HostA | None␣
↪                | True  | UP    | neutron-linuxbridge-agent |
| 3066d20c-9f8f-440c-ae7c-a40ffb4256b6 | Linux bridge agent | HostB | nova␣
↪                | True  | UP    | neutron-linuxbridge-agent |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | DHCP agent         | HostB | nova␣
↪                | True  | UP    | neutron-dhcp-agent        |
+--------------------------------------+--------------------+-------+------
↪-------------+-------+-------+---------------------------+
```

After deletion, if you restart the DHCP agent, it appears on the agent list again.

Enabling DHCP high availability by default

You can control the default number of DHCP agents assigned to a network by setting the following configuration option in the file `/etc/neutron/neutron.conf`.

```
dhcp_agents_per_network = 3
```

8.2.8 DNS integration

This page serves as a guide for how to use the DNS integration functionality of the Networking service and its interaction with the Compute service.

The integration of the Networking service with an external DNSaaS (DNS-as-a-Service) is described in *DNS integration with an external service*.

Users can control the behavior of the Networking service in regards to DNS using two attributes associated with ports, networks, and floating IPs. The following table shows the attributes available for each one of these resources:

Resource	dns_name	dns_domain
Ports	Yes	Yes
Networks	No	Yes
Floating IPs	Yes	Yes

Note: The `DNS Integration` extension enables all the attribute and resource combinations shown in the previous table, except for `dns_domain` for ports, which requires the `dns_domain for ports` extension.

Note: Since the `DNS Integration` extension is a subset of `dns_domain for ports`, if dns_domain functionality for ports is required, only the latter extension has to be configured.

Note: When the dns_domain for ports extension is configured, DNS Integration is also included when the Neutron server responds to a request to list the active API extensions. This preserves backwards API compatibility.

The Networking service internal DNS resolution

The Networking service enables users to control the name assigned to ports by the internal DNS. To enable this functionality, do the following:

1. Edit the /etc/neutron/neutron.conf file and assign a value different to openstacklocal (its default value) to the dns_domain parameter in the [default] section. As an example:

```
dns_domain = example.org.
```

2. Add dns (for the DNS Integration extension) or dns_domain_ports (for the dns_domain for ports extension) to extension_drivers in the [ml2] section of /etc/neutron/plugins/ml2/ml2_conf.ini. The following is an example:

```
[ml2]
extension_drivers = port_security,dns_domain_ports
```

After re-starting the neutron-server, users will be able to assign a dns_name attribute to their ports.

Note: The enablement of this functionality is prerequisite for the enablement of the Networking service integration with an external DNS service, which is described in detail in *DNS integration with an external service*.

The following illustrates the creation of a port with my-port in its dns_name attribute.

Note: The name assigned to the port by the Networking service internal DNS is now visible in the response in the dns_assignment attribute.

```
$ openstack port create --network my-net --dns-name my-port test
+-----------------------+-----------------------------------------------
↪-------------------------------+
| Field                 | Value                                          ␣
↪                              |
+-----------------------+-----------------------------------------------
↪-------------------------------+
| admin_state_up        | UP                                             ␣
↪                              |
| allowed_address_pairs |                                                ␣
↪                              |
| binding_host_id       |                                                ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| binding_profile       |
↪                           |
| binding_vif_details   |
↪                           |
| binding_vif_type      | unbound
↪                           |
| binding_vnic_type     | normal
↪                           |
| created_at            | 2016-02-05T21:35:04Z
↪                           |
| data_plane_status     | None
↪                           |
| description           |
↪                           |
| device_id             |
↪                           |
| device_owner          |
↪                           |
| dns_assignment        | fqdn='my-port.example.org.', hostname='my-port',
↪ip_address='192.0.2.67'    |
| dns_domain            | None
↪                           |
| dns_name              | my-port
↪                           |
| extra_dhcp_opts       |
↪                           |
| fixed_ips             | ip_address='192.0.2.67', subnet_id='6141b474-
↪56cd-430f-b731-71660bb79b79'     |
| id                    | fb3c10f4-017e-420c-9be1-8f8c557ae21f
↪                           |
| mac_address           | fa:16:3e:aa:9b:e1
↪                           |
| name                  | test
↪                           |
| network_id            | bf2802a0-99a0-4e8c-91e4-107d03f158ea
↪                           |
| port_security_enabled | True
↪                           |
| project_id            | d5660cb1e6934612a01b4fb2fb630725
↪                           |
| qos_policy_id         | None
↪                           |
| revision_number       | 1
↪                           |
| security_group_ids    | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635
↪                           |
| status                | DOWN
↪                           |
| tags                  |
↪                           |
| trunk_details         | None
↪                           |
| updated_at            | 2016-02-05T21:35:04Z
↪                           |
+-----------------------+-----------------------------------------------
↪-----------------------------+
```

When this functionality is enabled, it is leveraged by the Compute service when creating instances. When allocating ports for an instance during boot, the Compute service populates the `dns_name` attributes of these ports with the `hostname` attribute of the instance, which is a DNS sanitized version of its display name. As a consequence, at the end of the boot process, the allocated ports will be known in the dnsmasq associated to their networks by their instance `hostname`.

The following is an example of an instance creation, showing how its `hostname` populates the `dns_name` attribute of the allocated port:

```
$ openstack server create --image cirros --flavor 42 \
  --nic net-id=37aaff3a-6047-45ac-bf4f-a825e56fd2b3 my_vm
+--------------------------------------+----------------------------------
↪------------------------------+
| Field                                | Value                            ␣
↪                              |
+--------------------------------------+----------------------------------
↪------------------------------+
| OS-DCF:diskConfig                    | MANUAL                           ␣
↪                              |
| OS-EXT-AZ:availability_zone          |                                  ␣
↪                              |
| 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                            | dB45Zvo8Jpfe                     ␣
↪                              |
| config_drive                         |                                  ␣
↪                              |
| created                              | 2016-02-05T21:35:04Z             ␣
↪                              |
| flavor                               | m1.nano (42)                     ␣
↪                              |
| hostId                               |                                  ␣
↪                              |
| id                                   | 66c13cb4-3002-4ab3-8400-
↪7efc2659c363                          |
| image                                | cirros-0.3.5-x86_64-uec(b9d981eb-
↪d21c-4ce2-9dbc-dd38f3d9015f)  |
| key_name                             | -                                ␣
↪                              |
| locked                               | False                            ␣
↪                              |
| metadata                             | {}                               ␣
↪                              |
| name                                 | my_vm                            ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| os-extended-volumes:volumes_attached | []                                   ␣
↪                                |
| progress                             | 0                                    ␣
↪                                |
| security_groups                      | default                              ␣
↪                                |
| status                               | BUILD                                ␣
↪                                |
| tenant_id                            | d5660cb1e6934612a01b4fb2fb630725 ␣
↪                                |
| updated                              | 2016-02-05T21:35:04Z                 ␣
↪                                |
| user_id                              | 8bb6e578cba24e7db9d3810633124525 ␣
↪                                |
+--------------------------------------+-----------------------------------
↪------------------------------+

$ openstack port list --device-id 66c13cb4-3002-4ab3-8400-7efc2659c363
+--------------------------------------+------+-------------------+---------
↪------------------------------------------------------------------------
↪------+--------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                            ␣
↪        | Status |
+--------------------------------------+------+-------------------+---------
↪------------------------------------------------------------------------
↪------+--------+
| b3ecc464-1263-44a7-8c38-2d8a52751773 |      | fa:16:3e:a8:ce:b8 | ip_
↪address='203.0.113.8', subnet_id='277eca5d-9869-474b-960e-6da5951d09f7' ␣
↪           | ACTIVE |
|                                      |      |                   | ip_
↪address='2001:db8:10::8', subnet_id='eab47748-3f0a-4775-a09f-b0c24bb64bc4
↪'          |        |
+--------------------------------------+------+-------------------+---------
↪------------------------------------------------------------------------
↪------+--------+

$ openstack port show b3ecc464-1263-44a7-8c38-2d8a52751773
+-----------------------+---------------------------------------------------
↪------------------------------------------------------------+
| Field                 | Value                                              ␣
↪                                                            |
+-----------------------+---------------------------------------------------
↪------------------------------------------------------------+
| admin_state_up        | UP                                                 ␣
↪                                                            |
| allowed_address_pairs |                                                    ␣
↪                                                            |
| binding_host_id       | vultr.guest                                        ␣
↪                                                            |
| binding_profile       |                                                    ␣
↪                                                            |
| binding_vif_details   | datapath_type='system', ovs_hybrid_plug='True',␣
↪port_filter='True'                                                          |
| binding_vif_type      | ovs                                                ␣
↪                                                            |
```

(continues on next page)

(continued from previous page)

```
| binding_vnic_type     | normal                                                 
↪                                                                  |
| created_at            | 2016-02-05T21:35:04Z                                   
↪                                                                  |
| data_plane_status     | None                                                   
↪                                                                  |
| description           |                                                        
↪                                                                  |
| device_id             | 66c13cb4-3002-4ab3-8400-7efc2659c363                   
↪                                                                  |
| device_owner          | compute:None                                           
↪                                                                  |
| dns_assignment        | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='203.0.113.8'                                               |
|                       | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='2001:db8:10::8'                                            |
| dns_domain            | example.org.                                           
↪                                                                  |
| dns_name              | my-vm                                                  
↪                                                                  |
| extra_dhcp_opts       |                                                        
↪                                                                  |
| fixed_ips             | ip_address='203.0.113.8', subnet_id='277eca5d-
↪9869-474b-960e-6da5951d09f7'                                          |
|                       | ip_address='2001:db8:10::8', subnet_id='eab47748-
↪3f0a-4775-a09f-b0c24bb64bc4'                                        |
| id                    | b3ecc464-1263-44a7-8c38-2d8a52751773                   
↪                                                                  |
| mac_address           | fa:16:3e:a8:ce:b8                                      
↪                                                                  |
| name                  |                                                        
↪                                                                  |
| network_id            | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3                   
↪                                                                  |
| port_security_enabled | True                                                   
↪                                                                  |
| project_id            | d5660cb1e6934612a01b4fb2fb630725                       
↪                                                                  |
| qos_policy_id         | None                                                   
↪                                                                  |
| revision_number       | 1                                                      
↪                                                                  |
| security_group_ids    | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635                   
↪                                                                  |
| status                | ACTIVE                                                 
↪                                                                  |
| tags                  |                                                        
↪                                                                  |
| trunk_details         | None                                                   
↪                                                                  |
| updated_at            | 2016-02-05T21:35:04Z                                   
↪                                                                  |
+-----------------------+--------------------------------------------------------
↪------------------------------------------------------------------+
```

In the above example notice that:

- The name given to the instance by the user, `my_vm`, is sanitized by the Compute service and becomes `my-vm` as the ports `dns_name`.
- The ports `dns_assignment` attribute shows that its FQDN is `my-vm.example.org.` in the Networking service internal DNS, which is the result of concatenating the ports `dns_name` with the value configured in the `dns_domain` parameter in `neutron.conf`, as explained previously.
- The `dns_assignment` attribute also shows that the ports `hostname` in the Networking service internal DNS is `my-vm`.
- Instead of having the Compute service create the port for the instance, the user might have created it and assigned a value to its `dns_name` attribute. In this case, the value assigned to the `dns_name` attribute must be equal to the value that Compute service will assign to the instances `hostname`, in this example `my-vm`. Otherwise, the instance boot will fail.

8.2.9 DNS integration with an external service

This page serves as a guide for how to use the DNS integration functionality of the Networking service with an external DNSaaS (DNS-as-a-Service).

As a prerequisite this needs the internal DNS functionality offered by the Networking service to be enabled, see *DNS integration*.

Configuring OpenStack Networking for integration with an external DNS service

The first step to configure the integration with an external DNS service is to enable the functionality described in *The Networking service internal DNS resolution*. Once this is done, the user has to take the following steps and restart `neutron-server`.

1. Edit the `[default]` section of `/etc/neutron/neutron.conf` and specify the external DNS service driver to be used in parameter `external_dns_driver`. The valid options are defined in namespace `neutron.services.external_dns_drivers`. The following example shows how to set up the driver for the OpenStack DNS service:

   ```
   external_dns_driver = designate
   ```

2. If the OpenStack DNS service is the target external DNS, the `[designate]` section of `/etc/neutron/neutron.conf` must define the following parameters:

   - `url`: the OpenStack DNS service public endpoint URL. Note that this must always be the versioned endpoint currently.
   - `auth_type`: the authorization plugin to use. Usually this should be `password`, see https://docs.openstack.org/keystoneauth/latest/authentication-plugins.html for other options.
   - `auth_url`: the Identity service authorization endpoint url. This endpoint will be used by the Networking service to authenticate as an user to create and update reverse lookup (PTR) zones.
   - `username`: the username to be used by the Networking service to create and update reverse lookup (PTR) zones.
   - `password`: the password of the user to be used by the Networking service to create and update reverse lookup (PTR) zones.

- `project_name`: the name of the project to be used by the Networking service to create and update reverse lookup (PTR) zones.
- `project_domain_name`: the name of the domain for the project to be used by the Networking service to create and update reverse lookup (PTR) zones.
- `user_domain_name`: the name of the domain for the user to be used by the Networking service to create and update reverse lookup (PTR) zones.
- `region_name`: the name of the region to be used by the Networking service to create and update reverse lookup (PTR) zones.
- `allow_reverse_dns_lookup`: a boolean value specifying whether to enable or not the creation of reverse lookup (PTR) records.
- `ipv4_ptr_zone_prefix_size`: the size in bits of the prefix for the IPv4 reverse lookup (PTR) zones.
- `ipv6_ptr_zone_prefix_size`: the size in bits of the prefix for the IPv6 reverse lookup (PTR) zones.
- `ptr_zone_email`: the email address to use when creating new reverse lookup (PTR) zones. The default is `admin@<dns_domain>` where `<dns_domain>` is the domain for the first record being created in that zone.
- `insecure`: whether to disable SSL certificate validation. By default, certificates are validated.
- `cafile`: Path to a valid Certificate Authority (CA) certificate. Optional, the system CAs are used as default.

The following is an example:

```
[designate]
url = http://192.0.2.240:9001/v2
auth_type = password
auth_url = http://192.0.2.240:5000
username = neutron
password = PASSWORD
project_name = service
project_domain_name = Default
user_domain_name = Default
allow_reverse_dns_lookup = True
ipv4_ptr_zone_prefix_size = 24
ipv6_ptr_zone_prefix_size = 116
ptr_zone_email = admin@example.org
cafile = /etc/ssl/certs/my_ca_cert
```

Once the `neutron-server` has been configured and restarted, users will have functionality that covers three use cases, described in the following sections. In each of the use cases described below:

- The examples assume the OpenStack DNS service as the external DNS.
- A, AAAA and PTR records will be created in the DNS service.
- Before executing any of the use cases, the user must create in the DNS service under their project a DNS zone where the A and AAAA records will be created. For the description of the use cases below, it is assumed the zone `example.org.` was created previously.

- The PTR records will be created in zones owned by the project specified for `project_name` above.

Use case 1: Floating IPs are published with associated port DNS attributes

In this use case, the address of a floating IP is published in the external DNS service in conjunction with the `dns_name` of its associated port and the `dns_domain` of the ports network. The steps to execute in this use case are the following:

1. Assign a valid domain name to the networks `dns_domain` attribute. This name must end with a period (`.`).
2. Boot an instance or alternatively, create a port specifying a valid value to its `dns_name` attribute. If the port is going to be used for an instance boot, the value assigned to `dns_name` must be equal to the `hostname` that the Compute service will assign to the instance. Otherwise, the boot will fail.
3. Create a floating IP and associate it to the port.

Following is an example of these steps:

```
$ openstack network set --dns-domain example.org. 38c5e950-b450-4c30-83d4-
↪ee181c28aad3

$ openstack network show 38c5e950-b450-4c30-83d4-ee181c28aad3
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        | nova                                 |
| created_at                | 2016-05-04T19:27:34Z                 |
| description               |                                      |
| dns_domain                | example.org.                         |
| id                        | 38c5e950-b450-4c30-83d4-ee181c28aad3 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| is_vlan_transparent       | None                                 |
| mtu                       | 1450                                 |
| name                      | private                              |
| port_security_enabled     | True                                 |
| project_id                | d5660cb1e6934612a01b4fb2fb630725     |
| provider:network_type     | vlan                                 |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 24                                   |
| qos_policy_id             | None                                 |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   | 43414c53-62ae-49bc-aa6c-c9dd7705818a |
|                           | 5b9282a1-0be1-4ade-b478-7868ad2a16ff |
| tags                      |                                      |
| updated_at                | 2016-05-04T19:27:34Z                 |
```

(continues on next page)

(continued from previous page)

```
+-----------------------------+---------------------------------------+

$ openstack server create --image cirros --flavor 42 \
  --nic net-id=38c5e950-b450-4c30-83d4-ee181c28aad3 my_vm
+--------------------------------------+-----------------------------------
↪-----------------------------+
| Field                                | Value                             ␣
↪                             |
+--------------------------------------+-----------------------------------
↪-----------------------------+
| OS-DCF:diskConfig                    | MANUAL                            ␣
↪                             |
| OS-EXT-AZ:availability_zone          |                                   ␣
↪                             |
| 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                            | oTLQLR3Kezmt                      ␣
↪                             |
| config_drive                         |                                   ␣
↪                             |
| created                              | 2016-02-15T19:27:34Z              ␣
↪                             |
| flavor                               | m1.nano (42)                      ␣
↪                             |
| hostId                               |                                   ␣
↪                             |
| id                                   | 43f328bb-b2d1-4cf1-a36f-
↪3b2593397cb1                   |
| image                                | cirros-0.3.5-x86_64-uec (b9d981eb-
↪d21c-4ce2-9dbc-dd38f3d9015f) |
| key_name                             | -                                 ␣
↪                             |
| locked                               | False                             ␣
↪                             |
| metadata                             | {}                                ␣
↪                             |
| name                                 | my_vm                             ␣
↪                             |
| os-extended-volumes:volumes_attached | []                                ␣
↪                             |
| progress                             | 0                                 ␣
↪                             |
| security_groups                      | default                           ␣
↪                             |
```

(continues on next page)

(continued from previous page)

```
| status                         | BUILD                                  ␣
↪                               |
| tenant_id                      | d5660cb1e6934612a01b4fb2fb630725 ␣
↪                               |
| updated                        | 2016-02-15T19:27:34Z                   ␣
↪                               |
| user_id                        | 8bb6e578cba24e7db9d3810633124525 ␣
↪                               |
+--------------------------------+--------------------------------------
↪-----------------------------+

$ openstack server list
+--------------------------------------+-------+--------+-----------------
↪----------------------------------------+---------+----------+
| ID                                   | Name  | Status | Networks        ␣
↪                                           | Image   | Flavor   |
+--------------------------------------+-------+--------+-----------------
↪----------------------------------------+---------+----------+
| 43f328bb-b2d1-4cf1-a36f-3b2593397cb1 | my_vm | ACTIVE |␣
↪private=fda4:653e:71b0:0:f816:3eff:fe16:b5f2, 192.0.2.15 | cirros | m1.
↪nano |
+--------------------------------------+-------+--------+-----------------
↪----------------------------------------+---------+----------+

$ openstack port list --device-id 43f328bb-b2d1-4cf1-a36f-3b2593397cb1
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪----------------------------+---------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                            ␣
↪                               | Status |
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪----------------------------+---------+
| da0b1f75-c895-460f-9fc1-4d6ec84cf85f |      | fa:16:3e:16:b5:f2 | ip_
↪address='192.0.2.15', subnet_id='5b9282a1-0be1-4ade-b478-7868ad2a16ff'  ␣
↪                                  | ACTIVE |
|                                      |      |                   | ip_
↪address='fda4:653e:71b0:0:f816:3eff:fe16:b5f2', subnet_id='43414c53-62ae-
↪49bc-aa6c-c9dd7705818a'          |        |
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪----------------------------+---------+

$ openstack port show da0b1f75-c895-460f-9fc1-4d6ec84cf85f
+-----------------------+-------------------------------------------------
↪--------------------------------------------------------+
| Field                 | Value                                           ␣
↪                                                        |
+-----------------------+-------------------------------------------------
↪--------------------------------------------------------+
| admin_state_up        | UP                                              ␣
↪                                                        |
| allowed_address_pairs |                                                 ␣
↪                                                        |
| binding_host_id       | vultr.guest                                     ␣
↪
```

(continues on next page)

(continued from previous page)

```
| binding_profile       |                                                          ␣
↪                                                                               |
| binding_vif_details   | datapath_type='system', ovs_hybrid_plug='True',␣
↪port_filter='True'                                                              |
| binding_vif_type      | ovs                                                      ␣
↪                                                                               |
| binding_vnic_type     | normal                                                   ␣
↪                                                                               |
| created_at            | 2016-02-15T19:27:34Z                                     ␣
↪                                                                               |
| data_plane_status     | None                                                     ␣
↪                                                                               |
| description           |                                                          ␣
↪                                                                               |
| device_id             | 43f328bb-b2d1-4cf1-a36f-3b2593397cb1                     ␣
↪                                                                               |
| device_owner          | compute:None                                             ␣
↪                                                                               |
| dns_assignment        | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='192.0.2.15'                                                            |
|                       | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='fda4:653e:71b0:0:f816:3eff:fe16:b5f2'                                  |
| dns_domain            | example.org.                                             ␣
↪                                                                               |
| dns_name              | my-vm                                                    ␣
↪                                                                               |
| extra_dhcp_opts       |                                                          ␣
↪                                                                               |
| fixed_ips             | ip_address='192.0.2.15', subnet_id='5b9282a1-
↪0be1-4ade-b478-7868ad2a16ff'                                                        |
|                       | ip_address='fda4:653e:71b0:0:f816:3eff:fe16:b5f2
↪', subnet_id='43414c53-62ae-49bc-aa6c-c9dd7705818a'                                 |
| id                    | da0b1f75-c895-460f-9fc1-4d6ec84cf85f                     ␣
↪                                                                               |
| mac_address           | fa:16:3e:16:b5:f2                                        ␣
↪                                                                               |
| name                  |                                                          ␣
↪                                                                               |
| network_id            | 38c5e950-b450-4c30-83d4-ee181c28aad3                     ␣
↪                                                                               |
| port_security_enabled | True                                                     ␣
↪                                                                               |
| project_id            | d5660cb1e6934612a01b4fb2fb630725                         ␣
↪                                                                               |
| qos_policy_id         | None                                                     ␣
↪                                                                               |
| revision_number       | 1                                                        ␣
↪                                                                               |
| security_group_ids    | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635                     ␣
↪                                                                               |
| status                | ACTIVE                                                   ␣
↪                                                                               |
| tags                  |                                                          ␣
↪                                                                               |
| trunk_details         | None                                                     ␣
↪                                                                               |
```

(continues on next page)

(continued from previous page)

```
| updated_at            | 2016-02-15T19:27:34Z                                         ␣
↪                                 |
+-----------------------+-----------------------------------------------------------
↪------------------------------------------------------------+

$ openstack recordset list example.org.
+--------------------------------------+----------------------+------+-------
↪--------------------------------------------------------------+--------
↪+--------+
| id                                   | name                 | type |␣
↪records                                                         |␣
↪status | action |
+--------------------------------------+----------------------+------+-------
↪--------------------------------------------------------------+--------
↪+--------+
| a5fe696d-203f-4018-b0d8-590221adb513 | example.org.         | NS   | ns1.
↪devstack.org.                                               |␣
↪ACTIVE | NONE   |
| e7c05a5d-83a0-4fe5-8bd5-ab058a3326aa | example.org.         | SOA  | ns1.
↪devstack.org. malavall.us.ibm.com. 1513767794 3532 600 86400 3600 |␣
↪ACTIVE | NONE   |
+--------------------------------------+----------------------+------+-------
↪--------------------------------------------------------------+--------
↪+--------+

$ openstack floating ip create 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a \
  --port da0b1f75-c895-460f-9fc1-4d6ec84cf85f
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2016-02-15T20:27:34Z                 |
| description         |                                      |
| dns_domain          |                                      |
| dns_name            |                                      |
| fixed_ip_address    | 192.0.2.15                           |
| floating_ip_address | 198.51.100.4                         |
| floating_network_id | 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a |
| id                  | e78f6eb1-a35f-4a90-941d-87c888d5fcc7 |
| name                | 198.51.100.4                         |
| port_id             | da0b1f75-c895-460f-9fc1-4d6ec84cf85f |
| project_id          | d5660cb1e6934612a01b4fb2fb630725     |
| qos_policy_id       | None                                 |
| revision_number     | 1                                    |
| router_id           | 970ebe83-c4a3-4642-810e-43ab7b0c2b5f |
| status              | DOWN                                 |
| subnet_id           | None                                 |
| tags                | []                                   |
| updated_at          | 2016-02-15T20:27:34Z                 |
+---------------------+--------------------------------------+

$ openstack recordset list example.org.
+--------------------------------------+----------------------+------+-------
↪--------------------------------------------------------------+--------
↪+--------+
| id                                   | name                 | type |␣
↪records                                                         |␣
↪status | action |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+--------------------+------+-------
↪---------------------------------------------------------------+--------
↪+--------+
| a5fe696d-203f-4018-b0d8-590221adb513 | example.org.       | NS   | ns1.
↪devstack.org.                                                  |␣
↪ACTIVE | NONE   |
| e7c05a5d-83a0-4fe5-8bd5-ab058a3326aa | example.org.       | SOA  | ns1.
↪devstack.org. malavall.us.ibm.com. 1513768814 3532 600 86400 3600 |␣
↪ACTIVE | NONE   |
| 5ff53fd0-3746-48da-b9c9-77ed3004ec67 | my-vm.example.org. | A    | 198.
↪51.100.4                                                       |␣
↪ACTIVE | NONE   |
+--------------------------------------+--------------------+------+-------
↪---------------------------------------------------------------+--------
↪+--------+
```

In this example, notice that the data is published in the DNS service when the floating IP is associated to the port.

Following are the PTR records created for this example. Note that for IPv4, the value of `ipv4_ptr_zone_prefix_size` is 24. Also, since the zone for the PTR records is created in the `service` project, you need to use admin credentials in order to be able to view it.

```
$ openstack recordset list --all-projects 100.51.198.in-addr.arpa.
+--------------------------------------+----------------------------------
↪+----------------------------+------+----------------------------------
↪---------------------------------+--------+--------+
| id                                   | project_id                      ␣
↪| name                       | type | data                            ␣
↪                                 | status | action |
+--------------------------------------+----------------------------------
↪+----------------------------+------+----------------------------------
↪---------------------------------+--------+--------+
| 2dd0b894-25fa-4563-9d32-9f13bd67f329 | 07224d17d76d42499a38f00ba4339710␣
↪| 100.51.198.in-addr.arpa.   | NS   | ns1.devstack.org.               ␣
↪                                 | ACTIVE | NONE   |
| 47b920f1-5eff-4dfa-9616-7cb5b7cb7ca6 | 07224d17d76d42499a38f00ba4339710␣
↪| 100.51.198.in-addr.arpa.   | SOA  | ns1.devstack.org. admin.example.
↪org. 1455564862 3600 600 86400 3600 | ACTIVE | NONE   |
| fb1edf42-abba-410c-8397-831f45fd0cd7 | 07224d17d76d42499a38f00ba4339710␣
↪| 4.100.51.198.in-addr.arpa. | PTR  | my-vm.example.org.              ␣
↪                                 | ACTIVE | NONE   |
+--------------------------------------+----------------------------------
↪+----------------------------+------+----------------------------------
↪---------------------------------+--------+--------+
```

Use case 2: Floating IPs are published in the external DNS service

In this use case, the user assigns `dns_name` and `dns_domain` attributes to a floating IP when it is created. The floating IP data becomes visible in the external DNS service as soon as it is created. The floating IP can be associated with a port on creation or later on. The following example shows a user booting an instance and then creating a floating IP associated to the port allocated for the instance:

```
$ openstack network show 38c5e950-b450-4c30-83d4-ee181c28aad3
+---------------------------+----------------------------------------------
↪-------------------------------+
| Field                     | Value                                        ␣
↪                               |
+---------------------------+----------------------------------------------
↪-------------------------------+
| admin_state_up            | UP                                           ␣
↪                               |
| availability_zone_hints   |                                              ␣
↪                               |
| availability_zones        | nova                                         ␣
↪                               |
| created_at                | 2016-05-04T19:27:34Z                         ␣
↪                               |
| description               |                                              ␣
↪                               |
| dns_domain                | example.org.                                 ␣
↪                               |
| id                        | 38c5e950-b450-4c30-83d4-ee181c28aad3         ␣
↪                               |
| ipv4_address_scope        | None                                         ␣
↪                               |
| ipv6_address_scope        | None                                         ␣
↪                               |
| is_default                | None                                         ␣
↪                               |
| is_vlan_transparent       | None                                         ␣
↪                               |
| mtu                       | 1450                                         ␣
↪                               |
| name                      | private                                      ␣
↪                               |
| port_security_enabled     | True                                         ␣
↪                               |
| project_id                | d5660cb1e6934612a01b4fb2fb630725             ␣
↪                               |
| provider:network_type     | vlan                                         ␣
↪                               |
| provider:physical_network | None                                         ␣
↪                               |
| provider:segmentation_id  | 24                                           ␣
↪                               |
| qos_policy_id             | None                                         ␣
↪                               |
| revision_number           | 1                                            ␣
↪                               |
| router:external           | Internal                                     ␣
↪                               |
| segments                  | None                                         ␣
↪                               |
```

(continues on next page)

(continued from previous page)

```
| shared                  | False                                                   ␣
↪                            |
| status                  | ACTIVE                                                  ␣
↪                            |
| subnets                 | 43414c53-62ae-49bc-aa6c-c9dd7705818a,␣
↪5b9282a1-0be1-4ade-b478-7868ad2a16ff |
| tags                    |                                                         ␣
↪                            |
| updated_at              | 2016-05-04T19:27:34Z                                    ␣
↪                            |
+-------------------------+---------------------------------------------------------
↪----------------------------+

$ openstack server create --image cirros --flavor 42 \
  --nic net-id=38c5e950-b450-4c30-83d4-ee181c28aad3 my_vm
+--------------------------------------+--------------------------------------------
↪----------------------------+
| Field                                | Value                                      ␣
↪                            |
+--------------------------------------+--------------------------------------------
↪----------------------------+
| OS-DCF:diskConfig                    | MANUAL                                     ␣
↪                            |
| OS-EXT-AZ:availability_zone          |                                            ␣
↪                            |
| 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                            | HLXGznYqXM4J                               ␣
↪                            |
| config_drive                         |                                            ␣
↪                            |
| created                              | 2016-02-15T19:42:44Z                       ␣
↪                            |
| flavor                               | m1.nano (42)                               ␣
↪                            |
| hostId                               |                                            ␣
↪                            |
| id                                   | 71fb4ac8-eed8-4644-8113-
↪0641962bb125                          |
| image                                | cirros-0.3.5-x86_64-uec (b9d981eb-
↪d21c-4ce2-9dbc-dd38f3d9015f) |
| key_name                             | -                                          ␣
↪                            |
| locked                               | False                                      ␣
↪                            |
```

(continues on next page)

(continued from previous page)

```
| metadata                             | {}                                ␣
↪                          |
| name                                 | my_vm                             ␣
↪                          |
| os-extended-volumes:volumes_attached | []                                ␣
↪                          |
| progress                             | 0                                 ␣
↪                          |
| security_groups                      | default                           ␣
↪                          |
| status                               | BUILD                             ␣
↪                          |
| tenant_id                            | d5660cb1e6934612a01b4fb2fb630725 ␣
↪                          |
| updated                              | 2016-02-15T19:42:44Z              ␣
↪                          |
| user_id                              | 8bb6e578cba24e7db9d3810633124525 ␣
↪                          |
+--------------------------------------+------------------------------------
↪----------------------------+

$ openstack server list
+--------------------------------------+-------+--------+------------------
↪----------------------------------------+--------+---------+
| ID                                   | Name  | Status | Networks          ␣
↪                                         | Image  | Flavor  |
+--------------------------------------+-------+--------+------------------
↪----------------------------------------+--------+---------+
| 71fb4ac8-eed8-4644-8113-0641962bb125 | my_vm | ACTIVE |␣
↪private=fda4:653e:71b0:0:f816:3eff:fe24:8614, 192.0.2.16 | cirros | m1.
↪nano |
+--------------------------------------+-------+--------+------------------
↪----------------------------------------+--------+---------+

$ openstack port list --device-id 71fb4ac8-eed8-4644-8113-0641962bb125
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                              ␣
↪                    | Status |
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+
| 1e7033fb-8e9d-458b-89ed-8312cafcfdcb |      | fa:16:3e:24:86:14 | ip_
↪address='192.0.2.16', subnet_id='5b9282a1-0be1-4ade-b478-7868ad2a16ff'  ␣
↪                       | ACTIVE |
|                                      |      |                   | ip_
↪address='fda4:653e:71b0:0:f816:3eff:fe24:8614', subnet_id='43414c53-62ae-
↪49bc-aa6c-c9dd7705818a' |        |
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+

$ openstack port show 1e7033fb-8e9d-458b-89ed-8312cafcfdcb
+-----------------------+---------------------------------------------------
↪-------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| Field                 | Value                                                     ␣
↪                                                                    |
+-----------------------+-----------------------------------------------------------
↪-----------------------------------------------------------+
| admin_state_up        | UP                                                        ␣
↪                                                                    |
| allowed_address_pairs |                                                           ␣
↪                                                                    |
| binding_host_id       | vultr.guest                                               ␣
↪                                                                    |
| binding_profile       |                                                           ␣
↪                                                                    |
| binding_vif_details   | datapath_type='system', ovs_hybrid_plug='True',␣
↪port_filter='True'                                                  |
| binding_vif_type      | ovs                                                       ␣
↪                                                                    |
| binding_vnic_type     | normal                                                    ␣
↪                                                                    |
| created_at            | 2016-02-15T19:42:44Z                                      ␣
↪                                                                    |
| data_plane_status     | None                                                      ␣
↪                                                                    |
| description           |                                                           ␣
↪                                                                    |
| device_id             | 71fb4ac8-eed8-4644-8113-0641962bb125                      ␣
↪                                                                    |
| device_owner          | compute:None                                              ␣
↪                                                                    |
| dns_assignment        | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='192.0.2.16'                                                |
|                       | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='fda4:653e:71b0:0:f816:3eff:fe24:8614'                      |
| dns_domain            | example.org.                                              ␣
↪                                                                    |
| dns_name              | my-vm                                                     ␣
↪                                                                    |
| extra_dhcp_opts       |                                                           ␣
↪                                                                    |
| fixed_ips             | ip_address='192.0.2.16', subnet_id='5b9282a1-
↪0be1-4ade-b478-7868ad2a16ff'                                        |
|                       | ip_address='fda4:653e:71b0:0:f816:3eff:fe24:8614
↪', subnet_id='43414c53-62ae-49bc-aa6c-c9dd7705818a'                 |
| id                    | 1e7033fb-8e9d-458b-89ed-8312cafcfdcb                      ␣
↪                                                                    |
| mac_address           | fa:16:3e:24:86:14                                         ␣
↪                                                                    |
| name                  |                                                           ␣
↪                                                                    |
| network_id            | 38c5e950-b450-4c30-83d4-ee181c28aad3                      ␣
↪                                                                    |
| port_security_enabled | True                                                      ␣
↪                                                                    |
| project_id            | d5660cb1e6934612a01b4fb2fb630725                          ␣
↪                                                                    |
| qos_policy_id         | None                                                      ␣
↪                                                                    |
```

(continues on next page)

(continued from previous page)

```
| revision_number       | 1                                                    ␣
↪                                                                            |
| security_group_ids    | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635                 ␣
↪                                                                            |
| status                | ACTIVE                                               ␣
↪                                                                            |
| tags                  |                                                      ␣
↪                                                                            |
| trunk_details         | None                                                 ␣
↪                                                                            |
| updated_at            | 2016-02-15T19:42:44Z                                 ␣
↪                                                                            |
+-----------------------+------------------------------------------------------
↪------------------------------------------------------------+

$ openstack recordset list example.org.
+--------------------------------------+--------------------+------+-------
↪-----------------------------------------------------------------+--------
↪+---------+
| id                                   | name               | type |␣
↪records                                                              |␣
↪status | action |
+--------------------------------------+--------------------+------+-------
↪-----------------------------------------------------------------+--------
↪+---------+
| 56ca0b88-e343-4c98-8faa-19746e169baf | example.org.       | NS   | ns1.
↪devstack.org.                                                   |␣
↪ACTIVE | NONE    |
| 10a36008-6ecf-47c3-b321-05652a929b04 | example.org.       | SOA  | ns1.
↪devstack.org. malavall.us.ibm.com. 1455565110 3532 600 86400 3600 |␣
↪ACTIVE | NONE    |
+--------------------------------------+--------------------+------+-------
↪-----------------------------------------------------------------+--------
↪+---------+

$ openstack floating ip create --dns-domain example.org. --dns-name my-
↪floatingip 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2019-06-12T15:54:45Z                 |
| description         |                                      |
| dns_domain          | example.org.                         |
| dns_name            | my-floatingip                        |
| fixed_ip_address    | None                                 |
| floating_ip_address | 198.51.100.5                         |
| floating_network_id | 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a |
| id                  | 3ae82f53-3349-4aac-810e-ed2a8f6374b8 |
| name                | 198.51.100.53                        |
| port_details        | None                                 |
| port_id             | None                                 |
| project_id          | d5660cb1e6934612a01b4fb2fb630725     |
| qos_policy_id       | None                                 |
| revision_number     | 0                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
```

(continues on next page)

(continued from previous page)

```
| subnet_id           | None                                 |
| tags                | []                                   |
| updated_at          | 2019-06-12T15:54:45Z                 |
+---------------------+--------------------------------------+

$ openstack recordset list example.org.
+--------------------------------------+-----------------------------+------
↪+-------------------------------------------------------------------------
↪+--------+--------+
| id                                   | name                        | type␣
↪| records                                                                  ␣
↪| status | action |
+--------------------------------------+-----------------------------+------
↪+-------------------------------------------------------------------------
↪+--------+--------+
| 56ca0b88-e343-4c98-8faa-19746e169baf | example.org.                | NS  ␣
↪| ns1.devstack.org.                                                        ␣
↪| ACTIVE | NONE   |
| 10a36008-6ecf-47c3-b321-05652a929b04 | example.org.                | SOA ␣
↪| ns1.devstack.org. malavall.us.ibm.com. 1455565110 3532 600 86400 3600␣
↪| ACTIVE | NONE   |
| 8884c56f-3ef5-446e-ae4d-8053cc8bc2b4 | my-floatingip.example.org.  | A   ␣
↪| 198.51.100.53                                                            ␣
↪| ACTIVE | NONE   |
+--------------------------------------+-----------------------------+------
↪+-------------------------------------------------------------------------
↪+--------+--------+
```

Note that in this use case:

- The `dns_name` and `dns_domain` attributes of a floating IP must be specified together on creation. They cannot be assigned to the floating IP separately and they cannot be changed after the floating IP has been created.
- The `dns_name` and `dns_domain` of a floating IP have precedence, for purposes of being published in the external DNS service, over the `dns_name` of its associated port and the `dns_domain` of the ports network, whether they are specified or not. Only the `dns_name` and the `dns_domain` of the floating IP are published in the external DNS service.

Following are the PTR records created for this example. Note that for IPv4, the value of `ipv4_ptr_zone_prefix_size` is 24. Also, since the zone for the PTR records is created in the project specified in the `[designate]` section in the config above, usually the `service` project, you need to use admin credentials in order to be able to view it.

```
$ openstack recordset list --all-projects 100.51.198.in-addr.arpa.
+--------------------------------------+-----------------------------------
↪+----------------------------+-------+------------------------------------
↪---------------------------------+--------+--------+
| id                                   | project_id                        ␣
↪| name                       | type | data                                ␣
↪                                  | status | action |
+--------------------------------------+-----------------------------------
↪+----------------------------------+------------------------------------
↪---------------------------------+--------+--------+
| 2dd0b894-25fa-4563-9d32-9f13bd67f329 | 07224d17d76d42499a38f00ba4339710␣
↪| 100.51.198.in-addr.arpa.    | NS   | ns1.devstack.org.                   ␣
↪                                  | ACTIVE | NONE   |
```

(continues on next page)

(continued from previous page)

```
| 47b920f1-5eff-4dfa-9616-7cb5b7cb7ca6 | 07224d17d76d42499a38f00ba4339710␣
↪| 100.51.198.in-addr.arpa.   | SOA  | ns1.devstack.org. admin.example.
↪org. 1455564862 3600 600 86400 3600 | ACTIVE | NONE   |
| 589a0171-e77a-4ab6-ba6e-23114f2b9366 | 07224d17d76d42499a38f00ba4339710␣
↪| 5.100.51.198.in-addr.arpa. | PTR  | my-floatingip.example.org.      ␣
↪                                 | ACTIVE | NONE   |
+--------------------------------------+----------------------------------
↪+----------------------------+------+------------------------------------
↪---------------------------------+--------+--------+
```

Use case 3: Ports are published directly in the external DNS service

In this case, the user is creating ports or booting instances on a network that is accessible externally. There are multiple possible scenarios here depending on which of the DNS extensions is enabled in the Neutron configuration. These extensions are described in the following in descending order of priority.

Use case 3a: The `subnet_dns_publish_fixed_ip` extension

When the `subnet_dns_publish_fixed_ip` extension is enabled, it is possible to make a selection per subnet whether DNS records should be published for fixed IPs that are assigned to ports from that subnet. This happens via the `dns_publish_fixed_ips` attribute that this extension adds to the definition of the subnet resource. It is a boolean flag with a default value of `False` but it can be set to `True` when creating or updating subnets. When the flag is `True`, all fixed IPs from this subnet are published in the external DNS service, while at the same time IPs from other subnets having the flag set to `False` are not published, even if they otherwise would meet the criteria from the other use cases below.

A typical scenario for this use case is a dual stack deployment, where a tenant network would be configured with both an IPv4 and an IPv6 subnet. The IPv4 subnet will usually be using some RFC1918 address space and being NATted towards the outside on the attached router, therefore the fixed IPs from this subnet will not be globally routed and they also should not be published in the DNS service. (One can still bind floating IPs to these fixed IPs and DNS records for those floating IPs can still be published as described above in use cases 1 and 2).

But for the IPv6 subnet, no NAT will happen, instead the subnet will be configured with some globally routable prefix and thus the user will want to publish DNS records for fixed IPs from this subnet. This can be achieved by setting the `dns_publish_fixed_ips` attribute for the IPv6 subnet to `True` while leaving the flag set to `False` for the IPv4 subnet. Example:

```
$ openstack network create dualstack
... output omitted ...
$ openstack subnet create --network dualstack dualstackv4 --subnet-range␣
↪192.0.2.0/24
... output omitted ...
$ openstack subnet create --network dualstack dualstackv6 --protocol ipv6 -
↪-subnet-range 2001:db8:42:42::/64 --dns-publish-fixed-ip
... output omitted ...
$ openstack zone create example.org. --email mail@example.org
... output omitted ...
$ openstack recordset list example.org.
+--------------------------------------+--------------+------+------------
↪                                                              +
↪+
```

(continues on next page)

(continued from previous page)

```
| id                                   | name         | type | records  ␣
↪                                                      | status | action␣
↪|
+--------------------------------------+--------------+------+-----------
↪-------------------------------------------------------+--------+-------
↪+
| 404e9846-1482-433b-8bbc-67677e587d28 | example.org. | NS   | ns1.
↪devstack.org.                                                | ACTIVE␣
↪| NONE   |
| de73576a-f9c7-4892-934c-259b77ff02c0 | example.org. | SOA  | ns1.
↪devstack.org. mail.example.org. 1575897792 3559 600 86400 3600 | ACTIVE␣
↪| NONE   |
+--------------------------------------+--------------+------+-----------
↪-------------------------------------------------------+--------+-------
↪+
$ openstack port create port1 --dns-domain example.org. --dns-name port1 --
↪network dualstack
+-------------------------+-----------------------------------------------
↪-------------------------------------------------------------------------
↪-----------------------------------------------+
| Field                   | Value                                         ␣
↪                                                                         ␣
↪                                               |
+-------------------------+-----------------------------------------------
↪-------------------------------------------------------------------------
↪-----------------------------------------------+
| admin_state_up          | UP                                            ␣
↪                                                                         ␣
↪                                               |
| allowed_address_pairs   |                                               ␣
↪                                                                         ␣
↪                                               |
| binding_host_id         | None                                          ␣
↪                                                                         ␣
↪                                               |
| binding_profile         | None                                          ␣
↪                                                                         ␣
↪                                               |
| binding_vif_details     | None                                          ␣
↪                                                                         ␣
↪                                               |
| binding_vif_type        | None                                          ␣
↪                                                                         ␣
↪                                               |
| binding_vnic_type       | normal                                        ␣
↪                                                                         ␣
↪                                               |
| created_at              | 2019-12-09T13:23:52Z                          ␣
↪                                                                         ␣
↪                                               |
| data_plane_status       | None                                          ␣
↪                                                                         ␣
↪                                               |
| description             |                                               ␣
↪                                                                         ␣
↪                                               |
```

(continues on next page)

(continued from previous page)

```
| device_id               |
↪
↪                                                         |
| device_owner            |
↪
↪                                                         |
| dns_assignment          | fqdn='port1.openstackgate.local.', hostname=
↪'port1', ip_address='192.0.2.100'
↪                                                         |
|                         | fqdn='port1.openstackgate.local.', hostname=
↪'port1', ip_address='2001:db8:42:42::2a2'
↪                                                         |
| dns_domain              | example.org.
↪
↪                                                         |
| dns_name                | port1
↪
↪                                                         |
| extra_dhcp_opts         |
↪
↪                                                         |
| fixed_ips               | ip_address='192.0.2.100', subnet_id='47cc9a39-
↪c88b-4082-a52c-1237c2a1d479'
↪                                                         |
|                         | ip_address='2001:db8:42:42::2a2', subnet_id=
↪'f9c04195-1000-4575-a203-3c174772617f'
↪                                                         |
| id                      | f8bc991b-1f84-435a-a5f8-814bd8b9ae9f
↪
↪                                                         |
| location                | cloud='devstack', project.domain_id='default',
↪project.domain_name=, project.id='86de4dab952d48f79e625b106f7a75f7',
↪project.name='demo', region_name='RegionOne', zone= |
| mac_address             | fa:16:3e:13:7a:56
↪
↪                                                         |
| name                    | port1
↪
↪                                                         |
| network_id              | fa8118ed-b7c2-41b8-89bc-97e46f0491ac
↪
↪                                                         |
| port_security_enabled   | True
↪
↪                                                         |
| project_id              | 86de4dab952d48f79e625b106f7a75f7
↪
↪                                                         |
| propagate_uplink_status | None
↪
↪                                                         |
| qos_policy_id           | None
↪
↪                                                         |
| resource_request        | None
↪
↪                                                         |
```

(continues on next page)

(continued from previous page)

```
| revision_number         | 1                                                       ␣
↪                                                                                   ␣
↪                                               |
| security_group_ids      | f0b02df0-a0b9-4ce8-b067-8b61a8679e9d                    ␣
↪                                                                                   ␣
↪                                               |
| status                  | DOWN                                                    ␣
↪                                                                                   ␣
↪                                               |
| tags                    |                                                         ␣
↪                                                                                   ␣
↪                                               |
| trunk_details           | None                                                    ␣
↪                                                                                   ␣
↪                                               |
| updated_at              | 2019-12-09T13:23:53Z                                    ␣
↪                                                                                   ␣
↪                                               |
+-------------------------+-------------------------------------------------------
↪-----------------------------------------------------------------------------------
↪-----------------------------------------------+
$ openstack recordset list example.org.
+--------------------------------------+--------------------+------+-------
↪----------------------------------------------------------------+--------+--
↪------+
| id                                   | name               | type |␣
↪records                                                          |␣
↪status | action |
+--------------------------------------+--------------------+------+-------
↪----------------------------------------------------------------+--------+--
↪------+
| 404e9846-1482-433b-8bbc-67677e587d28 | example.org.       | NS   | ns1.
↪devstack.org.                                              | ACTIVE␣
↪| NONE   |
| de73576a-f9c7-4892-934c-259b77ff02c0 | example.org.       | SOA  | ns1.
↪devstack.org. mail.example.org. 1575897833 3559 600 86400 3600 | ACTIVE␣
↪| NONE   |
| 85ce74a5-7dd6-42d3-932c-c9a029dea05e | port1.example.org. | AAAA |␣
↪2001:db8:42:42::2a2                                              |␣
↪ACTIVE | NONE   |
+--------------------------------------+--------------------+------+-------
↪----------------------------------------------------------------+--------+--
↪------+
```

Use case 3b: The `dns_domain_ports` extension

If the `dns_domain for ports` extension has been configured, the user can create a port specifying a non-blank value in its `dns_domain` attribute. If the port is created in an externally accessible network, DNS records will be published for this port:

```
$ openstack port create --network 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 --
↪dns-name my-vm --dns-domain port-domain.org. test
+-------------------------+-------------------------------------------------------
↪-------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Field                 | Value                                                                    ␣
↪                              |
+-----------------------+-------------------------------------------------
↪------------------------------+
| admin_state_up        | UP                                                                       ␣
↪                              |
| allowed_address_pairs |                                                                          ␣
↪                              |
| binding_host_id       | None                                                                     ␣
↪                              |
| binding_profile       | None                                                                     ␣
↪                              |
| binding_vif_details   | None                                                                     ␣
↪                              |
| binding_vif_type      | None                                                                     ␣
↪                              |
| binding_vnic_type     | normal                                                                   ␣
↪                              |
| created_at            | 2019-06-12T15:43:29Z                                                     ␣
↪                              |
| data_plane_status     | None                                                                     ␣
↪                              |
| description           |                                                                          ␣
↪                              |
| device_id             |                                                                          ␣
↪                              |
| device_owner          |                                                                          ␣
↪                              |
| dns_assignment        | fqdn='my-vm.example.org.', hostname='my-vm',␣
↪ip_address='203.0.113.9'           |
|                       | fqdn='my-vm.example.org.', hostname='my-vm',␣
↪ip_address='2001:db8:10::9'        |
| dns_domain            | port-domain.org.                                                         ␣
↪                              |
| dns_name              | my-vm                                                                    ␣
↪                              |
| extra_dhcp_opts       |                                                                          ␣
↪                              |
| fixed_ips             | ip_address='203.0.113.9', subnet_id='277eca5d-
↪9869-474b-960e-6da5951d09f7'     |
|                       | ip_address='2001:db8:10::9', subnet_id=
↪'eab47748-3f0a-4775-a09f-b0c24bb64bc4' |
| id                    | 57541c27-f8a9-41f1-8dde-eb10155496e6                                     ␣
↪                              |
| mac_address           | fa:16:3e:55:d6:c7                                                        ␣
↪                              |
| name                  | test                                                                     ␣
↪                              |
| network_id            | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3                                     ␣
↪                              |
| port_security_enabled | True                                                                     ␣
↪                              |
| project_id            | 07b21ad4-edb6-420b-bd76-9bb4aab0d135                                     ␣
↪                              |
| propagate_uplink_status | None                                                                   ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| qos_policy_id         | None                                                  ␣
↪                               |
| resource_request      | None                                                  ␣
↪                               |
| revision_number       | 1                                                     ␣
↪                               |
| security_group_ids    | 82227b10-d135-4bca-b41f-63c1f2286b3e                  ␣
↪                               |
| status                | DOWN                                                  ␣
↪                               |
| tags                  |                                                       ␣
↪                               |
| trunk_details         | None                                                  ␣
↪                               |
| updated_at            | 2019-06-12T15:43:29Z                                  ␣
↪                               |
+-----------------------+-------------------------------------------------------
↪-------------------------------+
```

In this case, the ports `dns_name` (`my-vm`) will be published in the `port-domain.org.` zone, as shown here:

```
$ openstack recordset list port-domain.org.
+--------------------------------------+-------------------------+------+--
↪-----------------------------------------------------------------------+---
↪-----+---------+
| id                                   | name                    | type |␣
↪records                                                                 |␣
↪status | action |
+--------------------------------------+-------------------------+------+--
↪-----------------------------------------------------------------------+---
↪-----+---------+
| 03e5a35b-d984-4d10-942a-2de8ccb9b941 | port-domain.org.        | SOA  |␣
↪ns1.devstack.org. malavall.us.ibm.com. 1503272259 3549 600 86400 3600 |␣
↪ACTIVE | NONE   |
| d2dd1dfe-531d-4fea-8c0e-f5b559942ac5 | port-domain.org.        | NS   |␣
↪ns1.devstack.org.                                                       |␣
↪ACTIVE | NONE   |
| 67a8e83d-7e3c-4fb1-9261-0481318bb7b5 | my-vm.port-domain.org.  | A    |␣
↪203.0.113.9                                                             |␣
↪ACTIVE | NONE   |
| 5a4f671c-9969-47aa-82e1-e05754021852 | my-vm.port-domain.org.  | AAAA |␣
↪2001:db8:10::9                                                          |␣
↪ACTIVE | NONE   |
+--------------------------------------+-------------------------+------+--
↪-----------------------------------------------------------------------+---
↪-----+---------+
```

Note: If both the port and its network have a valid non-blank string assigned to their `dns_domain` attributes, the ports `dns_domain` takes precedence over the networks.

Note: The name assigned to the ports `dns_domain` attribute must end with a period (`.`).

Note: In the above example, the `port-domain.org.` zone must be created before Neutron can publish any port data to it.

Note: See *Configuration of the externally accessible network for use cases 3b and 3c* for detailed instructions on how to create the externally accessible network.

Use case 3c: The `dns` extension

If the user wants to publish a port in the external DNS service in a zone specified by the `dns_domain` attribute of the network, these are the steps to be taken:

1. Assign a valid domain name to the networks `dns_domain` attribute. This name must end with a period (`.`).
2. Boot an instance specifying the externally accessible network. Alternatively, create a port on the externally accessible network specifying a valid value to its `dns_name` attribute. If the port is going to be used for an instance boot, the value assigned to `dns_name` must be equal to the `hostname` that the Compute service will assign to the instance. Otherwise, the boot will fail.

Once these steps are executed, the ports DNS data will be published in the external DNS service. This is an example:

```
$ openstack network list
+--------------------------------------+----------+-----------------------
↪-----------------------------------------------------+
| ID                                   | Name     | Subnets               ␣
↪                                                     |
+--------------------------------------+----------+-----------------------
↪-----------------------------------------------------+
| 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a | public   | a67cfdf7-9d5d-406f-
↪8a19-3f38e4fc3e74, cbd8c6dc-ca81-457e-9c5d-f8ece7ef67f8   |
| 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 | external | 277eca5d-9869-474b-
↪960e-6da5951d09f7, eab47748-3f0a-4775-a09f-b0c24bb64bc4   |
| bf2802a0-99a0-4e8c-91e4-107d03f158ea | my-net   | 6141b474-56cd-430f-
↪b731-71660bb79b79                                         |
| 38c5e950-b450-4c30-83d4-ee181c28aad3 | private  | 43414c53-62ae-49bc-
↪aa6c-c9dd7705818a, 5b9282a1-0be1-4ade-b478-7868ad2a16ff   |
+--------------------------------------+----------+-----------------------
↪-----------------------------------------------------+

$ openstack network set --dns-domain example.org. 37aaff3a-6047-45ac-bf4f-
↪a825e56fd2b3

$ openstack network show 37aaff3a-6047-45ac-bf4f-a825e56fd2b3
+---------------------------+---------------------------------------------
↪----------------------------+
| Field                     | Value                                       ␣
↪                            |
+---------------------------+---------------------------------------------
↪----------------------------+
| admin_state_up            | UP                                          ␣
↪                            |
```

(continues on next page)

(continued from previous page)

```
| availability_zone_hints   |                                                ␣
↪                              |
| availability_zones        | nova                                           ␣
↪                              |
| created_at                | 2016-02-14T19:42:44Z                           ␣
↪                              |
| description               |                                                ␣
↪                              |
| dns_domain                | example.org.                                   ␣
↪                              |
| id                        | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3           ␣
↪                              |
| ipv4_address_scope        | None                                           ␣
↪                              |
| ipv6_address_scope        | None                                           ␣
↪                              |
| is_default                | None                                           ␣
↪                              |
| is_vlan_transparent       | None                                           ␣
↪                              |
| mtu                       | 1450                                           ␣
↪                              |
| name                      | external                                       ␣
↪                              |
| port_security_enabled     | True                                           ␣
↪                              |
| project_id                | 04fc2f83966245dba907efb783f8eab9               ␣
↪                              |
| provider:network_type     | vlan                                           ␣
↪                              |
| provider:physical_network | None                                           ␣
↪                              |
| provider:segmentation_id  | 2016                                           ␣
↪                              |
| qos_policy_id             | None                                           ␣
↪                              |
| revision_number           | 4                                              ␣
↪                              |
| router:external           | Internal                                       ␣
↪                              |
| segments                  | None                                           ␣
↪                              |
| shared                    | True                                           ␣
↪                              |
| status                    | ACTIVE                                         ␣
↪                              |
| subnets                   | eab47748-3f0a-4775-a09f-b0c24bb64bc4,␣
↪277eca5d-9869-474b-960e-6da5951d09f7 |
| tags                      |                                                ␣
↪                              |
| updated_at                | 2016-02-15T13:42:44Z                           ␣
↪                              |
+---------------------------+-----------------------------------------------
↪-----------------------------+

$ openstack recordset list example.org.
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+--------------+------+-------------
↪-----------------------------------------------------------+--------+-----
↪---+
| id                                   | name         | type | records    ␣
↪                                                            | status |␣
↪action |
+--------------------------------------+--------------+------+-------------
↪-----------------------------------------------------------+--------+-----
↪---+
| a5fe696d-203f-4018-b0d8-590221adb513 | example.org. | NS   | ns1.
↪devstack.org.                                              |␣
↪ACTIVE | NONE   |
| e7c05a5d-83a0-4fe5-8bd5-ab058a3326aa | example.org. | SOA  | ns1.
↪devstack.org. malavall.us.ibm.com. 1513767619 3532 600 86400 3600 |␣
↪ACTIVE | NONE   |
+--------------------------------------+--------------+------+-------------
↪-----------------------------------------------------------+--------+-----
↪---+

$ openstack port create --network 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 --
↪dns-name my-vm test
+-----------------------+-----------------------------------------------------
↪------------------------------+
| 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            | 2016-02-15T16:42:44Z                               ␣
↪                              |
| data_plane_status     | None                                               ␣
↪                              |
| description           |                                                    ␣
↪                              |
| device_id             |                                                    ␣
↪                              |
| device_owner          |                                                    ␣
↪                              |
| dns_assignment        | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='203.0.113.9'         |
|                       | fqdn='my-vm.example.org.', hostname='my-vm', ip_
↪address='2001:db8:10::9'      |
| dns_domain            | None                                               ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| dns_name              | my-vm                                                                      ␣
↪                       |
| extra_dhcp_opts       |                                                                            ␣
↪                       |
| fixed_ips             | ip_address='203.0.113.9', subnet_id='277eca5d-
↪9869-474b-960e-6da5951d09f7'     |
|                       | ip_address='2001:db8:10::9', subnet_id=eab47748-
↪3f0a-4775-a09f-b0c24bb64bc4 |
| id                    | 04be331b-dc5e-410a-9103-9c8983aeb186                                       ␣
↪                       |
| mac_address           | fa:16:3e:0f:4b:e4                                                          ␣
↪                       |
| name                  | test                                                                       ␣
↪                       |
| network_id            | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3                                       ␣
↪                       |
| port_security_enabled | True                                                                       ␣
↪                       |
| project_id            | d5660cb1e6934612a01b4fb2fb630725                                           ␣
↪                       |
| qos_policy_id         | None                                                                       ␣
↪                       |
| revision_number       | 1                                                                          ␣
↪                       |
| security_group_ids    | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635                                       ␣
↪                       |
| status                | DOWN                                                                       ␣
↪                       |
| tags                  |                                                                            ␣
↪                       |
| trunk_details         | None                                                                       ␣
↪                       |
| updated_at            | 2016-02-15T16:42:44Z                                                       ␣
↪                       |
+-----------------------+------------------------------------------------------------------------------
↪-----------------------------+

$ openstack recordset list example.org.
+--------------------------------------+--------------------+------+-------
↪----------------------------------------------------------------+--------
↪+--------+
| id                                   | name               | type |␣
↪records                                                         |␣
↪status | action |
+--------------------------------------+--------------------+------+-------
↪----------------------------------------------------------------+--------
↪+--------+
| a5fe696d-203f-4018-b0d8-590221adb513 | example.org.       | NS   | ns1.
↪devstack.org.                                                   |␣
↪ACTIVE | NONE   |
| e7c05a5d-83a0-4fe5-8bd5-ab058a3326aa | example.org.       | SOA  | ns1.
↪devstack.org. malavall.us.ibm.com. 1513767794 3532 600 86400 3600 |␣
↪ACTIVE | NONE   |
| fa753ab8-bffa-400d-9ef8-d4a3b1a7ffbf | my-vm.example.org. | A    | 203.0.
↪113.9                                                           | ACTIVE␣
↪| NONE   |
```

(continues on next page)

(continued from previous page)

```
| 04abf9f8-c7a3-43f6-9a55-95cee9b144a9 | my-vm.example.org. | AAAA |␣
↪2001:db8:10::9                                                    |␣
↪ACTIVE | NONE    |
+--------------------------------------+--------------------+------+-------
↪-------------------------------------------------------------------+--------
↪+---------+

$ openstack server create --image cirros --flavor 42 \
  --nic port-id=04be331b-dc5e-410a-9103-9c8983aeb186 my_vm
+--------------------------------------+-------------------------------------
↪-----------------------------+
| Field                                | Value                              ␣
↪                             |
+--------------------------------------+-------------------------------------
↪-----------------------------+
| OS-DCF:diskConfig                    | MANUAL                             ␣
↪                             |
| OS-EXT-AZ:availability_zone          |                                    ␣
↪                             |
| 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                            | TDc9EpBT3B9W                       ␣
↪                             |
| config_drive                         |                                    ␣
↪                             |
| created                              | 2016-02-15T19:10:43Z               ␣
↪                             |
| flavor                               | m1.nano (42)                       ␣
↪                             |
| hostId                               |                                    ␣
↪                             |
| id                                   | 62c19691-d1c7-4d7b-a88e-
↪9cc4d95d4f41                         |
| image                                | cirros-0.3.5-x86_64-uec (b9d981eb-
↪d21c-4ce2-9dbc-dd38f3d9015f) |
| key_name                             | -                                  ␣
↪                             |
| locked                               | False                              ␣
↪                             |
| metadata                             | {}                                 ␣
↪                             |
| name                                 | my_vm                              ␣
↪                             |
| os-extended-volumes:volumes_attached | []                                 ␣
↪                             |
```

(continues on next page)

(continued from previous page)

```
| progress                             | 0                                ␣
↪                                 |
| security_groups                      | default                          ␣
↪                                 |
| status                               | BUILD                            ␣
↪                                 |
| tenant_id                            | d5660cb1e6934612a01b4fb2fb630725 ␣
↪                                 |
| updated                              | 2016-02-15T19:10:43Z             ␣
↪                                 |
| user_id                              | 8bb6e578cba24e7db9d3810633124525 ␣
↪                                 |
+--------------------------------------+----------------------------------
↪-------------------------------+

$ openstack server list
+--------------------------------------+-------+--------+-----------------
↪--------------------+--------+---------+
| ID                                   | Name  | Status | Networks         ␣
↪                    | Image  | Flavor  |
+--------------------------------------+-------+--------+-----------------
↪--------------------+--------+---------+
| 62c19691-d1c7-4d7b-a88e-9cc4d95d4f41 | my_vm | ACTIVE | external=203.0.
↪113.9, 2001:db8:10::9 | cirros | m1.nano |
+--------------------------------------+-------+--------+-----------------
↪--------------------+--------+---------+
```

In this example the port is created manually by the user and then used to boot an instance. Notice that:

- The ports data was visible in the DNS service as soon as it was created.
- See *Performance considerations* for an explanation of the potential performance impact associated with this use case.

Following are the PTR records created for this example. Note that for IPv4, the value of `ipv4_ptr_zone_prefix_size` is 24. In the case of IPv6, the value of `ipv6_ptr_zone_prefix_size` is 116.

```
$ openstack recordset list --all-projects 113.0.203.in-addr.arpa.
+--------------------------------------+----------------------------------
↪+---------------------------+------+-----------------------------------
↪---------------------------------+--------+--------+
| id                                   | project_id                      ␣
↪| name                      | type | records                          ␣
↪                                 | status | action |
+--------------------------------------+----------------------------------
↪+---------------------------+------+-----------------------------------
↪---------------------------------+--------+--------+
| 32f1c05b-7c5d-4230-9088-961a0a462d28 | 07224d17d76d42499a38f00ba4339710␣
↪| 113.0.203.in-addr.arpa.   | SOA  | ns1.devstack.org. admin.example.org.
↪ 1455563035 3600 600 86400 3600 | ACTIVE | NONE   |
| 3d402c43-b215-4a75-a730-51cbb8999cb8 | 07224d17d76d42499a38f00ba4339710␣
↪| 113.0.203.in-addr.arpa.   | NS   | ns1.devstack.org.                ␣
↪                                 | ACTIVE | NONE   |
| 8e4e618c-24b0-43db-ab06-91b741a91c10 | 07224d17d76d42499a38f00ba4339710␣
↪| 9.113.0.203.in-addr.arpa. | PTR  | my-vm.example.org.               ␣
↪                                 | ACTIVE | NONE   |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+----------------------------------
↪+----------------------------+-------+------------------------------------
↪--------------------------------+---------+---------+

$ openstack recordset list --all-projects  0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
↪0.0.1.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
+--------------------------------------+----------------------------------
↪+-------------------------------------------------------------------------
↪---+------+---------------------------------------------------------------
↪-------+---------+---------+
| id                                   | project_id                        ␣
↪| name                                                                     ␣
↪   | type | records                                                        ␣
↪       | status | action |
+--------------------------------------+----------------------------------
↪+-------------------------------------------------------------------------
↪---+------+---------------------------------------------------------------
↪-------+---------+---------+
| d8923354-13eb-4bd9-914a-0a2ae5f95989 | 07224d17d76d42499a38f00ba4339710␣
↪| 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.8.b.d.0.1.0.0.2.ip6.arpa.    ␣
↪   | SOA  | ns1.devstack.org. admin.example.org. 1455563036 3600 600␣
↪86400 3600 | ACTIVE | NONE   |
| 72e60acd-098d-41ea-9771-5b6546c9c06f | 07224d17d76d42499a38f00ba4339710␣
↪| 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.8.b.d.0.1.0.0.2.ip6.arpa.    ␣
↪   | NS   | ns1.devstack.org.                                              ␣
↪       | ACTIVE | NONE   |
| 877e0215-2ddf-4d01-a7da-47f1092dfd56 | 07224d17d76d42499a38f00ba4339710␣
↪| 9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.8.b.d.0.1.0.0.2.ip6.
↪arpa. | PTR  | my-vm.example.org.                                         ␣
↪           | ACTIVE | NONE   |
+--------------------------------------+----------------------------------
↪+-------------------------------------------------------------------------
↪---+------+---------------------------------------------------------------
↪-------+---------+---------+
```

See *Configuration of the externally accessible network for use cases 3b and 3c* for detailed instructions on how to create the externally accessible network.

Performance considerations

Only for *Use case 3: Ports are published directly in the external DNS service*, if the port binding extension is enabled in the Networking service, the Compute service will execute one additional port update operation when allocating the port for the instance during the boot process. This may have a noticeable adverse effect in the performance of the boot process that should be evaluated before adoption of this use case.

Configuration of the externally accessible network for use cases 3b and 3c

For use cases 3b and 3c, the externally accessible network must meet the following requirements:

- The network may not have attribute `router:external` set to `True`.
- The network type can be FLAT, VLAN, GRE, VXLAN or GENEVE.
- For network types VLAN, GRE, VXLAN or GENEVE, the segmentation ID must be outside the ranges assigned to project networks.

This usually implies that these use cases only work for networks specifically created for this purpose by an admin, they do not work for networks which tenants can create on their own.

8.2.10 DNS resolution for instances

The Networking service offers several methods to configure name resolution (DNS) for instances. Most deployments should implement case 1 or 2a. Case 2b requires security considerations to prevent leaking internal DNS information to instances.

Note: All of these setups require the configured DNS resolvers to be reachable from the virtual network in question. So unless the resolvers are located inside the virtual network itself, this implies the need for a router to be attached to that network having an external gateway configured.

Case 1: Each virtual network uses unique DNS resolver(s)

In this case, the DHCP agent offers one or more unique DNS resolvers to instances via DHCP on each virtual network. You can configure a DNS resolver when creating or updating a subnet. To configure more than one DNS resolver, repeat the option multiple times.

- Configure a DNS resolver when creating a subnet.

  ```
  $ openstack subnet create --dns-nameserver DNS_RESOLVER
  ```

  Replace `DNS_RESOLVER` with the IP address of a DNS resolver reachable from the virtual network. Repeat the option if you want to specify multiple IP addresses. For example:

  ```
  $ openstack subnet create --dns-nameserver 203.0.113.8 --dns-
  ↪nameserver 198.51.100.53
  ```

  Note: This command requires additional options outside the scope of this content.

- Add a DNS resolver to an existing subnet.

  ```
  $ openstack subnet set --dns-nameserver DNS_RESOLVER SUBNET_ID_OR_NAME
  ```

  Replace `DNS_RESOLVER` with the IP address of a DNS resolver reachable from the virtual network and `SUBNET_ID_OR_NAME` with the UUID or name of the subnet. For example, using the `selfservice` subnet:

```
$ openstack subnet set --dns-nameserver 203.0.113.9 selfservice
```

- Remove all DNS resolvers from a subnet.

```
$ openstack subnet set --no-dns-nameservers SUBNET_ID_OR_NAME
```

Replace `SUBNET_ID_OR_NAME` with the UUID or name of the subnet. For example, using the `selfservice` subnet:

```
$ openstack subnet set --no-dns-nameservers selfservice
```

Note: You can use this option in combination with the previous one in order to replace all existing DNS resolver addresses with new ones.

You can also set the DNS resolver address to `0.0.0.0` for IPv4 subnets, or `::` for IPv6 subnets, which are special values that indicate to the DHCP agent that it should not announce any DNS resolver at all on the subnet.

Note: When DNS resolvers are explicitly specified for a subnet this way, that setting will take precedence over the options presented in case 2.

Case 2: DHCP agents forward DNS queries from instances

In this case, the DHCP agent offers the list of all DHCP agents IP addresses on a subnet as DNS resolver(s) to instances via DHCP on that subnet.

The DHCP agent then runs a masquerading forwarding DNS resolver with two possible options to determine where the DNS queries are sent to.

Note: The DHCP agent will answer queries for names and addresses of instances running within the virtual network directly instead of forwarding them.

Case 2a: Queries are forwarded to an explicitly configured set of DNS resolvers

In the `dhcp_agent.ini` file, configure one or more DNS resolvers. To configure more than one DNS resolver, use a comma between the values.

```
[DEFAULT]
dnsmasq_dns_servers = DNS_RESOLVER
```

Replace `DNS_RESOLVER` with a list of IP addresses of DNS resolvers reachable from all virtual networks. For example:

```
[DEFAULT]
dnsmasq_dns_servers = 203.0.113.8, 198.51.100.53
```

Note: You must configure this option for all eligible DHCP agents and restart them to activate the values.

Case 2b: Queries are forwarded to DNS resolver(s) configured on the host

In this case, the DHCP agent forwards queries from the instances to the DNS resolver(s) configured in the `resolv.conf` file on the host running the DHCP agent. This requires these resolvers being reachable from all virtual networks.

In the `dhcp_agent.ini` file, enable using the DNS resolver(s) configured on the host.

```
[DEFAULT]
dnsmasq_local_resolv = True
```

Note: You must configure this option for all eligible DHCP agents and restart them to activate this setting.

8.2.11 Distributed Virtual Routing with VRRP

Open vSwitch: High availability using DVR supports augmentation using Virtual Router Redundancy Protocol (VRRP). Using this configuration, virtual routers support both the `--distributed` and `--ha` options.

Similar to legacy HA routers, DVR/SNAT HA routers provide a quick fail over of the SNAT service to a backup DVR/SNAT router on an l3-agent running on a different node.

SNAT high availability is implemented in a manner similar to the *Linux bridge: High availability using VRRP* and *Open vSwitch: High availability using VRRP* examples where `keepalived` uses VRRP to provide quick failover of SNAT services.

During normal operation, the primary router periodically transmits *heartbeat* packets over a hidden project network that connects all HA routers for a particular project.

If the DVR/SNAT backup router stops receiving these packets, it assumes failure of the primary DVR/SNAT router and promotes itself to primary router by configuring IP addresses on the interfaces in the `snat` namespace. In environments with more than one backup router, the rules of VRRP are followed to select a new primary router.

Warning: There is a known bug with `keepalived` v1.2.15 and earlier which can cause packet loss when `max_l3_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to `keepalived` v1.2.16 or greater when using this feature.

Configuration example

The basic deployment model consists of one controller node, two or more network nodes, and multiple computes nodes.

Controller node configuration

1. Add the following to `/etc/neutron/neutron.conf`:

```
[DEFAULT]
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
router_distributed = True
l3_ha = True
l3_ha_net_cidr = 169.254.192.0/18
max_l3_agents_per_router = 3
```

   When the `router_distributed = True` flag is configured, routers created by all users are distributed. Without it, only privileged users can create distributed routers by using `--distributed True`.

   Similarly, when the `l3_ha = True` flag is configured, routers created by all users default to HA.

   It follows that with these two flags set to `True` in the configuration file, routers created by all users will default to distributed HA routers (DVR HA).

   The same can explicitly be accomplished by a user with administrative credentials setting the flags in the **`openstack router create`** command:

```
$ openstack router create name-of-router --distributed --ha
```

   Note: The *max_l3_agents_per_router* determine the number of backup DVR/SNAT routers which will be instantiated.

2. Add the following to `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ml2]
type_drivers = flat,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
extension_drivers = port_security

[ml2_type_flat]
flat_networks = external

[ml2_type_vxlan]
vni_ranges = MIN_VXLAN_ID:MAX_VXLAN_ID
```

   Replace `MIN_VXLAN_ID` and `MAX_VXLAN_ID` with VXLAN ID minimum and maximum values suitable for your environment.

Note: The first value in the `tenant_network_types` option becomes the default project network type when a regular user creates a network.

Network nodes

1. Configure the Open vSwitch agent. Add the following to `/etc/neutron/plugins/ml2/openvswitch_agent.ini`:

```
[ovs]
local_ip = TUNNEL_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
enable_distributed_routing = True
tunnel_types = vxlan
l2_population = True
```

   Replace `TUNNEL_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN project networks.

2. Configure the L3 agent. Add the following to `/etc/neutron/l3_agent.ini`:

```
[DEFAULT]
ha_vrrp_auth_password = password
interface_driver = openvswitch
agent_mode = dvr_snat
```

Compute nodes

1. Configure the Open vSwitch agent. Add the following to `/etc/neutron/plugins/ml2/openvswitch_agent.ini`:

```
[ovs]
local_ip = TUNNEL_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
enable_distributed_routing = True
tunnel_types = vxlan
l2_population = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
↪OVSHybridIptablesFirewallDriver
```

2. Configure the L3 agent. Add the following to `/etc/neutron/l3_agent.ini`:

```
[DEFAULT]
interface_driver = openvswitch
agent_mode = dvr
```

Replace TUNNEL_INTERFACE_IP_ADDRESS with the IP address of the interface that handles VXLAN project networks.

Keepalived VRRP health check

The health of your keepalived instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the l3_agent.ini file:

```
ha_vrrp_health_check_interval = 30
```

Where ha_vrrp_health_check_interval indicates how often in seconds the health check should run. The default value is 0, which indicates that the check should not run at all.

Known limitations

- Migrating a router from distributed only, HA only, or legacy to distributed HA is not supported at this time. The router must be created as distributed HA. The reverse direction is also not supported. You cannot reconfigure a distributed HA router to be only distributed, only HA, or legacy.
- There are certain scenarios where l2pop and distributed HA routers do not interact in an expected manner. These situations are the same that affect HA only routers and l2pop.

8.2.12 Floating IP port forwarding

Floating IP port forwarding enables users to forward traffic from a TCP/UDP/other protocol port of a floating IP to a TCP/UDP/other protocol port associated to one of the fixed IPs of a Neutron port. This is accomplished by associating port_forwarding sub-resource to a floating IP.

CRUD operations for port forwarding are implemented by a Neutron API extension and a service plug-in. Please refer to the Neutron API Reference documentation for details on the CRUD operations.

Configuring floating IP port forwarding

To configure floating IP port forwarding, take the following steps:

- Add the port_forwarding service to the service_plugins setting in /etc/neutron/neutron.conf. For example:

  ```
  service_plugins = router,segments,port_forwarding
  ```

- Set the extensions option in the [agent] section of /etc/neutron/l3_agent.ini to include port_forwarding. This has to be done in each network and compute node where the L3 agent is running. For example:

  ```
  extensions = port_forwarding
  ```

Note: The `router` service plug-in manages floating IPs and routers. As a consequence, it has to be configured along with the `port_forwarding` service plug-in.

Note: After updating the options in the configuration files, the neutron-server and every neutron-l3-agent need to be restarted for the new values to take effect.

After configuring floating IP port forwarding, the `floating-ip-port-forwarding` extension alias will be included in the output of the following command:

```
$ openstack extension list --network
```

8.2.13 IPAM configuration

Starting with the Liberty release, OpenStack Networking includes a pluggable interface for the IP Address Management (IPAM) function. This interface creates a driver framework for the allocation and de-allocation of subnets and IP addresses, enabling the integration of alternate IPAM implementations or third-party IP Address Management systems.

The basics

In Liberty and Mitaka, the IPAM implementation within OpenStack Networking provided a pluggable and non-pluggable flavor. As of Newton, the non-pluggable flavor is no longer available. Instead, it is completely replaced with a reference driver implementation of the pluggable framework. All data will be automatically migrated during the upgrade process, unless you have previously configured a pluggable IPAM driver. In that case, no migration is necessary.

To configure a driver other than the reference driver, specify it in the `neutron.conf` file. Do this after the migration is complete. There is no need to specify any value if you wish to use the reference driver.

```
ipam_driver = ipam-driver-name
```

There is no need to specify any value if you wish to use the reference driver, though specifying `internal` will explicitly choose the reference driver. The documentation for any alternate drivers will include the value to use when specifying that driver.

Known limitations

- The driver interface is designed to allow separate drivers for each subnet pool. However, the current implementation allows only a single IPAM driver system-wide.
- Third-party drivers must provide their own migration mechanisms to convert existing OpenStack installations to their IPAM.

8.2.14 IPv6

This section describes the OpenStack Networking reference implementation for IPv6, including the following items:

- How to enable dual-stack (IPv4 and IPv6 enabled) instances.
- How those instances receive an IPv6 address.
- How those instances communicate across a router to other subnets or the internet.
- How those instances interact with other OpenStack services.

Enabling a dual-stack network in OpenStack Networking simply requires creating a subnet with the `ip_version` field set to `6`, then the IPv6 attributes (`ipv6_ra_mode` and `ipv6_address_mode`) set. The `ipv6_ra_mode` and `ipv6_address_mode` will be described in detail in the next section. Finally, the subnets `cidr` needs to be provided.

This section does not include the following items:

- Single stack IPv6 project networking
- OpenStack control communication between servers and services over an IPv6 network.
- Connection to the OpenStack APIs via an IPv6 transport network
- IPv6 multicast
- IPv6 support in conjunction with any out of tree routers, switches, services or agents whether in physical or virtual form factors.

Neutron subnets and the IPv6 API attributes

As of Juno, the OpenStack Networking service (neutron) provides two new attributes to the subnet object, which allows users of the API to configure IPv6 subnets.

There are two IPv6 attributes:

- `ipv6_ra_mode`
- `ipv6_address_mode`

These attributes can be set to the following values:

- `slaac`
- `dhcpv6-stateful`
- `dhcpv6-stateless`

The attributes can also be left unset.

IPv6 addressing

The `ipv6_address_mode` attribute is used to control how addressing is handled by OpenStack. There are a number of different ways that guest instances can obtain an IPv6 address, and this attribute exposes these choices to users of the Networking API.

Router advertisements

The `ipv6_ra_mode` attribute is used to control router advertisements for a subnet.

The IPv6 Protocol uses Internet Control Message Protocol packets (ICMPv6) as a way to distribute information about networking. ICMPv6 packets with the type flag set to 134 are called Router Advertisement messages, which contain information about the router and the route that can be used by guest instances to send network traffic.

The `ipv6_ra_mode` is used to specify if the Networking service should generate Router Advertisement messages for a subnet.

ipv6_ra_mode and ipv6_address_mode combinations

ipv6 ra mode	ipv6 address mode	radvd A,M,O	External Router A,M,O	Description
N/S	*N/S*	Off	Not Defined	Backwards compatibility with pre-Juno IPv6 behavior.
N/S	slaac	Off	1,0,0	Guest instance obtains IPv6 address from non-OpenStack router using SLAAC.
N/S	dhcpv6-stateful	Off	0,1,1	Not currently implemented in the reference implementation.
N/S	dhcpv6-stateless	Off	1,0,1	Not currently implemented in the reference implementation.
slaac	*N/S*	1,0,0	Off	Not currently implemented in the reference implementation.
dhcpv6-stateful	*N/S*	0,1,1	Off	Not currently implemented in the reference implementation.
dhcpv6-stateless	*N/S*	1,0,1	Off	Not currently implemented in the reference implementation.
slaac	slaac	1,0,0	Off	Guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC.
dhcpv6-stateful	dhcpv6-stateful	0,1,1	Off	Guest instance obtains IPv6 address from dnsmasq using DHCPv6 stateful and optional info from dnsmasq using DHCPv6.
dhcpv6-stateless	dhcpv6-stateless	1,0,1	Off	Guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6.
slaac	dhcpv6-stateful			*Invalid combination.*
slaac	dhcpv6-stateless			*Invalid combination.*
dhcpv6-stateful	slaac			*Invalid combination.*
dhcpv6-stateful	dhcpv6-stateless			*Invalid combination.*
dhcpv6-stateless	slaac			*Invalid combination.*
dhcpv6-stateless	dhcpv6-stateful			*Invalid combination.*

Project network considerations

Dataplane

Both the Linux bridge and the Open vSwitch dataplane modules support forwarding IPv6 packets amongst the guests and router ports. Similar to IPv4, there is no special configuration or setup required to enable the dataplane to properly forward packets from the source to the destination using IPv6. Note that these dataplanes will forward Link-local Address (LLA) packets between hosts on the same network just fine without any participation or setup by OpenStack components after the ports are all connected and MAC addresses learned.

Addresses for subnets

There are three methods currently implemented for a subnet to get its `cidr` in OpenStack:

1. Direct assignment during subnet creation via command line or Horizon
2. Referencing a subnet pool during subnet creation
3. Using a Prefix Delegation (PD) client to request a prefix for a subnet from a PD server

In the future, additional techniques could be used to allocate subnets to projects, for example, use of an external IPAM module.

Address modes for ports

Note: An external DHCPv6 server in theory could override the full address OpenStack assigns based on the EUI-64 address, but that would not be wise as it would not be consistent through the system.

IPv6 supports three different addressing schemes for address configuration and for providing optional network information.

Stateless Address Auto Configuration (SLAAC) Address configuration using Router Advertisements.

DHCPv6-stateless Address configuration using Router Advertisements and optional information using DHCPv6.

DHCPv6-stateful Address configuration and optional information using DHCPv6.

OpenStack can be setup such that OpenStack Networking directly provides Router Advertisements, DHCP relay and DHCPv6 address and optional information for their networks or this can be delegated to external routers and services based on the drivers that are in use. There are two neutron subnet attributes - `ipv6_ra_mode` and `ipv6_address_mode` that determine how IPv6 addressing and network information is provided to project instances:

- `ipv6_ra_mode`: Determines who sends Router Advertisements.
- `ipv6_address_mode`: Determines how instances obtain IPv6 address, default gateway, or optional information.

For the above two attributes to be effective, `enable_dhcp` of the subnet object must be set to True.

Using SLAAC for addressing

When using SLAAC, the currently supported combinations for `ipv6_ra_mode` and `ipv6_address_mode` are as follows.

ipv6_ra_mode	ipv6_address_mode	Result
Not specified.	SLAAC	Addresses are assigned using EUI-64, and an external router will be used for routing.
SLAAC	SLAAC	Address are assigned using EUI-64, and OpenStack Networking provides routing.

Setting SLAAC for `ipv6_ra_mode` configures the neutron router with an radvd agent to send Router Advertisements. The list below captures the values set for the address configuration flags in the Router Advertisement messages in this scenario.

- Auto Configuration Flag = 1
- Managed Configuration Flag = 0
- Other Configuration Flag = 0

New or existing neutron networks that contain a SLAAC enabled IPv6 subnet will result in all neutron ports attached to the network receiving IPv6 addresses. This is because when Router Advertisement messages are multicast on a neutron network, they are received by all IPv6 capable ports on the network, and each port will then configure an IPv6 address based on the information contained in the Router Advertisement messages. In some cases, an IPv6 SLAAC address will be added to a port, in addition to other IPv4 and IPv6 addresses that the port already has been assigned.

Note: If a router is not created and added to the subnet, SLAAC addressing will not succeed for instances since no Router Advertisement messages will be generated.

DHCPv6

For DHCPv6, the currently supported combinations are as follows:

ipv6_ra_mode	ipv6_address_mode	Result
DHCPv6-stateless	DHCPv6-stateless	Addresses are assigned through Router Advertisements (see SLAAC above) and optional information is delivered through DHCPv6.
DHCPv6-stateful	DHCPv6-stateful	Addresses and optional information are assigned using DHCPv6.

Setting DHCPv6-stateless for `ipv6_ra_mode` configures the neutron router with an radvd agent to send Router Advertisements. The list below captures the values set for the address configuration flags in the Router Advertisement messages in this scenario. Similarly, setting DHCPv6-stateless for `ipv6_address_mode` configures neutron DHCP implementation to provide the additional network information.

- Auto Configuration Flag = 1
- Managed Configuration Flag = 0

- Other Configuration Flag = 1

Setting DHCPv6-stateful for `ipv6_ra_mode` configures the neutron router with an radvd agent to send Router Advertisements. The list below captures the values set for the address configuration flags in the Router Advertisements messages in this scenario. Similarly, setting DHCPv6-stateful for `ipv6_address_mode` configures neutron DHCP implementation to provide addresses and additional network information through DHCPv6.

- Auto Configuration Flag = 0
- Managed Configuration Flag = 1
- Other Configuration Flag = 1

Note: If a router is not created and added to the subnet, DHCPv6 addressing will not succeed for instances since no Router Advertisement messages will be generated.

Router support

The behavior of the neutron router for IPv6 is different than for IPv4 in a few ways.

Internal router ports, that act as default gateway ports for a network, will share a common port for all IPv6 subnets associated with the network. This implies that there will be an IPv6 internal router interface with multiple IPv6 addresses from each of the IPv6 subnets associated with the network and a separate IPv4 internal router interface for the IPv4 subnet. On the other hand, external router ports are allowed to have a dual-stack configuration with both an IPv4 and an IPv6 address assigned to them.

Neutron project networks that are assigned Global Unicast Address (GUA) prefixes and addresses dont require NAT on the neutron router external gateway port to access the outside world. As a consequence of the lack of NAT the external router port doesnt require a GUA to send and receive to the external networks. This implies a GUA IPv6 subnet prefix is not necessarily needed for the neutron external network. By default, a IPv6 LLA associated with the external gateway port can be used for routing purposes. To handle this scenario, the implementation of router-gateway-set API in neutron has been modified so that an IPv6 subnet is not required for the external network that is associated with the neutron router. The LLA address of the upstream router can be learned in two ways.

1. In the absence of an upstream Router Advertisement message, the `ipv6_gateway` flag can be set with the external router gateway LLA in the neutron L3 agent configuration file. This also requires that no subnet is associated with that port.
2. The upstream router can send a Router Advertisement and the neutron router will automatically learn the next-hop LLA, provided again that no subnet is assigned and the `ipv6_gateway` flag is not set.

Effectively the `ipv6_gateway` flag takes precedence over a Router Advertisements that is received from the upstream router. If it is desired to use a GUA next hop that is accomplished by allocating a subnet to the external router port and assigning the upstream routers GUA address as the gateway for the subnet.

Note: It should be possible for projects to communicate with each other on an isolated network (a network without a router port) using LLA with little to no participation on the part of OpenStack. The authors of this section have not proven that to be true for all scenarios.

Note: When using the neutron L3 agent in a configuration where it is auto-configuring an IPv6 address via SLAAC, and the agent is learning its default IPv6 route from the ICMPv6 Router Advertisement, it may be necessary to set the `net.ipv6.conf.<physical_interface>.accept_ra` sysctl to the value `2` in order for routing to function correctly. For a more detailed description, please see the bug.

Neutrons Distributed Router feature and IPv6

IPv6 does work when the Distributed Virtual Router functionality is enabled, but all ingress/egress traffic is via the centralized router (hence, not distributed). More work is required to fully enable this functionality.

Advanced services

VPNaaS

VPNaaS supports IPv6, but support in Kilo and prior releases will have some bugs that may limit how it can be used. More thorough and complete testing and bug fixing is being done as part of the Liberty release. IPv6-based VPN-as-a-Service is configured similar to the IPv4 configuration. Either or both the `peer_address` and the `peer_cidr` can specified as an IPv6 address. The choice of addressing modes and router modes described above should not impact support.

FWaaS

FWaaS allows creation of IPv6 based rules.

NAT & Floating IPs

At the current time OpenStack Networking does not provide any facility to support any flavor of NAT with IPv6. Unlike IPv4 there is no current embedded support for floating IPs with IPv6. It is assumed that the IPv6 addressing amongst the projects is using GUAs with no overlap across the projects.

Security considerations

For more information about security considerations, see the `Security groups` section in *OpenStack Networking*.

Configuring interfaces of the guest

OpenStack currently doesnt support the Privacy Extensions defined by RFC 4941, or the Opaque Identifier generation methods defined in RFC 7217. The interface identifier and DUID used must be directly derived from the MAC address as described in RFC 2373. The compute instances must not be set up to utilize either of these methods when generating their interface identifier, or they might not be able to communicate properly on the network. For example, in Linux guests, these are controlled via these two `sysctl` variables:

- `net.ipv6.conf.*.use_tempaddr` (Privacy Extensions)

This allows the use of non-changing interface identifiers for IPv6 addresses according to RFC3041 semantics. It should be disabled (zero) so that stateless addresses are constructed using a stable, EUI64-based value.

- `net.ipv6.conf.*.addr_gen_mode`

This defines how link-local and auto-configured IPv6 addresses are generated. It should be set to zero (default) so that IPv6 addresses are generated using an EUI64-based value.

Note: Support for `addr_gen_mode` was added in kernel version 4.11.

Other types of guests might have similar configuration options, please consult your distribution documentation for more information.

There are no provisions for an IPv6-based metadata service similar to what is provided for IPv4. In the case of dual-stacked guests though it is always possible to use the IPv4 metadata service instead. IPv6-only guests will have to use another method for metadata injection such as using a configuration drive, which is described in the Nova documentation on config-drive.

Unlike IPv4, the MTU of a given network can be conveyed in both the Router Advertisement messages sent by the router, as well as in DHCP messages.

OpenStack control & management network considerations

As of the Kilo release, considerable effort has gone in to ensuring the project network can handle dual stack IPv6 and IPv4 transport across the variety of configurations described above. OpenStack control network can be run in a dual stack configuration and OpenStack API endpoints can be accessed via an IPv6 network. At this time, Open vSwitch (OVS) tunnel types - STT, VXLAN, GRE, support both IPv4 and IPv6 endpoints.

Prefix delegation

From the Liberty release onwards, OpenStack Networking supports IPv6 prefix delegation. This section describes the configuration and workflow steps necessary to use IPv6 prefix delegation to provide automatic allocation of subnet CIDRs. This allows you as the OpenStack administrator to rely on an external (to the OpenStack Networking service) DHCPv6 server to manage your project network prefixes.

Note: Prefix delegation became available in the Liberty release, it is not available in the Kilo release. HA and DVR routers are not currently supported by this feature.

Configuring OpenStack Networking for prefix delegation

To enable prefix delegation, edit the /etc/neutron/neutron.conf file.

```
ipv6_pd_enabled = True
```

Note: If you are not using the default dibbler-based driver for prefix delegation, then you also need to set the driver in /etc/neutron/neutron.conf:

```
pd_dhcp_driver = <class path to driver>
```

Drivers other than the default one may require extra configuration.

This tells OpenStack Networking to use the prefix delegation mechanism for subnet allocation when the user does not provide a CIDR or subnet pool id when creating a subnet.

Requirements

To use this feature, you need a prefix delegation capable DHCPv6 server that is reachable from your OpenStack Networking node(s). This could be software running on the OpenStack Networking node(s) or elsewhere, or a physical router. For the purposes of this guide we are using the open-source DHCPv6 server, Dibbler. Dibbler is available in many Linux package managers, or from source at tomaszmrugalski/dibbler.

When using the reference implementation of the OpenStack Networking prefix delegation driver, Dibbler must also be installed on your OpenStack Networking node(s) to serve as a DHCPv6 client. Version 1.0.1 or higher is required.

This guide assumes that you are running a Dibbler server on the network node where the external network bridge exists. If you already have a prefix delegation capable DHCPv6 server in place, then you can skip the following section.

Configuring the Dibbler server

After installing Dibbler, edit the /etc/dibbler/server.conf file:

```
script "/var/lib/dibbler/pd-server.sh"

iface "br-ex" {
    pd-class {
        pd-pool 2001:db8:2222::/48
        pd-length 64
    }
}
```

The options used in the configuration file above are:

- script Points to a script to be run when a prefix is delegated or released. This is only needed if you want instances on your subnets to have external network access. More on this below.
- iface The name of the network interface on which to listen for prefix delegation messages.

- `pd-pool` The larger prefix from which you want your delegated prefixes to come. The example given is sufficient if you do not need external network access, otherwise a unique globally routable prefix is necessary.
- `pd-length` The length that delegated prefixes will be. This must be 64 to work with the current OpenStack Networking reference implementation.

To provide external network access to your instances, your Dibbler server also needs to create new routes for each delegated prefix. This is done using the script file named in the config file above. Edit the `/var/lib/dibbler/pd-server.sh` file:

```
if [ "$PREFIX1" != "" ]; then
    if [ "$1" == "add" ]; then
        sudo ip -6 route add ${PREFIX1}/64 via $REMOTE_ADDR dev $IFACE
    fi
    if [ "$1" == "delete" ]; then
        sudo ip -6 route del ${PREFIX1}/64 via $REMOTE_ADDR dev $IFACE
    fi
fi
```

The variables used in the script file above are:

- `$PREFIX1` The prefix being added/deleted by the Dibbler server.
- `$1` The operation being performed.
- `$REMOTE_ADDR` The IP address of the requesting Dibbler client.
- `$IFACE` The network interface upon which the request was received.

The above is all you need in this scenario, but more information on installing, configuring, and running Dibbler is available in the Dibbler user guide, at Dibbler a portable DHCPv6.

To start your Dibbler server, run:

```
# dibbler-server run
```

Or to run in headless mode:

```
# dibbler-server start
```

When using DevStack, it is important to start your server after the `stack.sh` script has finished to ensure that the required network interfaces have been created.

User workflow

First, create a network and IPv6 subnet:

```
$ openstack network create ipv6-pd
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-01-25T19:26:01Z                 |
| description               |                                      |
```

(continues on next page)

(continued from previous page)

```
| headers                   |                                      |
| id                        | 4b782725-6abe-4a2d-b061-763def1bb029 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | ipv6-pd                              |
| port_security_enabled     | True                                 |
| project_id                | 61b7eba037fd41f29cfba757c010faff     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 46                                   |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2017-01-25T19:26:01Z                 |
+---------------------------+--------------------------------------+

$ openstack subnet create --ip-version 6 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --use-default-subnet-pool \
--network ipv6-pd ipv6-pd-1
+------------------------+--------------------------------------+
| Field                  | Value                                |
+------------------------+--------------------------------------+
| allocation_pools       | ::2-::ffff:ffff:ffff:ffff            |
| cidr                   | ::/64                                |
| created_at             | 2017-01-25T19:31:53Z                 |
| description            |                                      |
| dns_nameservers        |                                      |
| enable_dhcp            | True                                 |
| gateway_ip             | ::1                                  |
| headers                |                                      |
| host_routes            |                                      |
| id                     | 1319510d-c92c-4532-bf5d-8bcf3da761a1 |
| ip_version             | 6                                    |
| ipv6_address_mode      | slaac                                |
| ipv6_ra_mode           | slaac                                |
| name                   | ipv6-pd-1                            |
| network_id             | 4b782725-6abe-4a2d-b061-763def1bb029 |
| project_id             | 61b7eba037fd41f29cfba757c010faff     |
| revision_number        | 2                                    |
| service_types          |                                      |
| subnetpool_id          | prefix_delegation                    |
| tags                   | []                                   |
| updated_at             | 2017-01-25T19:31:53Z                 |
| use_default_subnetpool | True                                 |
+------------------------+--------------------------------------+
```

The subnet is initially created with a temporary CIDR before one can be assigned by prefix delegation. Any number of subnets with this temporary CIDR can exist without raising an overlap error. The subnetpool_id is automatically set to `prefix_delegation`.

To trigger the prefix delegation process, create a router interface between this subnet and a router with an active interface on the external network:

```
$ openstack router add subnet router1 ipv6-pd-1
```

The prefix delegation mechanism then sends a request via the external network to your prefix delegation server, which replies with the delegated prefix. The subnet is then updated with the new prefix, including issuing new IP addresses to all ports:

```
$ openstack subnet show ipv6-pd-1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 2001:db8:2222:6977::2-2001:db8:2222: |
|                   | 6977:ffff:ffff:ffff:ffff             |
| cidr              | 2001:db8:2222:6977::/64              |
| created_at        | 2017-01-25T19:31:53Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 2001:db8:2222:6977::1                |
| host_routes       |                                      |
| id                | 1319510d-c92c-4532-bf5d-8bcf3da761a1 |
| ip_version        | 6                                    |
| ipv6_address_mode | slaac                                |
| ipv6_ra_mode      | slaac                                |
| name              | ipv6-pd-1                            |
| network_id        | 4b782725-6abe-4a2d-b061-763def1bb029 |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| revision_number   | 4                                    |
| service_types     |                                      |
| subnetpool_id     | prefix_delegation                    |
| tags              | []                                   |
| updated_at        | 2017-01-25T19:35:26Z                 |
+-------------------+--------------------------------------+
```

If the prefix delegation server is configured to delegate globally routable prefixes and setup routes, then any instance with a port on this subnet should now have external network access.

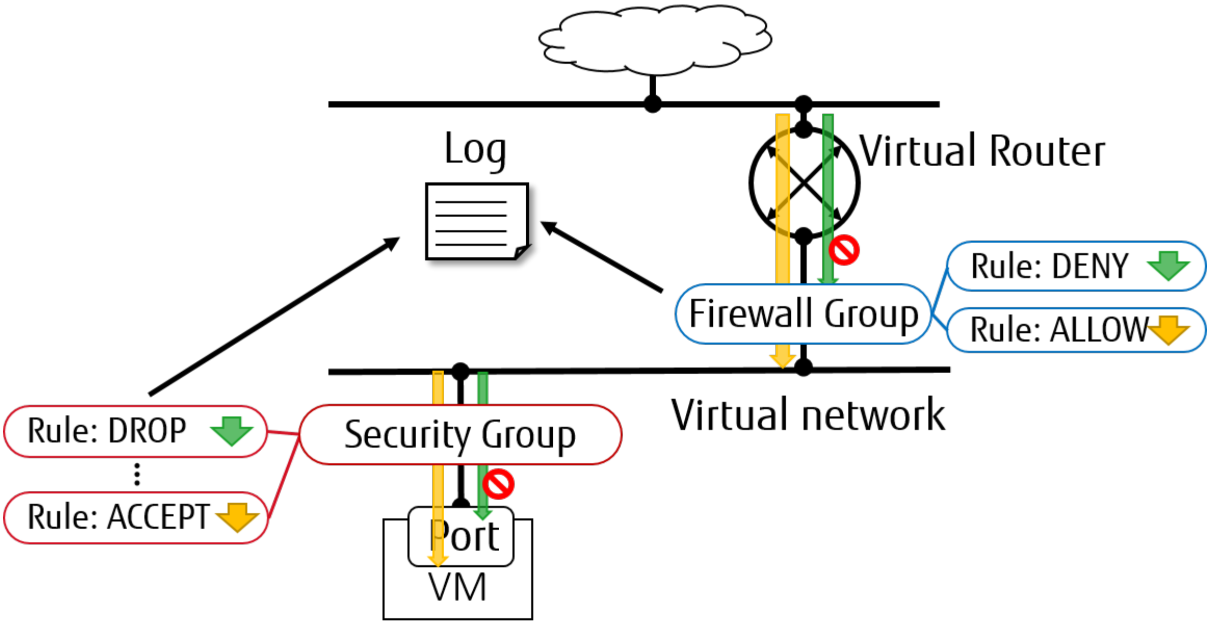
Deleting the router interface causes the subnet to be reverted to the temporary CIDR, and all ports have their IPs updated. Prefix leases are released and renewed automatically as necessary.

References

The following presentation from the Barcelona Summit provides a great guide for setting up IPv6 with OpenStack: Deploying IPv6 in OpenStack Environments.

8.2.15 Neutron Packet Logging Framework

Packet logging service is designed as a Neutron plug-in that captures network packets for relevant resources (e.g. security group or firewall group) when the registered events occur.



Supported loggable resource types

From Rocky release, both of `security_group` and `firewall_group` are supported as resource types in Neutron packet logging framework.

Service Configuration

To enable the logging service, follow the below steps.

1. On Neutron controller node, add `log` to `service_plugins` setting in `/etc/neutron/neutron.conf` file. For example:

```
service_plugins = router,metering,log
```

2. To enable logging service for `security_group` in Layer 2, add `log` to option `extensions` in section `[agent]` in `/etc/neutron/plugins/ml2/ml2_conf.ini` for controller node and in `/etc/neutron/plugins/ml2/openvswitch_agent.ini` for compute/network nodes. For example:

```
[agent]
extensions = log
```

Note: Fwaas v2 log is currently only supported by openvswitch, the firewall logging driver of linuxbridge is not implemented.

3. To enable logging service for `firewall_group` in Layer 3, add `fwaas_v2_log` to option `extensions` in section `[AGENT]` in `/etc/neutron/l3_agent.ini` for network nodes. For example:

```
[AGENT]
extensions = fwaas_v2,fwaas_v2_log
```

4. On compute/network nodes, add configuration for logging service to `[network_log]` in `/etc/neutron/plugins/ml2/openvswitch_agent.ini` and in `/etc/neutron/l3_agent.ini` as shown bellow:

```
[network_log]
rate_limit = 100
burst_limit = 25
#local_output_log_base = <None>
```

In which, `rate_limit` is used to configure the maximum number of packets to be logged per second (packets per second). When a high rate triggers `rate_limit`, logging queues packets to be logged. `burst_limit` is used to configure the maximum of queued packets. And logged packets can be stored anywhere by using `local_output_log_base`.

Note:

- It requires at least `100` for `rate_limit` and at least `25` for `burst_limit`.
- If `rate_limit` is unset, logging will log unlimited.
- If we dont specify `local_output_log_base`, logged packets will be stored in system journal like `/var/log/syslog` by default.

Trusted projects policy.json configuration

With the default `/etc/neutron/policy.json`, administrators must set up resource logging on behalf of the cloud projects.

If projects are trusted to administer their own loggable resources in their cloud, neutrons policy file `policy.json` can be modified to allow this.

Modify `/etc/neutron/policy.json` entries as follows:

```
"get_loggable_resources": "rule:regular_user",
"create_log": "rule:regular_user",
"get_log": "rule:regular_user",
"get_logs": "rule:regular_user",
"update_log": "rule:regular_user",
"delete_log": "rule:regular_user",
```

Service workflow for Operator

1. To check the loggable resources that are supported by framework:

```
$ openstack network loggable resources list
+-----------------+
| Supported types |
+-----------------+
| security_group  |
| firewall_group  |
+-----------------+
```

Note:

- In VM ports, logging for `security_group` in currently works with `openvswitch` firewall driver only. `linuxbridge` is under development.
- Logging for `firewall_group` works on internal router ports only. VM ports would be supported in the future.

---

2. Log creation:

   - Create a logging resource with an appropriate resource type

```
$ openstack network log create --resource-type security_
↪group \
  --description "Collecting all security events" \
  --event ALL Log_Created
+-----------------+----------------------------------------
↪-------+
| Field           | Value                                  ␣
↪       |
+-----------------+----------------------------------------
↪-------+
| Description     | Collecting all security events         ␣
↪       |
| Enabled         | True                                   ␣
↪       |
| Event           | ALL                                    ␣
↪       |
| ID              | 8085c3e6-0fa2-4954-b5ce-ff6207931b6d   ␣
↪       |
| Name            | Log_Created                            ␣
↪       |
| Project         | 02568bd62b414221956f15dbe9527d16       ␣
↪       |
| Resource        | None                                   ␣
↪       |
| Target          | None                                   ␣
↪       |
| Type            | security_group                         ␣
↪       |
| created_at      | 2017-07-05T02:56:43Z                   ␣
↪       |
| revision_number | 0                                      ␣
↪       |
```

(continues on next page)

(continued from previous page)

```
| tenant_id         | 02568bd62b414221956f15dbe9527d16                    ␣
↪         |
| updated_at        | 2017-07-05T02:56:43Z                                ␣
↪         |
+-------------------+----------------------------------------------------
↪--------+
```

Warning: In the case of `--resource` and `--target` are not specified from the request, these arguments will be assigned to `ALL` by default. Hence, there is an enormous range of log events will be created.

- Create logging resource with a given resource (sg1 or fwg1)

```
$ openstack network log create my-log --resource-type␣
↪security_group --resource sg1
$ openstack network log create my-log --resource-type␣
↪firewall_group --resource fwg1
```

- Create logging resource with a given target (portA)

```
$ openstack network log create my-log --resource-type␣
↪security_group --target portA
```

- Create logging resource for only the given target (portB) and the given resource (sg1 or fwg1)

```
$ openstack network log create my-log --resource-type␣
↪security_group --target portB --resource sg1
$ openstack network log create my-log --resource-type␣
↪firewall_group --target portB --resource fwg1
```

Note:

- The `Enabled` field is set to `True` by default. If enabled, logged events are written to the destination if `local_output_log_base` is configured or `/var/log/syslog` in default.
- The `Event` field will be set to `ALL` if `--event` is not specified from log creation request.

3. Enable/Disable log

We can `enable` or `disable` logging objects at runtime. It means that it will apply to all registered ports with the logging object immediately. For example:

```
$ openstack network log set --disable Log_Created
$ openstack network log show Log_Created
+-----------------+------------------------------------------------+
| Field           | Value                                          |
+-----------------+------------------------------------------------+
| Description     | Collecting all security events                 |
| Enabled         | False                                          |
| Event           | ALL                                            |
```

(continues on next page)

(continued from previous page)

```
| ID              | 8085c3e6-0fa2-4954-b5ce-ff6207931b6d |
| Name            | Log_Created                          |
| Project         | 02568bd62b414221956f15dbe9527d16     |
| Resource        | None                                 |
| Target          | None                                 |
| Type            | security_group                       |
| created_at      | 2017-07-05T02:56:43Z                 |
| revision_number | 1                                    |
| tenant_id       | 02568bd62b414221956f15dbe9527d16     |
| updated_at      | 2017-07-05T03:12:01Z                 |
+-----------------+--------------------------------------+
```

Logged events description

Currently, packet logging framework supports to collect ACCEPT or DROP or both events related to registered resources. As mentioned above, Neutron packet logging framework offers two loggable resources through the log service plug-in: security_group and firewall_group.

The general characteristics of each event will be shown as the following:

- Log every DROP event: Every DROP security events will be generated when an incoming or outgoing session is blocked by the security groups or firewall groups
- Log an ACCEPT event: The ACCEPT security event will be generated only for each NEW incoming or outgoing session that is allowed by security groups or firewall groups. More details for the ACCEPT events are shown as bellow:
  - North/South ACCEPT: For a North/South session there would be a single ACCEPT event irrespective of direction.
  - East/West ACCEPT/ACCEPT: In an intra-project East/West session where the originating port allows the session and the destination port allows the session, i.e. the traffic is allowed, there would be two ACCEPT security events generated, one from the perspective of the originating port and one from the perspective of the destination port.
  - East/West ACCEPT/DROP: In an intra-project East/West session initiation where the originating port allows the session and the destination port does not allow the session there would be ACCEPT security events generated from the perspective of the originating port and DROP security events generated from the perspective of the destination port.

1. The security events that are collected by security group should include:
   - A timestamp of the flow.
   - A status of the flow ACCEPT/DROP.
   - An indication of the originator of the flow, e.g which project or log resource generated the events.
   - An identifier of the associated instance interface (neutron port id).
   - A layer 2, 3 and 4 information (mac, address, port, protocol, etc).
   - Security event record format:
     - Logged data of an ACCEPT event would look like:

```
May 5 09:05:07 action=ACCEPT project_
↪id=736672c700cd43e1bd321aeaf940365c
log_resource_ids=['4522efdf-8d44-4e19-b237-64cafc49469b',
↪'42332d89-df42-4588-a2bb-3ce50829ac51']
vm_port=e0259ade-86de-482e-a717-f58258f7173f
ethernet(dst='fa:16:3e:ec:36:32',ethertype=2048,src=
↪'fa:16:3e:50:aa:b5'),
ipv4(csum=62071,dst='10.0.0.4',flags=2,header_length=5,
↪identification=36638,offset=0,
option=None,proto=6,src='172.24.4.10',tos=0,total_length=60,
↪ttl=63,version=4),
tcp(ack=0,bits=2,csum=15097,dst_port=80,offset=10,
↪option=[TCPOptionMaximumSegmentSize(kind=2,length=4,max_seg_
↪size=1460),
TCPOptionSACKPermitted(kind=4,length=2),␣
↪TCPOptionTimestamps(kind=8,length=10,ts_ecr=0,ts_val=196418896),
TCPOptionNoOperation(kind=1,length=1),␣
↪TCPOptionWindowScale(kind=3,length=3,shift_cnt=3)],
seq=3284890090,src_port=47825,urgent=0,window_size=14600)
```

– Logged data of a DROP event:

```
May 5 09:05:07 action=DROP project_
↪id=736672c700cd43e1bd321aeaf940365c
log_resource_ids=['4522efdf-8d44-4e19-b237-64cafc49469b'] vm_
↪port=e0259ade-86de-482e-a717-f58258f7173f
ethernet(dst='fa:16:3e:ec:36:32',ethertype=2048,src=
↪'fa:16:3e:50:aa:b5'),
ipv4(csum=62071,dst='10.0.0.4',flags=2,header_length=5,
↪identification=36638,offset=0,
option=None,proto=6,src='172.24.4.10',tos=0,total_length=60,
↪ttl=63,version=4),
tcp(ack=0,bits=2,csum=15097,dst_port=80,offset=10,
↪option=[TCPOptionMaximumSegmentSize(kind=2,length=4,max_seg_
↪size=1460),
TCPOptionSACKPermitted(kind=4,length=2),␣
↪TCPOptionTimestamps(kind=8,length=10,ts_ecr=0,ts_val=196418896),
TCPOptionNoOperation(kind=1,length=1),␣
↪TCPOptionWindowScale(kind=3,length=3,shift_cnt=3)],
seq=3284890090,src_port=47825,urgent=0,window_size=14600)
```

2. The events that are collected by firewall group should include:
   - A timestamp of the flow.
   - A status of the flow ACCEPT/DROP.
   - The identifier of log objects that are collecting this event
   - An identifier of the associated instance interface (neutron port id).
   - A layer 2, 3 and 4 information (mac, address, port, protocol, etc).
   - Security event record format:
     – Logged data of an ACCEPT event would look like:

```
Jul 26 14:46:20:
action=ACCEPT, log_resource_ids=[u'2e030f3a-e93d-4a76-bc60-
↪1d11c0f6561b'], port=9882c485-b808-4a34-a3fb-b537642c66b2
pkt=ethernet(dst='fa:16:3e:8f:47:c5',ethertype=2048,src=
↪'fa:16:3e:1b:3e:67')
ipv4(csum=47423,dst='10.10.1.16',flags=2,header_length=5,
↪identification=27969,offset=0,option=None,proto=1,src='10.10.0.5
↪',tos=0,total_length=84,ttl=63,version=4)
icmp(code=0,csum=41376,data=echo(data='\xe5\xf2\xfej\x00\x00\x00\
↪x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
↪x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00',id=29185,seq=0),type=8)
```

– Logged data of a `DROP` event:

```
Jul 26 14:51:20:
action=DROP, log_resource_ids=[u'2e030f3a-e93d-4a76-bc60-
↪1d11c0f6561b'], port=9882c485-b808-4a34-a3fb-b537642c66b2
pkt=ethernet(dst='fa:16:3e:32:7d:ff',ethertype=2048,src=
↪'fa:16:3e:28:83:51')
ipv4(csum=17518,dst='10.10.0.5',flags=2,header_length=5,
↪identification=57874,offset=0,option=None,proto=1,src='10.10.1.
↪16',tos=0,total_length=84,ttl=63,version=4)
icmp(code=0,csum=23772,data=echo(data='\x8a\xa0\xac|\x00\x00\x00\
↪x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
↪x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00',id=25601,seq=5),type=8)
```

Note: No other extraneous events are generated within the security event logs, e.g. no debugging data, etc.

8.2.16 Macvtap mechanism driver

The Macvtap mechanism driver for the ML2 plug-in generally increases network performance of instances.

Consider the following attributes of this mechanism driver to determine practicality in your environment:

- Supports only instance ports. Ports for DHCP and layer-3 (routing) services must use another mechanism driver such as Linux bridge or Open vSwitch (OVS).
- Supports only untagged (flat) and tagged (VLAN) networks.
- Lacks support for security groups including basic (sanity) and anti-spoofing rules.
- Lacks support for layer-3 high-availability mechanisms such as Virtual Router Redundancy Protocol (VRRP) and Distributed Virtual Routing (DVR).
- Only compute resources can be attached via macvtap. Attaching other resources like DHCP, Routers and others is not supported. Therefore run either OVS or linux bridge in VLAN or flat mode on the controller node.

- Instance migration requires the same values for the `physical_interface_mapping` configuration option on each compute node. For more information, see https://bugs.launchpad.net/neutron/+bug/1550400.

Prerequisites

You can add this mechanism driver to an existing environment using either the Linux bridge or OVS mechanism drivers with only provider networks or provider and self-service networks. You can change the configuration of existing compute nodes or add compute nodes with the Macvtap mechanism driver. The example configuration assumes addition of compute nodes with the Macvtap mechanism driver to the *Linux bridge: Self-service networks* or *Open vSwitch: Self-service networks* deployment examples.

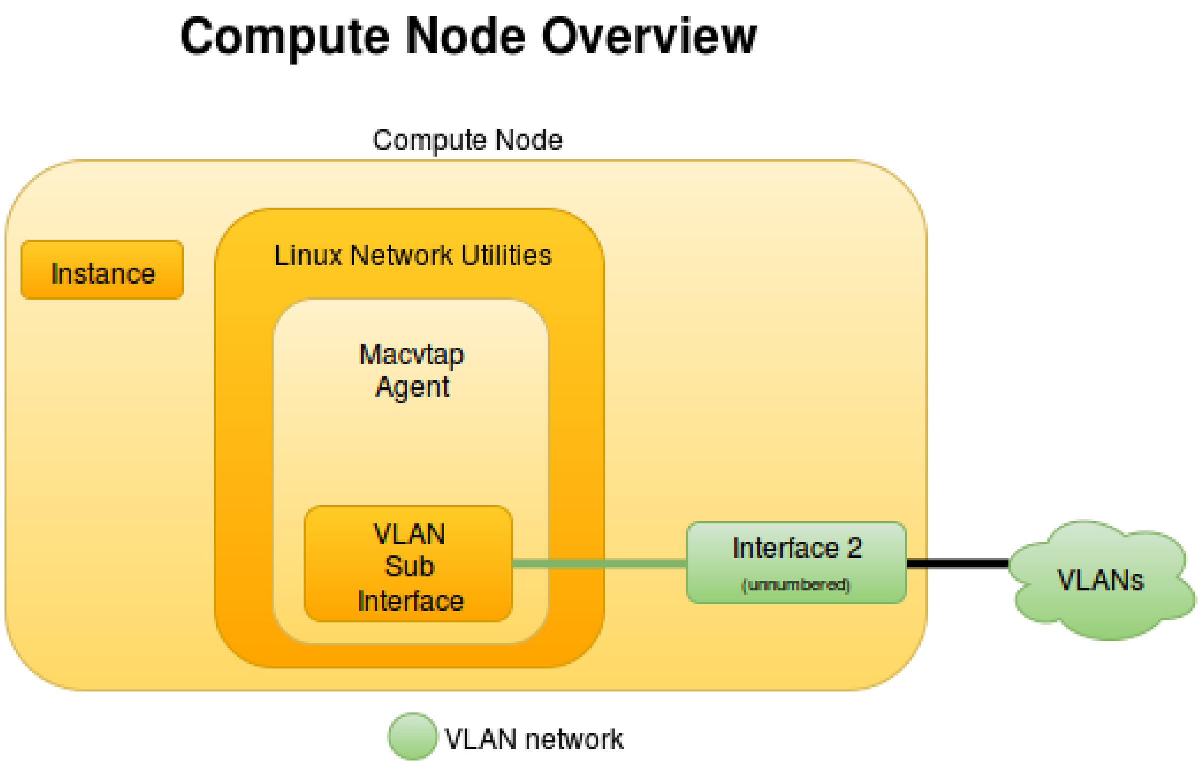
Add one or more compute nodes with the following components:

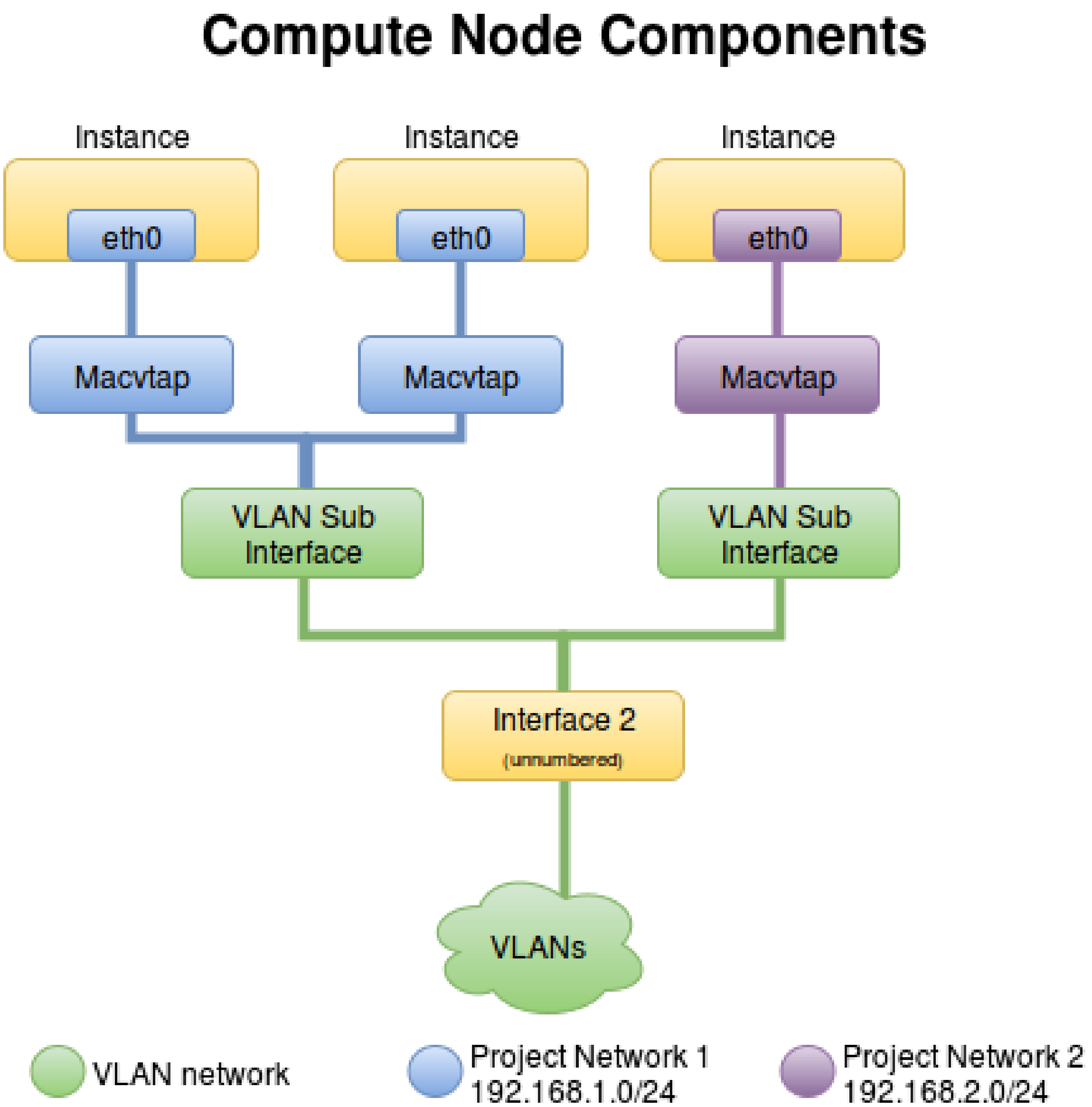
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking Macvtap layer-2 agent and any dependencies.

Note: To support integration with the deployment examples, this content configures the Macvtap mechanism driver to use the overlay network for untagged (flat) or tagged (VLAN) networks in addition to overlay networks such as VXLAN. Your physical network infrastructure must support VLAN (802.1q) tagging on the overlay network.

Architecture

The Macvtap mechanism driver only applies to compute nodes. Otherwise, the environment resembles the prerequisite deployment example.





Example configuration

Use the following example configuration as a template to add support for the Macvtap mechanism driver to an existing operational environment.

Controller node

1. In the `ml2_conf.ini` file:

   - Add `macvtap` to mechanism drivers.

   ```
   [ml2]
   mechanism_drivers = macvtap
   ```

   - Configure network mappings.

```
[ml2_type_flat]
flat_networks = provider,macvtap

[ml2_type_vlan]
network_vlan_ranges = provider,macvtap:VLAN_ID_START:VLAN_ID_END
```

Note: Use of `macvtap` is arbitrary. Only the self-service deployment examples require VLAN ID ranges. Replace `VLAN_ID_START` and `VLAN_ID_END` with appropriate numerical values.

2. Restart the following services:
   - Server

Network nodes

No changes.

Compute nodes

1. Install the Networking service Macvtap layer-2 agent.
2. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

   See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

3. In the `macvtap_agent.ini` file, configure the layer-2 agent.

```
[macvtap]
physical_interface_mappings = macvtap:MACVTAP_INTERFACE

[securitygroup]
firewall_driver = noop
```

Replace MACVTAP_INTERFACE with the name of the underlying interface that handles Macvtap mechanism driver interfaces. If using a prerequisite deployment example, replace MACVTAP_INTERFACE with the name of the underlying interface that handles overlay networks. For example, eth1.

4. Start the following services:
   - Macvtap agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents:

```
$ openstack network agent list
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host     ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| 31e1bc1b-c872-4429-8fc3-2c8eba52634e | Metadata agent     |␣
↪compute1 | None              | True  | UP    | neutron-metadata-
↪agent    |
| 378f5550-feee-42aa-a1cb-e548b7c2601f | Open vSwitch agent |␣
↪compute1 | None              | True  | UP    | neutron-openvswitch-
↪agent |
| 7d2577d0-e640-42a3-b303-cb1eb077f2b6 | L3 agent           |␣
↪compute1 | nova              | True  | UP    | neutron-l3-agent         ␣
↪       |
| d5d7522c-ad14-4c63-ab45-f6420d6a81dd | Metering agent     |␣
↪compute1 | None              | True  | UP    | neutron-metering-
↪agent    |
| e838ef5c-75b1-4b12-84da-7bdbd62f1040 | DHCP agent         |␣
↪compute1 | nova              | True  | UP    | neutron-dhcp-agent       ␣
↪       |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
```

Create initial networks

This mechanism driver simply changes the virtual network interface driver for instances. Thus, you can reference the `Create initial networks` content for the prerequisite deployment example.

Verify network operation

This mechanism driver simply changes the virtual network interface driver for instances. Thus, you can reference the `Verify network operation` content for the prerequisite deployment example.

Network traffic flow

This mechanism driver simply removes the Linux bridge handling security groups on the compute nodes. Thus, you can reference the network traffic flow scenarios for the prerequisite deployment example.

8.2.17 MTU considerations

The Networking service uses the MTU of the underlying physical network to calculate the MTU for virtual network components including instance network interfaces. By default, it assumes a standard 1500-byte MTU for the underlying physical network.

The Networking service only references the underlying physical network MTU. Changing the underlying physical network device MTU requires configuration of physical network devices such as switches and routers.

Jumbo frames

The Networking service supports underlying physical networks using jumbo frames and also enables instances to use jumbo frames minus any overlay protocol overhead. For example, an underlying physical network with a 9000-byte MTU yields a 8950-byte MTU for instances using a VXLAN network with IPv4 endpoints. Using IPv6 endpoints for overlay networks adds 20 bytes of overhead for any protocol.

The Networking service supports the following underlying physical network architectures. Case 1 refers to the most common architecture. In general, architectures should avoid cases 2 and 3.

Note: After you adjust MTU configuration options in `neutron.conf` and `ml2_conf.ini`, you should update `mtu` attribute for all existing networks that need a new MTU. (Network MTU update is available for all core plugins that implement the `net-mtu-writable` API extension.)

Case 1

For typical underlying physical network architectures that implement a single MTU value, you can leverage jumbo frames using two options, one in the `neutron.conf` file and the other in the `ml2_conf.ini` file. Most environments should use this configuration.

For example, referencing an underlying physical network with a 9000-byte MTU:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `ml2_conf.ini` file:

```
[ml2]
path_mtu = 9000
```

Case 2

Some underlying physical network architectures contain multiple layer-2 networks with different MTU values. You can configure each flat or VLAN provider network in the bridge or interface mapping options of the layer-2 agent to reference a unique MTU value.

For example, referencing a 4000-byte MTU for `provider2`, a 1500-byte MTU for `provider3`, and a 9000-byte MTU for other networks using the Open vSwitch agent:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `openvswitch_agent.ini` file:

```
[ovs]
bridge_mappings = provider1:eth1,provider2:eth2,provider3:eth3
```

3. In the `ml2_conf.ini` file:

```
[ml2]
physical_network_mtus = provider2:4000,provider3:1500
path_mtu = 9000
```

Case 3

Some underlying physical network architectures contain a unique layer-2 network for overlay networks using protocols such as VXLAN and GRE.

For example, referencing a 4000-byte MTU for overlay networks and a 9000-byte MTU for other networks:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `ml2_conf.ini` file:

```
[ml2]
path_mtu = 4000
```

Note: Other networks including provider networks and flat or VLAN self-service networks assume the value of the `global_physnet_mtu` option.

Instance network interfaces (VIFs)

The DHCP agent provides an appropriate MTU value to instances using IPv4, while the L3 agent provides an appropriate MTU value to instances using IPv6. IPv6 uses RA via the L3 agent because the DHCP agent only supports IPv4. Instances using IPv4 and IPv6 should obtain the same MTU value regardless of method.

8.2.18 Network segment ranges

The network segment range service exposes the segment range management to be administered via the Neutron API. In addition, it introduces the ability for the administrator to control the segment ranges globally or on a per-tenant basis.

Why you need it

Before Stein, network segment ranges were configured as an entry in ML2 config file `ml2_conf.ini` that was statically defined for tenant network allocation and therefore had to be managed as part of the host deployment and management. When a regular tenant user creates a network, Neutron assigns the next free segmentation ID (VLAN ID, VNI etc.) from the configured segment ranges. Only an administrator can assign a specific segment ID via the provider extension.

The network segment range management service provides the following capabilities that the administrator may be interested in:

1. To check out the network segment ranges defined by the operators in the ML2 config file so that the admin can use this information to make segment range allocation.
2. To dynamically create and assign network segment ranges, which can help with the distribution of the underlying network connection mapping for privacy or dedicated business connection needs. This includes:
   - global shared network segment ranges
   - tenant-specific network segment ranges
3. To dynamically update a network segment range to offer the ability to adapt to the connection mapping changes.

4. To dynamically manage a network segment range when there are no segment ranges defined within the ML2 config file `ml2_conf.ini` and no restart of the Neutron server is required in this situation.
5. To check the availability and usage statistics of network segment ranges.

How it works

A network segment range manages a set of segments from which self-service networks can be allocated. The network segment range management service is admin-only.

As a regular project in an OpenStack cloud, you can not create a network segment range of your own and you just create networks in regular way.

If you are an admin, you can create a network segment range which can be shared (i.e. used by any regular project) or tenant-specific (i.e. assignment on a per-tenant basis). Your network segment ranges will not be visible to any other regular projects. Other CRUD operations are also supported.

When a tenant allocates a segment, it will first be allocated from an available segment range assigned to the tenant, and then a shared range if no tenant specific allocation is possible.

Default network segment ranges

A set of `default` network segment ranges are created out of the values defined in the ML2 config file: `network_vlan_ranges` for ml2_type_vlan, `vni_ranges` for ml2_type_vxlan, `tunnel_id_ranges` for ml2_type_gre and `vni_ranges` for ml2_type_geneve. They will be reloaded when Neutron server starts or restarts. The `default` network segment ranges are `read-only`, but will be treated as any other `shared` ranges on segment allocation.

The administrator can use the default network segment range information to make shared and/or per-tenant range creation and assignment.

Example configuration

Controller node

1. Enable the network segment range service plugin by appending `network_segment_range` to the list of `service_plugins` in the `neutron.conf` file on all nodes running the `neutron-server` service:

```
[DEFAULT]
# ...
service_plugins = ...,network_segment_range,...
```

2. Restart the `neutron-server` service.

Verify service operation

1. Source the administrative project credentials and list the enabled extensions.
2. Use the command **openstack extension list --network** to verify that the `Neutron Network Segment Range` extension with Alias `network-segment-range` is enabled.

```
$ openstack extension list --network
+-------------------------------+-----------------------+-----------------
↪----------------------------------------+
| Name                          | Alias                 | Description     ␣
↪                                        |
+-------------------------------+-----------------------+-----------------
↪----------------------------------------+
| ......                        | ......                | ......          ␣
↪                                        |
+-------------------------------+-----------------------+-----------------
↪----------------------------------------+
| Neutron Network Segment Range | network-segment-range | Provides support␣
↪for the network segment range management |
+-------------------------------+-----------------------+-----------------
↪----------------------------------------+
| ......                        | ......                | ......          ␣
↪                                        |
+-------------------------------+-----------------------+-----------------
↪----------------------------------------+
```

Workflow

At a high level, the basic workflow for a network segment range creation is the following:

1. The Cloud administrator:
   - Lists the existing network segment ranges.
   - Creates a shared or a tenant-specific network segment range based on the requirement.
2. A regular tenant creates a network in regular way. The network created will automatically allocate a segment from the segment ranges assigned to the tenant or shared if no tenant specific range available.

At a high level, the basic workflow for a network segment range update is the following:

1. The Cloud administrator:
   - Lists the existing network segment ranges and identifies the one that needs to be updated.
   - Updates the network segment range based on the requirement.
2. A regular tenant creates a network in regular way. The network created will automatically allocate a segment from the updated network segment ranges available.

List the network segment ranges or show a network segment range

As admin, list the existing network segment ranges:

```
$ openstack network segment range list
+--------------------------------------+-------------------+---------+-----
↪---+----------------------------------+--------------+------------------
↪+------------+------------+
| ID                                   | Name              | Default |␣
↪Shared | Project ID                       | Network Type | Physical␣
↪Network | Minimum ID | Maximum ID |
+--------------------------------------+-------------------+---------+-----
↪---+----------------------------------+--------------+------------------
↪+------------+------------+
| 20ce94e1-4e51-4aa0-a5f1-26bdfb5bd90e |                   | True    |␣
↪True    | None                             | vxlan        | None              ␣
↪   |          1 |        200 |
| 4b7af684-ec97-422d-ba38-8b9c2919ae67 | test_range_3      | False   |␣
↪False   | 7011dc7fccac4efda89dc3b7f0d0975a | gre          | None              ␣
↪   |        100 |        120 |
| a021e582-6b0f-49f5-90cb-79a670c61973 |                   | True    |␣
↪True    | None                             | vlan         | default           ␣
↪   |          1 |        100 |
| a3373630-969b-4ce9-bae7-dff0f8fa2f92 | test_range_2      | False   |␣
↪True    | None                             | vxlan        | None              ␣
↪   |        501 |        505 |
| a5707a8f-76f0-4f90-9aa7-c42bf54e94b5 |                   | True    |␣
↪True    | None                             | gre          | None              ␣
↪   |          1 |        150 |
| aad1b55b-43f1-46f9-8c35-85f270863ed6 |                   | True    |␣
↪True    | None                             | geneve       | None              ␣
↪   |          1 |        120 |
| e3233178-2866-4f40-b794-7c6fecdc8655 | test_range_1      | False   |␣
↪False   | 7011dc7fccac4efda89dc3b7f0d0975a | vlan         | group0-data0      ␣
↪   |         11 |         11 |
+--------------------------------------+-------------------+---------+-----
↪---+----------------------------------+--------------+------------------
↪+------------+------------+
```

The network segment ranges with `Default` as `True` are the ranges specified by the operators in the ML2 config file. Besides, there are also shared and tenant specific network segment ranges created by the admin previously.

The admin is also able to check/show the detailed information (e.g. availability and usage statistics) of a network segment range:

```
$ openstack network segment range show test_range_1
+------------------+-----------------------------------------------+
| Field            | Value                                         |
+------------------+-----------------------------------------------+
| available        | []                                            |
| default          | False                                         |
| id               | e3233178-2866-4f40-b794-7c6fecdc8655          |
| location         | None                                          |
| maximum          | 11                                            |
| minimum          | 11                                            |
```

(continues on next page)

(continued from previous page)

```
| name              | test_range_1                                 |
| network_type      | vlan                                         |
| physical_network  | group0-data0                                 |
| project_id        | 7011dc7fccac4efda89dc3b7f0d0975a             |
| shared            | False                                        |
| used              | {u'7011dc7fccac4efda89dc3b7f0d0975a': ['11']} |
+-------------------+----------------------------------------------+
```

Create or update the network segment range

As admin, create a network segment range based on your requirement:

```
$ openstack network segment range create --private --project demo \
--network-type vxlan --minimum 120 --maximum 140 test_range_4
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| available         | ['120-140']                          |
| default           | False                                |
| id                | c016dcda-5bc3-4e98-b41f-6773e92fcd2d |
| location          | None                                 |
| maximum           | 140                                  |
| minimum           | 120                                  |
| name              | test_range_4                         |
| network_type      | vxlan                                |
| physical_network  | None                                 |
| project_id        | 7011dc7fccac4efda89dc3b7f0d0975a     |
| shared            | False                                |
| used              | {}                                   |
+-------------------+--------------------------------------+
```

Update a network segment range based on your requirement:

```
$ openstack network segment range set --minimum 100 --maximum 150 \
test_range_4
```

Create a tenant network

Now, as project `demo` (to source the client environment script `demo-openrc` for `demo` project according to https://docs.openstack.org/keystone/latest/install/keystone-openrc-rdo.html), create a network in a regular way.

```
$ source demo-openrc
$ openstack network create test_net
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2019-02-25T23:20:36Z                 |
```

(continues on next page)

(continued from previous page)

```
| description               |                                      |
| dns_domain                |                                      |
| id                        | 39e5b95c-ad7a-40b5-9ec1-a4b4a8a43f14 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| location                  | None                                 |
| mtu                       | 1450                                 |
| name                      | test_net                             |
| port_security_enabled     | True                                 |
| project_id                | 7011dc7fccac4efda89dc3b7f0d0975a     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | None                                  |
| qos_policy_id             | None                                 |
| revision_number           | 2                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2019-02-25T23:20:36Z                 |
+---------------------------+--------------------------------------+
```

Then, switch back to the admin to check the segmentation ID of the tenant network created.

```
$ source admin-openrc
$ openstack network show test_net
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2019-02-25T23:20:36Z                 |
| description               |                                      |
| dns_domain                |                                      |
| id                        | 39e5b95c-ad7a-40b5-9ec1-a4b4a8a43f14 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| location                  | None                                 |
| mtu                       | 1450                                 |
| name                      | test_net                             |
| port_security_enabled     | True                                 |
| project_id                | 7011dc7fccac4efda89dc3b7f0d0975a     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 137                                  |
| qos_policy_id             | None                                 |
| revision_number           | 2                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
```

(continues on next page)

(continued from previous page)

```
| shared                    | False                              |
| status                    | ACTIVE                             |
| subnets                   |                                    |
| tags                      |                                    |
| updated_at                | 2019-02-25T23:20:36Z               |
+---------------------------+------------------------------------+
```

The tenant network created automatically allocates a segment with segmentation ID `137` from the network segment range with segmentation ID range `120-140` that is assigned to the tenant.

If no more available segment in the network segment range assigned to this tenant, then the segment allocation would refer to the `shared` segment ranges to check whether theres one segment available. If still there is no segment available, the allocation will fail as follows:

```
$ openstack network create test_net
$ Unable to create the network. No tenant network is available for
  allocation.
```

In this case, the admin is advised to check the availability and usage statistics of the related network segment ranges in order to take further actions (e.g. enlarging a segment range etc.).

Known limitations

- This service plugin is only compatible with ML2 core plugin for now. However, it is possible for other core plugins to support this feature with a follow-on effort.

8.2.19 Open vSwitch with DPDK datapath

This page serves as a guide for how to use the OVS with DPDK datapath functionality available in the Networking service as of the Mitaka release.

The basics

Open vSwitch (OVS) provides support for a Data Plane Development Kit (DPDK) datapath since OVS 2.2, and a DPDK-backed `vhost-user` virtual interface since OVS 2.4. The DPDK datapath provides lower latency and higher performance than the standard kernel OVS datapath, while DPDK-backed `vhost-user` interfaces can connect guests to this datapath. For more information on DPDK, refer to the DPDK website.

OVS with DPDK, or OVS-DPDK, can be used to provide high-performance networking between instances on OpenStack compute nodes.

Prerequisites

Using DPDK in OVS requires the following minimum software versions:

- OVS 2.4
- DPDK 2.0
- QEMU 2.1.0
- libvirt 1.2.13

Support of `vhost-user` multiqueue that enables use of multiqueue with `virtio-net` and `igb_uio` is available if the following newer versions are used:

- OVS 2.5
- DPDK 2.2
- QEMU 2.5
- libvirt 1.2.17

In both cases, install and configure Open vSwitch with DPDK support for each node. For more information, see the OVS-DPDK installation guide (select an appropriate OVS version in the *Branch* drop-down menu).

Neutron Open vSwitch vhost-user support for configuration of neutron OVS agent.

In case you wish to configure multiqueue, see the OVS configuration chapter on vhost-user in QEMU documentation.

The technical background of multiqueue is explained in the corresponding blueprint.

Additionally, OpenStack supports `vhost-user` reconnect feature starting from the Ocata release, as implementation of fix for bug 1604924. Starting from OpenStack Ocata release this feature is used without any configuration necessary in case the following minimum software versions are used:

- OVS 2.6
- DPDK 16.07
- QEMU 2.7

The support of this feature is not yet present in ML2 OVN and ODL mechanism drivers.

Using vhost-user interfaces

Once OVS and neutron are correctly configured with DPDK support, `vhost-user` interfaces are completely transparent to the guest (except in case of multiqueue configuration described below). However, guests must request huge pages. This can be done through flavors. For example:

```
$ openstack flavor set m1.large --property hw:mem_page_size=large
```

For more information about the syntax for `hw:mem_page_size`, refer to the Flavors guide.

Note: `vhost-user` requires file descriptor-backed shared memory. Currently, the only way to request this is by requesting large pages. This is why instances spawned on hosts with OVS-DPDK must request large pages. The aggregate flavor affinity filter can be used to associate flavors with large page support to hosts with OVS-DPDK support.

Create and add `vhost-user` network interfaces to instances in the same fashion as conventional interfaces. These interfaces can use the kernel `virtio-net` driver or a DPDK-compatible driver in the guest

```
$ openstack server create --nic net-id=$net_id ... testserver
```

Using vhost-user multiqueue

To use this feature, the following should be set in the flavor extra specs (flavor keys):

```
$ openstack flavor set $m1.large --property hw:vif_multiqueue_enabled=true
```

This setting can be overridden by the image metadata property if the feature is enabled in the extra specs:

```
$ openstack image set --property hw_vif_multiqueue_enabled=true IMAGE_NAME
```

Support of `virtio-net` multiqueue needs to be present in kernel of guest VM and is available starting from Linux kernel 3.8.

Check pre-set maximum for number of combined channels in channel configuration. Configuration of OVS and flavor done successfully should result in maximum being more than 1):

```
$ ethtool -l INTERFACE_NAME
```

To increase number of current combined channels run following command in guest VM:

```
$ ethtool -L INTERFACE_NAME combined QUEUES_NR
```

The number of queues should typically match the number of vCPUs defined for the instance. In newer kernel versions this is configured automatically.

Known limitations

- This feature is only supported when using the libvirt compute driver, and the KVM/QEMU hypervisor.
- Huge pages are required for each instance running on hosts with OVS-DPDK. If huge pages are not present in the guest, the interface will appear but will not function.
- Expect performance degradation of services using tap devices: these devices do not support DPDK. Example services include DVR and FWaaS.
- When the `ovs_use_veth` option is set to `True`, any traffic sent from a DHCP namespace will have an incorrect TCP checksum. This means that if `enable_isolated_metadata` is set to `True` and metadata service is reachable through the DHCP namespace, responses from metadata will be dropped due to an invalid checksum. In such cases, `ovs_use_veth` should be switched to `False` and Open vSwitch (OVS) internal ports should be used instead.

8.2.20 Open vSwitch hardware offloading

The purpose of this page is to describe how to enable Open vSwitch hardware offloading functionality available in OpenStack (using OpenStack Networking). This functionality was first introduced in the OpenStack Pike release. This page intends to serve as a guide for how to configure OpenStack Networking and OpenStack Compute to enable Open vSwitch hardware offloading.

The basics

Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols. Open vSwitch (OVS) allows Virtual Machines (VM) to communicate with each other and with the outside world. The OVS software based solution is CPU intensive, affecting system performance and preventing fully utilizing available bandwidth.

Term	Definition
PF	Physical Function. The physical Ethernet controller that supports SR-IOV.
VF	Virtual Function. The virtual PCIe device created from a physical Ethernet controller.
Representor Port	Virtual network interface similar to SR-IOV port that represents Nova instance.
First Compute Node	OpenStack Compute Node that can host Compute instances (Virtual Machines).
Second Compute Node	OpenStack Compute Node that can host Compute instances (Virtual Machines).

Supported Ethernet controllers

The following manufacturers are known to work:

- Mellanox ConnectX-4 NIC (VLAN Offload)
- Mellanox ConnectX-4 Lx/ConnectX-5 NICs (VLAN/VXLAN Offload)
- Broadcom NetXtreme-S series NICs
- Broadcom NetXtreme-E series NICs

For information on **Mellanox Ethernet Cards**, see Mellanox: Ethernet Cards - Overview.

Prerequisites

- Linux Kernel >= 4.13
- Open vSwitch >= 2.8
- iproute >= 4.12
- Mellanox or Broadcom NIC

Note: Mellanox NIC FW that supports Open vSwitch hardware offloading:

ConnectX-5 >= 16.21.0338

ConnectX-4 >= 12.18.2000

ConnectX-4 Lx >= 14.21.0338

Using Open vSwitch hardware offloading

In order to enable Open vSwitch hardware offloading, the following steps are required:

1. Enable SR-IOV
2. Configure NIC to switchdev mode (relevant Nodes)
3. Enable Open vSwitch hardware offloading

Note: Throughout this guide, `enp3s0f0` is used as the PF and `eth3` is used as the representor port. These ports may vary in different environments.

Note: Throughout this guide, we use `systemctl` to restart OpenStack services. This is correct for `systemd` OS. Other methods to restart services should be used in other environments.

Create Compute virtual functions

Create the VFs for the network interface that will be used for SR-IOV. We use `enp3s0f0` as PF, which is also used as the interface for the VLAN provider network and has access to the private networks of all nodes.

Note: The following steps detail how to create VFs using Mellanox ConnectX-4 and SR-IOV Ethernet cards on an Intel system. Steps may be different for the hardware of your choice.

1. Ensure SR-IOV and VT-d are enabled on the system. Enable IOMMU in Linux by adding `intel_iommu=on` to kernel parameters, for example, using GRUB.
2. On each Compute node, create the VFs:

```
# echo '4' > /sys/class/net/enp3s0f0/device/sriov_numvfs
```

   Note: A network interface can be used both for PCI passthrough, using the PF, and SR-IOV, using the VFs. If the PF is used, the VF number stored in the `sriov_numvfs` file is lost. If the PF is attached again to the operating system, the number of VFs assigned to this interface will be zero. To keep the number of VFs always assigned to this interface, update a relevant file according to your OS. See some examples below:

   In Ubuntu, modifying the `/etc/network/interfaces` file:

```
auto enp3s0f0
iface enp3s0f0 inet dhcp
pre-up echo '4' > /sys/class/net/enp3s0f0/device/sriov_numvfs
```

In Red Hat, modifying the `/sbin/ifup-local` file:

```
#!/bin/sh
if [[ "$1" == "enp3s0f0" ]]
then
    echo '4' > /sys/class/net/enp3s0f0/device/sriov_numvfs
fi
```

Warning: Alternatively, you can create VFs by passing the `max_vfs` to the kernel module of your network interface. However, the `max_vfs` parameter has been deprecated, so the PCI /sys interface is the preferred method.

You can determine the maximum number of VFs a PF can support:

```
# cat /sys/class/net/enp3s0f0/device/sriov_totalvfs
8
```

3. Verify that the VFs have been created and are in `up` state:

Note: The PCI bus number of the PF (03:00.0) and VFs (03:00.2 .. 03:00.5) will be used later.

```
# ip link show enp3s0f0
8: enp3s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq␣
↪state UP mode DEFAULT qlen 1000
   link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
   vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

If the interfaces are down, set them to `up` before launching a guest, otherwise the instance will fail to spawn:

```
# ip link set enp3s0f0 up
```

Configure Open vSwitch hardware offloading

1. Change the e-switch mode from legacy to switchdev on the PF device. This will also create the VF representor network devices in the host OS.

```
# echo 0000:03:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
```

This tells the driver to unbind VF 03:00.2

Note: This should be done for all relevant VFs (in this example 0000:03:00.2 .. 0000:03:00.5)

2. Enable Open vSwitch hardware offloading, set PF to switchdev mode and bind VFs back.

```
# sudo devlink dev eswitch set pci/0000:03:00.0 mode switchdev
# sudo ethtool -K enp3s0f0 hw-tc-offload on
# echo 0000:03:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

Note: This should be done for all relevant VFs (in this example 0000:03:00.2 .. 0000:03:00.5)

3. Restart Open vSwitch

```
# sudo systemctl enable openvswitch.service
# sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
# sudo systemctl restart openvswitch.service
```

Note: The given aging of OVS is given in milliseconds and can be controlled with:

```
# ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

Configure Nodes (VLAN Configuration)

1. Update `/etc/neutron/plugins/ml2/ml2_conf.ini` on Controller nodes

```
[ml2]
tenant_network_types = vlan
type_drivers = vlan
mechanism_drivers = openvswitch
```

2. Update `/etc/neutron/neutron.conf` on Controller nodes

```
[DEFAULT]
core_plugin = ml2
```

3. Update `/etc/nova/nova.conf` on Controller nodes

```
[filter_scheduler]
enabled_filters = PciPassthroughFilter
```

4. Update `/etc/nova/nova.conf` on Compute nodes

```
[pci]
#VLAN Configuration passthrough_whitelist example
passthrough_whitelist ={"'"address"'":"'"*:'"03:00"'.*"'","'"physical_
↪network"'":"'"physnet2"'"}
```

Configure Nodes (VXLAN Configuration)

1. Update /etc/neutron/plugins/ml2/ml2_conf.ini on Controller nodes

```
[ml2]
tenant_network_types = vxlan
type_drivers = vxlan
mechanism_drivers = openvswitch
```

2. Update /etc/neutron/neutron.conf on Controller nodes

```
[DEFAULT]
core_plugin = ml2
```

3. Update /etc/nova/nova.conf on Controller nodes

```
[filter_scheduler]
enabled_filters = PciPassthroughFilter
```

4. Update /etc/nova/nova.conf on Compute nodes

Note: VXLAN configuration requires physical_network to be null.

```
[pci]
#VLAN Configuration passthrough_whitelist example
passthrough_whitelist ={"'"address"'":"'"*:'"03:00"'.*"'","'"physical_
↪network"'":null}
```

5. Restart nova and neutron services

```
# sudo systemctl restart openstack-nova-compute.service
# sudo systemctl restart openstack-nova-scheduler.service
# sudo systemctl restart neutron-server.service
```

Validate Open vSwitch hardware offloading

Note: In this example we will bring up two instances on different Compute nodes and send ICMP echo packets between them. Then we will check TCP packets on a representor port and we will see that only the first packet will be shown there. All the rest will be offloaded.

1. Create a port direct on private network

```
# openstack port create --network private --vnic-type=direct --
↪binding-profile '{"capabilities": ["switchdev"]}' direct_port1
```

2. Create an instance using the direct port on First Compute Node

```
# openstack server create --flavor m1.small --image cloud_image --nic␣
↪port-id=direct_port1 vm1
```

3. Repeat steps above and create a second instance on Second Compute Node

```
# openstack port create --network private --vnic-type=direct --
↪binding-profile '{"capabilities": ["switchdev"]}' direct_port2
# openstack server create --flavor m1.small --image mellanox_fedora --
↪nic port-id=direct_port2 vm2
```

Note: You can use availability-zone nova:compute_node_1 option to set the desired Compute Node

4. Connect to instance1 and send ICMP Echo Request packets to instance2

```
# vncviewer localhost:5900
vm_1# ping vm2
```

5. Connect to Second Compute Node and find representor port of the instance

Note: Find a representor port first, in our case its eth3

```
compute_node2# ip link show enp3s0f0
6: enp3s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq␣
↪master ovs-system state UP mode DEFAULT group default qlen 1000
   link/ether ec:0d:9a:46:9e:84 brd ff:ff:ff:ff:ff:ff
   vf 0 MAC 00:00:00:00:00:00, spoof checking off, link-state enable,␣
↪trust off, query_rss off
   vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-state enable,␣
↪trust off, query_rss off
   vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-state enable,␣
↪trust off, query_rss off
   vf 3 MAC fa:16:3e:b9:b8:ce, vlan 57, spoof checking on, link-state␣
↪enable, trust off, query_rss off

compute_node2# ls -l /sys/class/net/
lrwxrwxrwx 1 root root 0 Sep 11 10:54 eth0 -> ../../devices/virtual/
↪net/eth0
lrwxrwxrwx 1 root root 0 Sep 11 10:54 eth1 -> ../../devices/virtual/
↪net/eth1
lrwxrwxrwx 1 root root 0 Sep 11 10:54 eth2 -> ../../devices/virtual/
↪net/eth2
lrwxrwxrwx 1 root root 0 Sep 11 10:54 eth3 -> ../../devices/virtual/
↪net/eth3

compute_node2# sudo ovs-dpctl show
system@ovs-system:
  lookups: hit:1684 missed:1465 lost:0
  flows: 0
  masks: hit:8420 total:1 hit/pkt:2.67
  port 0: ovs-system (internal)
  port 1: br-enp3s0f0 (internal)
  port 2: br-int (internal)
  port 3: br-ex (internal)
  port 4: enp3s0f0
  port 5: tapfdc744bb-61 (internal)
  port 6: qr-a7b1e843-4f (internal)
```

(continues on next page)

(continued from previous page)

```
  port 7: qg-79a77e6d-8f (internal)
  port 8: qr-f55e4c5f-f3 (internal)
  port 9: eth3
```

6. Check traffic on the representor port. Verify that only the first ICMP packet appears.

```
compute_node2# tcpdump -nnn -i eth3

tcpdump: verbose output suppressed, use -v or -vv for full protocol␣
↪decode
listening on eth3, link-type EN10MB (Ethernet), capture size 262144␣
↪bytes
17:12:41.220447 ARP, Request who-has 172.0.0.10 tell 172.0.0.13,␣
↪length 46
17:12:41.220684 ARP, Reply 172.0.0.10 is-at fa:16:3e:f2:8b:23, length␣
↪42
17:12:41.260487 IP 172.0.0.13 > 172.0.0.10: ICMP echo request, id␣
↪1263, seq 1, length 64
17:12:41.260778 IP 172.0.0.10 > 172.0.0.13: ICMP echo reply, id 1263,␣
↪seq 1, length 64
17:12:46.268951 ARP, Request who-has 172.0.0.13 tell 172.0.0.10,␣
↪length 42
17:12:46.271771 ARP, Reply 172.0.0.13 is-at fa:16:3e:1a:10:05, length␣
↪46
17:12:55.354737 IP6 fe80::f816:3eff:fe29:8118 > ff02::1: ICMP6,␣
↪router advertisement, length 64
17:12:56.106705 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP,␣
↪Request from 62:21:f0:89:40:73, length 300
```

8.2.21 Native Open vSwitch firewall driver

Historically, Open vSwitch (OVS) could not interact directly with *iptables* to implement security groups. Thus, the OVS agent and Compute service use a Linux bridge between each instance (VM) and the OVS integration bridge `br-int` to implement security groups. The Linux bridge device contains the *iptables* rules pertaining to the instance. In general, additional components between instances and physical network infrastructure cause scalability and performance problems. To alleviate such problems, the OVS agent includes an optional firewall driver that natively implements security groups as flows in OVS rather than the Linux bridge device and *iptables*. This increases scalability and performance.

Configuring heterogeneous firewall drivers

L2 agents can be configured to use differing firewall drivers. There is no requirement that they all be the same. If an agent lacks a firewall driver configuration, it will default to what is configured on its server. This also means there is no requirement that the server has any firewall driver configured at all, as long as the agents are configured correctly.

Prerequisites

The native OVS firewall implementation requires kernel and user space support for *conntrack*, thus requiring minimum versions of the Linux kernel and Open vSwitch. All cases require Open vSwitch version 2.5 or newer.

- Kernel version 4.3 or newer includes *conntrack* support.
- Kernel version 3.3, but less than 4.3, does not include *conntrack* support and requires building the OVS modules.

Enable the native OVS firewall driver

- On nodes running the Open vSwitch agent, edit the `openvswitch_agent.ini` file and enable the firewall driver.

```
[securitygroup]
firewall_driver = openvswitch
```

For more information, see the *Open vSwitch Firewall Driver* and the video.

Using GRE tunnels inside VMs with OVS firewall driver

If GRE tunnels from VM to VM are going to be used, the native OVS firewall implementation requires `nf_conntrack_proto_gre` module to be loaded in the kernel on nodes running the Open vSwitch agent. It can be loaded with the command:

```
# modprobe nf_conntrack_proto_gre
```

Some Linux distributions have files that can be used to automatically load kernel modules at boot time, for example, `/etc/modules`. Check with your distribution for further information.

This isnt necessary to use `gre` tunnel network type Neutron.

8.2.22 Quality of Service (QoS)

QoS is defined as the ability to guarantee certain network requirements like bandwidth, latency, jitter, and reliability in order to satisfy a Service Level Agreement (SLA) between an application provider and end users.

Network devices such as switches and routers can mark traffic so that it is handled with a higher priority to fulfill the QoS conditions agreed under the SLA. In other cases, certain network traffic such as Voice over IP (VoIP) and video streaming needs to be transmitted with minimal bandwidth constraints. On a system without network QoS management, all traffic will be transmitted in a best-effort manner making it impossible to guarantee service delivery to customers.

QoS is an advanced service plug-in. QoS is decoupled from the rest of the OpenStack Networking code on multiple levels and it is available through the ml2 extension driver.

Details about the DB models, API extension, and use cases are out of the scope of this guide but can be found in the Neutron QoS specification.

Supported QoS rule types

QoS supported rule types are now available as `VALID_RULE_TYPES` in QoS rule types:

- bandwidth_limit: Bandwidth limitations on networks, ports or floating IPs.
- dscp_marking: Marking network traffic with a DSCP value.
- minimum_bandwidth: Minimum bandwidth constraints on certain types of traffic.

Any QoS driver can claim support for some QoS rule types by providing a driver property called `supported_rules`, the QoS driver manager will recalculate rule types dynamically that the QoS driver supports.

The following table shows the Networking back ends, QoS supported rules, and traffic directions (from the VM point of view).

Table 5: **Networking back ends, supported rules, and traffic direction**

Rule \ back end	Open vSwitch	SR-IOV	Linux bridge	OVN
Bandwidth limit	Egress \ Ingress	Egress (1)	Egress \ Ingress	Egress \ Ingress
Minimum bandwidth	Egress \ Ingress (2)	Egress \ Ingress (2)	•	•
DSCP marking	Egress	•	Egress	Egress

Note:

(1) Max burst parameter is skipped because it is not supported by the IP tool.

(2) Placement based enforcement works for both egress and ingress directions, but dataplane enforcement depends on the backend.

Table 6: **Neutron backends, supported directions and enforcement types for Minimum Bandwidth rule**

Enforcement type Backend	Open vSwitch	SR-IOV	Linux Bridge	OVN
Dataplane	Egress (3)	Egress (1)	•	•
Placement	Egress/Ingress (2)	Egress/Ingress (2)	•	•

Note:

(1) Since Newton

(2) Since Stein

(3) Open vSwitch minimum bandwidth support is only implemented for egress direction and only for networks without tunneled traffic (only VLAN and flat network types).

In the most simple case, the property can be represented by a simple Python list defined on the class.

For an ml2 plug-in, the list of supported QoS rule types and parameters is defined as a common subset of rules supported by all active mechanism drivers. A QoS rule is always attached to a QoS policy. When a rule is created or updated:

- The QoS plug-in will check if this rule and parameters are supported by any active mechanism driver if the QoS policy is not attached to any port or network.
- The QoS plug-in will check if this rule and parameters are supported by the mechanism drivers managing those ports if the QoS policy is attached to any port or network.

Valid DSCP Marks

Valid DSCP mark values are even numbers between 0 and 56, except 2-6, 42, 44, and 50-54. The full list of valid DSCP marks is:

0, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 46, 48, 56

Configuration

To enable the service on a cloud with the architecture described in Networking architecture, follow the steps below:

On the controller nodes:

1. Add the QoS service to the `service_plugins` setting in `/etc/neutron/neutron.conf`. For example:

```
service_plugins = router,metering,qos
```

2. Optionally, set the needed `notification_drivers` in the `[qos]` section in `/etc/neutron/neutron.conf` (`message_queue` is the default).
3. Optionally, in order to enable the floating IP QoS extension `qos-fip`, set the `service_plugins` option in `/etc/neutron/neutron.conf` to include both `router` and `qos`. For example:

```
service_plugins = router,qos
```

4. In `/etc/neutron/plugins/ml2/ml2_conf.ini`, add `qos` to `extension_drivers` in the `[ml2]` section. For example:

```
[ml2]
extension_drivers = port_security,qos
```

5. Edit the configuration file for the agent you are using and set the `extensions` to include `qos` in the `[agent]` section of the configuration file. The agent configuration file will reside in `/etc/neutron/plugins/ml2/<agent_name>_agent.ini` where `agent_name` is the name of the agent being used (for example `openvswitch`). For example:

```
[agent]
extensions = qos
```

On the network and compute nodes:

1. Edit the configuration file for the agent you are using and set the `extensions` to include `qos` in the `[agent]` section of the configuration file. The agent configuration file will reside in `/etc/neutron/plugins/ml2/<agent_name>_agent.ini` where `agent_name` is the name of the agent being used (for example `openvswitch`). For example:

```
[agent]
extensions = qos
```

2. Optionally, in order to enable QoS for floating IPs, set the `extensions` option in the `[agent]` section of `/etc/neutron/l3_agent.ini` to include `fip_qos`. If `dvr` is enabled, this has to be done for all the L3 agents. For example:

```
[agent]
extensions = fip_qos
```

Note: Floating IP associated to neutron port or to port forwarding can all have bandwidth limit since Stein release. These neutron server side and agent side extension configs will enable it once for all.

1. Optionally, in order to enable QoS for router gateway IPs, set the `extensions` option in the `[agent]` section of `/etc/neutron/l3_agent.ini` to include `gateway_ip_qos`. Set this to all the `dvr_snat` or `legacy` L3 agents. For example:

```
[agent]
extensions = gateway_ip_qos
```

   And `gateway_ip_qos` should work together with the `fip_qos` in L3 agent for centralized routers, then all L3 IPs with binding QoS policy can be limited under the QoS bandwidth limit rules:

```
[agent]
extensions = fip_qos, gateway_ip_qos
```

2. As rate limit doesnt work on Open vSwitchs `internal` ports, optionally, as a workaround, to make QoS bandwidth limit work on routers gateway ports, set `ovs_use_veth` to `True` in `DEFAULT` section in `/etc/neutron/l3_agent.ini`

```
[DEFAULT]
ovs_use_veth = True
```

Note: QoS currently works with ml2 only (SR-IOV, Open vSwitch, and linuxbridge are drivers enabled for QoS).

DSCP marking on outer header for overlay networks

When using overlay networks (e.g., VxLAN), the DSCP marking rule only applies to the inner header, and during encapsulation, the DSCP mark is not automatically copied to the outer header.

1. In order to set the DSCP value of the outer header, modify the `dscp` configuration option in `/etc/neutron/plugins/ml2/<agent_name>_agent.ini` where `<agent_name>` is the name of the agent being used (e.g., `openvswitch`):

   ```
   [agent]
   dscp = 8
   ```

2. In order to copy the DSCP field of the inner header to the outer header, change the `dscp_inherit` configuration option to true in `/etc/neutron/plugins/ml2/<agent_name>_agent.ini` where `<agent_name>` is the name of the agent being used (e.g., `openvswitch`):

   ```
   [agent]
   dscp_inherit = true
   ```

   If the `dscp_inherit` option is set to true, the previous `dscp` option is overwritten.

Trusted projects policy.json configuration

If projects are trusted to administrate their own QoS policies in your cloud, neutrons file `policy.json` can be modified to allow this.

Modify `/etc/neutron/policy.json` policy entries as follows:

```
"get_policy": "rule:regular_user",
"create_policy": "rule:regular_user",
"update_policy": "rule:regular_user",
"delete_policy": "rule:regular_user",
"get_rule_type": "rule:regular_user",
```

To enable bandwidth limit rule:

```
"get_policy_bandwidth_limit_rule": "rule:regular_user",
"create_policy_bandwidth_limit_rule": "rule:regular_user",
"delete_policy_bandwidth_limit_rule": "rule:regular_user",
"update_policy_bandwidth_limit_rule": "rule:regular_user",
```

To enable DSCP marking rule:

```
"get_policy_dscp_marking_rule": "rule:regular_user",
"create_dscp_marking_rule": "rule:regular_user",
"delete_dscp_marking_rule": "rule:regular_user",
"update_dscp_marking_rule": "rule:regular_user",
```

To enable minimum bandwidth rule:

```
"get_policy_minimum_bandwidth_rule": "rule:regular_user",
"create_policy_minimum_bandwidth_rule": "rule:regular_user",
"delete_policy_minimum_bandwidth_rule": "rule:regular_user",
"update_policy_minimum_bandwidth_rule": "rule:regular_user",
```

User workflow

QoS policies are only created by admins with the default `policy.json`. Therefore, you should have the cloud operator set them up on behalf of the cloud projects.

If projects are trusted to create their own policies, check the trusted projects `policy.json` configuration section.

First, create a QoS policy and its bandwidth limit rule:

```
$ openstack network qos policy create bw-limiter
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | 5df855e9-a833-49a3-9c82-c0839a5f103f |
| is_default        | False                                |
| name              | bw-limiter                           |
| project_id        | 4db7c1ed114a4a7fb0f077148155c500     |
| rules             | []                                   |
| shared            | False                                |
+-------------------+--------------------------------------+


$ openstack network qos rule create --type bandwidth-limit --max-kbps 3000 \
    --max-burst-kbits 2400 --egress bw-limiter
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| direction      | egress                               |
| id             | 92ceb52f-170f-49d0-9528-976e2fee2d6f |
| max_burst_kbps | 2400                                 |
| max_kbps       | 3000                                 |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+
```

Note: The QoS implementation requires a burst value to ensure proper behavior of bandwidth limit rules in the Open vSwitch and Linux bridge agents. Configuring the proper burst value is very important. If the burst value is set too low, bandwidth usage will be throttled even with a proper bandwidth limit setting. This issue is discussed in various documentation sources, for example in Junipers documentation. For TCP traffic it is recommended to set burst value as 80% of desired bandwidth limit value. For example, if the bandwidth limit is set to 1000kbps then enough burst value will be 800kbit. If the configured burst value is too low, achieved bandwidth limit will be lower than expected. If the configured burst value is too high, too few packets could be limited and achieved bandwidth limit would be higher than expected. If you do not provide a value, it defaults to 80% of the bandwidth limit which works for typical TCP traffic.

Second, associate the created policy with an existing neutron port. In order to do this, user extracts the port id to be associated to the already created policy. In the next example, we will assign the `bw-limiter` policy to the VM with IP address `192.0.2.1`.

```
$ openstack port list
+--------------------------------------+-----------------------------------
↪+
| ID                                   | Fixed IP Addresses                ␣
↪|
+--------------------------------------+-----------------------------------
↪+
| 0271d1d9-1b16-4410-bd74-82cdf6dcb5b3 | { ... , "ip_address": "192.0.2.1"}
↪|
| 88101e57-76fa-4d12-b0e0-4fc7634b874a | { ... , "ip_address": "192.0.2.3"}
↪|
| e04aab6a-5c6c-4bd9-a600-33333551a668 | { ... , "ip_address": "192.0.2.2"}
↪|
+--------------------------------------+-----------------------------------
↪+

$ openstack port set --qos-policy bw-limiter \
    88101e57-76fa-4d12-b0e0-4fc7634b874a
```

In order to detach a port from the QoS policy, simply update again the port configuration.

```
$ openstack port unset --qos-policy 88101e57-76fa-4d12-b0e0-4fc7634b874a
```

Ports can be created with a policy attached to them too.

```
$ openstack port create --qos-policy bw-limiter --network private port1
+-----------------------+--------------------------------------------------
↪+
| 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            | 2017-05-15T08:43:00Z                             ␣
↪|
| data_plane_status     | None                                             ␣
↪|
| description           |                                                  ␣
↪|
| device_id             |                                                  ␣
↪|
| device_owner          |                                                  ␣
↪|
```

(continues on next page)

(continued from previous page)

```
| dns_assignment        | None                                                    ␣
↪|
| dns_name              | None                                                    ␣
↪|
| extra_dhcp_opts       |                                                         ␣
↪|
| fixed_ips             | ip_address='10.0.10.4', subnet_id='292f8c1e-...'␣
↪|
| id                    | f51562ee-da8d-42de-9578-f6f5cb248226                    ␣
↪|
| ip_address            | None                                                    ␣
↪|
| mac_address           | fa:16:3e:d9:f2:ba                                       ␣
↪|
| name                  | port1                                                   ␣
↪|
| network_id            | 55dc2f70-0f92-4002-b343-ca34277b0234                    ␣
↪|
| option_name           | None                                                    ␣
↪|
| option_value          | None                                                    ␣
↪|
| port_security_enabled | False                                                   ␣
↪|
| project_id            | 4db7c1ed114a4a7fb0f077148155c500                        ␣
↪|
| qos_policy_id         | 5df855e9-a833-49a3-9c82-c0839a5f103f                    ␣
↪|
| revision_number       | 6                                                       ␣
↪|
| security_group_ids    | 0531cc1a-19d1-4cc7-ada5-49f8b08245be                    ␣
↪|
| status                | DOWN                                                    ␣
↪|
| subnet_id             | None                                                    ␣
↪|
| tags                  | []                                                      ␣
↪|
| trunk_details         | None                                                    ␣
↪|
| updated_at            | 2017-05-15T08:43:00Z                                    ␣
↪|
+-----------------------+--------------------------------------------------
↪+
```

You can attach networks to a QoS policy. The meaning of this is that any compute port connected to the network will use the network policy by default unless the port has a specific policy attached to it. Internal network owned ports like DHCP and internal router ports are excluded from network policy application.

In order to attach a QoS policy to a network, update an existing network, or initially create the network attached to the policy.

```
$ openstack network set --qos-policy bw-limiter private
```

The created policy can be associated with an existing floating IP. In order to do this, user extracts the floating IP id to be associated to the already created policy. In the next example, we will assign the `bw-limiter` policy to the floating IP address `172.16.100.18`.

```
$ openstack floating ip list
+--------------------------------------+---------------------+------------
↪-----+------+-----+
| ID                                   | Floating IP Address | Fixed IP␣
↪Address | Port | ... |
+--------------------------------------+---------------------+------------
↪-----+------+-----+
| 1163d127-6df3-44bb-b69c-c0e916303eb3 | 172.16.100.9        | None        ␣
↪      | None | ... |
| d0ed7491-3eb7-4c4f-a0f0-df04f10a067c | 172.16.100.18       | None        ␣
↪      | None | ... |
| f5a9ed48-2e9f-411c-8787-2b6ecd640090 | 172.16.100.2        | None        ␣
↪      | None | ... |
+--------------------------------------+---------------------+------------
↪-----+------+-----+
```

```
$ openstack floating ip set --qos-policy bw-limiter d0ed7491-3eb7-4c4f-
↪a0f0-df04f10a067c
```

In order to detach a floating IP from the QoS policy, simply update the floating IP configuration.

```
$ openstack floating ip set --no-qos-policy d0ed7491-3eb7-4c4f-a0f0-
↪df04f10a067c
```

Or use the `unset` action.

```
$ openstack floating ip unset --qos-policy d0ed7491-3eb7-4c4f-a0f0-
↪df04f10a067c
```

Floating IPs can be created with a policy attached to them too.

```
$ openstack floating ip create --qos-policy bw-limiter public
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-12-06T02:12:09Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 172.16.100.12                        |
| floating_network_id | 4065eb05-cccb-4048-988c-e8c5480a746f |
| id                  | 6a0efeef-462b-4312-b4ad-627cde8a20e6 |
| name                | 172.16.100.12                        |
| port_id             | None                                 |
| project_id          | 916e39e8be52433ba040da3a3a6d0847     |
| qos_policy_id       | 5df855e9-a833-49a3-9c82-c0839a5f103f |
| revision_number     | 1                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2017-12-06T02:12:09Z                 |
+---------------------+--------------------------------------+
```

The QoS bandwidth limit rules attached to a floating IP will become active when you associate the latter with a port. For example, to associate the previously created floating IP `172.16.100.12` to the instance port with uuid a7f25e73-4288-4a16-93b9-b71e6fd00862 and fixed IP 192.168.222.5:

```
$ openstack floating ip set --port a7f25e73-4288-4a16-93b9-b71e6fd00862 \
    0eeb1f8a-de96-4cd9-a0f6-3f535c409558
```

Note: The QoS policy attached to a floating IP is not applied to a port, it is applied to an associated floating IP only. Thus the ID of QoS policy attached to a floating IP will not be visible in a ports qos_policy_id field after asscoating a floating IP to the port. It is only visible in the floating IP attributes.

Note: For now, the L3 agent floating IP QoS extension only supports bandwidth_limit rules. Other rule types (like DSCP marking) will be silently ignored for floating IPs. A QoS policy that does not contain any bandwidth_limit rules will have no effect when attached to a floating IP.

If floating IP is bound to a port, and both have binding QoS bandwidth rules, the L3 agent floating IP QoS extension ignores the behavior of the port QoS, and installs the rules from the QoS policy associated to the floating IP on the appropriate device in the router namespace.

Each project can have at most one default QoS policy, although it is not mandatory. If a default QoS policy is defined, all new networks created within this project will have this policy assigned, as long as no other QoS policy is explicitly attached during the creation process. If the default QoS policy is unset, no change to existing networks will be made.

In order to set a QoS policy as default, the parameter --default must be used. To unset this QoS policy as default, the parameter --no-default must be used.

```
$ openstack network qos policy create --default bw-limiter
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | 5df855e9-a833-49a3-9c82-c0839a5f103f |
| is_default        | True                                 |
| name              | bw-limiter                           |
| project_id        | 4db7c1ed114a4a7fb0f077148155c500     |
| rules             | []                                   |
| shared            | False                                |
+-------------------+--------------------------------------+

$ openstack network qos policy set --no-default bw-limiter
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | 5df855e9-a833-49a3-9c82-c0839a5f103f |
| is_default        | False                                |
| name              | bw-limiter                           |
| project_id        | 4db7c1ed114a4a7fb0f077148155c500     |
| rules             | []                                   |
| shared            | False                                |
+-------------------+--------------------------------------+
```

Administrator enforcement

Administrators are able to enforce policies on project ports or networks. As long as the policy is not shared, the project is not be able to detach any policy attached to a network or port.

If the policy is shared, the project is able to attach or detach such policy from its own ports and networks.

Rule modification

You can modify rules at runtime. Rule modifications will be propagated to any attached port.

```
$ openstack network qos rule set --max-kbps 2000 --max-burst-kbits 1600 \
    --ingress bw-limiter 92ceb52f-170f-49d0-9528-976e2fee2d6f

$ openstack network qos rule show \
    bw-limiter 92ceb52f-170f-49d0-9528-976e2fee2d6f
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| direction      | ingress                              |
| id             | 92ceb52f-170f-49d0-9528-976e2fee2d6f |
| max_burst_kbps | 1600                                 |
| max_kbps       | 2000                                 |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+
```

Just like with bandwidth limiting, create a policy for DSCP marking rule:

```
$ openstack network qos policy create dscp-marking
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | d1f90c76-fbe8-4d6f-bb87-a9aea997ed1e |
| is_default        | False                                |
| name              | dscp-marking                         |
| project_id        | 4db7c1ed114a4a7fb0f077148155c500     |
| rules             | []                                   |
| shared            | False                                |
+-------------------+--------------------------------------+
```

You can create, update, list, delete, and show DSCP markings with the neutron client:

```
$ openstack network qos rule create --type dscp-marking --dscp-mark 26 \
    dscp-marking
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| dscp_mark      | 26                                   |
| id             | 115e4f70-8034-4176-8fe9-2c47f8878a7d |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+
```

```
$ openstack network qos rule set --dscp-mark 22 \
    dscp-marking 115e4f70-8034-4176-8fe9-2c47f8878a7d

$ openstack network qos rule list dscp-marking
+--------------------------------------+----------------------------------+
| ID                                   | DSCP Mark                        |
+--------------------------------------+----------------------------------+
| 115e4f70-8034-4176-8fe9-2c47f8878a7d | 22                               |
+--------------------------------------+----------------------------------+

$ openstack network qos rule show \
    dscp-marking 115e4f70-8034-4176-8fe9-2c47f8878a7d
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| dscp_mark      | 22                                   |
| id             | 115e4f70-8034-4176-8fe9-2c47f8878a7d |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+

$ openstack network qos rule delete \
    dscp-marking 115e4f70-8034-4176-8fe9-2c47f8878a7d
```

You can also include minimum bandwidth rules in your policy:

```
$ openstack network qos policy create bandwidth-control
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | 8491547e-add1-4c6c-a50e-42121237256c |
| is_default        | False                                |
| name              | bandwidth-control                    |
| project_id        | 7cc5a84e415d48e69d2b06aa67b317d8     |
| revision_number   | 1                                    |
| rules             | []                                   |
| shared            | False                                |
+-------------------+--------------------------------------+

$ openstack network qos rule create \
  --type minimum-bandwidth --min-kbps 1000 --egress bandwidth-control
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| direction  | egress                               |
| id         | da858b32-44bc-43c9-b92b-cf6e2fa836ab |
| min_kbps   | 1000                                 |
| name       | None                                 |
| project_id |                                      |
+------------+--------------------------------------+
```

A policy with a minimum bandwidth ensures best efforts are made to provide no less than the specified bandwidth to each port on which the rule is applied. However, as this feature is not yet integrated with the Compute scheduler, minimum bandwidth cannot be guaranteed.

It is also possible to combine several rules in one 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.

```
$ openstack network qos rule create --type bandwidth-limit \
    --max-kbps 50000 --max-burst-kbits 50000 --egress bandwidth-control
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| direction      | egress                               |
| id             | 0db48906-a762-4d32-8694-3f65214c34a6 |
| max_burst_kbps | 50000                                |
| max_kbps       | 50000                                |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+

$ openstack network qos rule create --type bandwidth-limit \
    --max-kbps 10000 --max-burst-kbits 10000 --ingress bandwidth-control
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| direction      | ingress                              |
| id             | faabef24-e23a-4fdf-8e92-f8cb66998834 |
| max_burst_kbps | 10000                                |
| max_kbps       | 10000                                |
| name           | None                                 |
| project_id     |                                      |
+----------------+--------------------------------------+

$ openstack network qos rule create --type minimum-bandwidth \
    --min-kbps 1000 --egress bandwidth-control
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| direction  | egress                               |
| id         | da858b32-44bc-43c9-b92b-cf6e2fa836ab |
| min_kbps   | 1000                                 |
| name       | None                                 |
| project_id |                                      |
+------------+--------------------------------------+

$ openstack network qos policy show bandwidth-control
+-------------------+--------------------------------------------------------------
↪-------------+
| Field             | Value                                                        ␣
↪             |
+-------------------+--------------------------------------------------------------
↪-------------+
| description       |                                                              ␣
↪             |
| id                | 8491547e-add1-4c6c-a50e-42121237256c                         ␣
↪             |
| is_default        | False                                                        ␣
↪             |
| name              | bandwidth-control                                            ␣
↪             |
| project_id        | 7cc5a84e415d48e69d2b06aa67b317d8                             ␣
↪             |
```

(continues on next page)

(continued from previous page)

```
| 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': 10000, u'direction': u'ingress',          ␣
↪                 |
|                   |   u'type': u'bandwidth_limit',                            ␣
↪                 |
|                   |   u'id': u'faabef24-e23a-4fdf-8e92-f8cb66998834',         ␣
↪                 |
|                   |   u'max_burst_kbps': 10000,                               ␣
↪                 |
|                   |   u'qos_policy_id': u'8491547e-add1-4c6c-a50e-
↪42121237256c'},      |
|                   |  {u'direction':                                           ␣
↪                 |
|                   |   u'egress', u'min_kbps': 1000, u'type': u'minimum_
↪bandwidth',      |
|                   |   u'id': u'da858b32-44bc-43c9-b92b-cf6e2fa836ab',         ␣
↪                 |
|                   |   u'qos_policy_id': u'8491547e-add1-4c6c-a50e-
↪42121237256c'}]      |
| shared            | False                                                     ␣
↪                 |
+-------------------+--------------------------------------------------------
↪--------------+
```

8.2.23 Quality of Service (QoS): Guaranteed Minimum Bandwidth

Most Networking Quality of Service (QoS) features are implemented solely by OpenStack Neutron and they are already documented in the *QoS configuration chapter of the Networking Guide*. Some more complex QoS features necessarily involve the scheduling of a cloud server, therefore their implementation is shared between OpenStack Nova, Neutron and Placement. As of the OpenStack Stein release the Guaranteed Minimum Bandwidth feature is like the latter.

This Networking Guide chapter does not aim to replace Nova or Placement documentation in any way, but it still hopes to give an overall OpenStack-level guide to understanding and configuring a deployment to use the Guaranteed Minimum Bandwidth feature.

A guarantee of minimum available bandwidth can be enforced on two levels:

- Scheduling a server on a compute host where the bandwidth is available. To be more precise: scheduling one or more ports of a server on a compute hosts physical network interfaces where the bandwidth is available.
- Queueing network packets on a physical network interface to provide the guaranteed bandwidth.

In short the enforcement has two levels:

- (server) placement and
- data plane.

Since the data plane enforcement is already documented in the *QoS chapter*, here we only document the placement-level enforcement.

Limitations

- A pre-created port with a `minimum-bandwidth` rule must be passed when booting a server (`openstack server create`). Passing a network with a minimum-bandwidth rule at boot is not supported because of technical reasons (in this case the port is created too late for Neutron to affect scheduling).
- Bandwidth guarantees for ports can only be requested on networks backed by a physical network (physnet).
- In Stein there is no support for networks with multiple physnets. However some simpler multi-segment networks are still supported:
  - Networks with multiple segments all having the same physnet name.
  - Networks with only one physnet segment (the other segments being tunneled segments).
- If you mix ports with and without bandwidth guarantees on the same physical interface then the ports without a guarantee may starve. Therefore mixing them is not recommended. Instead it is recommended to separate them by Nova host aggregates.
- Changing the guarantee of a QoS policy (adding/deleting a `minimum_bandwidth` rule, or changing the `min_kbps` field of a `minimum_bandwidth` rule) is only possible while the policy is not in effect. That is ports of the QoS policy are not yet used by Nova. Requests to change guarantees of in-use policies are rejected.
- The first data-plane-only Guaranteed Minimum Bandwidth implementation (for SR-IOV egress traffic) was released in the Newton release of Neutron. Because of the known lack of placement-level enforcement it was marked as best effort (5th bullet point). Since placement-level enforcement was not implemented bandwidth may have become overallocated and the system level resource inventory may have become inconsistent. Therefore for users of the data-plane-only implementation a migration/healing process is mandatory (see section *On Healing of Allocations*) to bring the system level resource inventory to a consistent state. Further operations that would reintroduce inconsistency (e.g. migrating a server with `minimum_bandwidth` QoS rule, but no resource allocation in Placement) are rejected now in a backward-incompatible way.
- The Guaranteed Minimum Bandwidth feature is not complete in the Stein release. Not all Nova server lifecycle operations can be executed on a server with bandwidth guarantees. Since Stein (Nova API microversion 2.72+) you can boot and delete a server with a guarantee and detach a port with a guarantee. Since Train you can also migrate and resize a server with a guarantee. Support for further server move operations (for example evacuate, live-migrate and unshelve after shelve-offload) is to be implemented later. For the definitive documentation please refer to the Port with Resource Request chapter of the OpenStack Compute API Guide.
- If an SR-IOV physical function is configured for use by the neutron-openvswitch-agent, and the same physical functions virtual functions are configured for use by the neutron-sriov-agent then the available bandwidth must be statically split between the corresponding resource providers by administrative choice. For example a 10 Gbps SR-IOV capable physical NIC could be treated as

two independent NICs - a 5 Gbps NIC (technically the physical function of the NIC) added to an Open vSwitch bridge, and another 5 Gbps NIC whose virtual functions can be handed out to servers by neutron-sriov-agent.

Placement pre-requisites

Placement must support microversion 1.29. This was first released in Rocky.

Nova pre-requisites

Nova must support microversion 2.72. This was first released in Stein.

Not all Nova virt drivers are supported, please refer to the Virt Driver Support section of the Nova Admin Guide.

Neutron pre-requisites

Neutron must support the following API extensions:

- `agent-resources-synced`
- `port-resource-request`
- `qos-bw-minimum-ingress`

These were all first released in Stein.

Supported drivers and agents

In release Stein the following agent-based ML2 mechanism drivers are supported:

- Open vSwitch (`openvswitch`) vnic_types: `normal`, `direct`
- SR-IOV (`sriovnicswitch`) vnic_types: `direct`, `macvtap`

neutron-server config

The `placement` service plugin synchronizes the agents resource provider information from neutron-server to Placement.

Since neutron-server talks to Placement you need to configure how neutron-server should find Placement and authenticate to it.

`/etc/neutron/neutron.conf` (on controller nodes):

```
[DEFAULT]
service_plugins = placement,...
auth_strategy = keystone

[placement]
auth_type = password
auth_url = https://controller/identity
```

(continues on next page)

(continued from previous page)

```
password = secret
project_domain_name = Default
project_name = service
user_domain_name = Default
username = placement
```

If a vnic_type is supported by default by multiple ML2 mechanism drivers (e.g. `vnic_type=direct` by both `openvswitch` and `sriovnicswitch`) and multiple agents resources are also meant to be tracked by Placement, then the admin must decide which driver to take ports of that vnic_type by prohibiting the vnic_type for the unwanted drivers. Use *ovs_driver.vnic_type_prohibit_list* in this case. Valid values are all the `supported_vnic_types` of the respective mechanism drivers.

`/etc/neutron/plugins/ml2/ml2_conf.ini` (on controller nodes):

```
[ovs_driver]
vnic_type_prohibit_list = direct

[sriov_driver]
#vnic_type_prohibit_list = direct
```

neutron-openvswitch-agent config

Set the agent configuration as the authentic source of the resources available. Set it on a per-bridge basis by *ovs.resource_provider_bandwidths*. The format is: `bridge:egress:ingress,...`. You may set only one direction and omit the other.

Note: `egress` / `ingress` is meant from the perspective of a cloud server. That is `egress` = cloud server upload, `ingress` = download.

Egress and ingress available bandwidth values are in `kilobit/sec (kbps)`.

If desired, resource provider inventory fields can be tweaked on a per-agent basis by setting *ovs.resource_provider_inventory_defaults*. Valid values are all the optional parameters of the update resource provider inventory call.

`/etc/neutron/plugins/ml2/ovs_agent.ini` (on compute and network nodes):

```
[ovs]
bridge_mappings = physnet0:br-physnet0,...
resource_provider_bandwidths = br-physnet0:10000000:10000000,...
#resource_provider_inventory_defaults = step_size:1000,...
```

neutron-sriov-agent config

The configuration of neutron-sriov-agent is analog to that of neutron-openvswitch-agent. However look out for:

- The different .ini section names as you can see below.
- That neutron-sriov-agent allows a physnet to be backed by multiple physical devices.
- Of course refer to SR-IOV physical functions instead of bridges in *sriov_nic.resource_provider_bandwidths*.

`/etc/neutron/plugins/ml2/sriov_agent.ini` (on compute nodes):

```
[sriov_nic]
physical_device_mappings = physnet0:ens5,physnet0:ens6,...
resource_provider_bandwidths = ens5:40000000:40000000,
↪ens6:40000000:40000000,...
#resource_provider_inventory_defaults = step_size:1000,...
```

Propagation of resource information

The flow of information is different for available and used resources.

The authentic source of available resources is neutron agent configuration - where the resources actually exist, as described in the agent configuration sections above. This information is propagated in the following chain: `neutron-l2-agent -> neutron-server -> Placement`.

From neutron agent to server the information is included in the `configurations` field of the agent heartbeat message sent on the message queue periodically.

```
# as admin
$ openstack network agent list --agent-type open-vswitch --host devstack0
+--------------------------------------+--------------------+-----------+--
↪-----------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host      |␣
↪Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-----------+--
↪-----------------+-------+-------+---------------------------+
| 5e57b85f-b017-419a-8745-9c406e149f9e | Open vSwitch agent | devstack0 |␣
↪None              | :-)   | UP    | neutron-openvswitch-agent |
+--------------------------------------+--------------------+-----------+--
↪-----------------+-------+-------+---------------------------+

# output shortened and pretty printed
# note: 'configurations' on the wire, but 'configuration' in the cli
$ openstack network agent show -f value -c configuration 5e57b85f-b017-
↪419a-8745-9c406e149f9e
{'bridge_mappings': {'physnet0': 'br-physnet0'},
 'resource_provider_bandwidths': {'br-physnet0': {'egress': 10000000,
                                                  'ingress': 10000000}},
 'resource_provider_inventory_defaults': {'allocation_ratio': 1.0,
                                          'min_unit': 1,
                                          'reserved': 0,
                                          'step_size': 1},
 ...
}
```

Re-reading the resource related subset of configuration on SIGHUP is not implemented. The agent must be restarted to pick up and send changed configuration.

Neutron-server propagates the information further to Placement for the resources of each agent via Placements HTTP REST API. To avoid overloading Placement this synchronization generally does not happen on every received heartbeat message. Instead the re-synchronization of the resources of one agent is triggered by:

- The creation of a network agent record (as queried by openstack network agent list). Please note that deleting an agent record and letting the next heartbeat to re-create it can be used to trigger synchronization without restarting an agent.
- The restart of that agent (technically start_flag being present in the heartbeat message).

Both of these can be used by an admin to force a re-sync if needed.

The success of a synchronization attempt from neutron-server to Placement is persisted into the relevant agents resources_synced attribute. For example:

```
# as admin
$ openstack network agent show -f value -c resources_synced 5e57b85f-b017-
↪419a-8745-9c406e149f9e
True
```

resources_synced may take the value True, False and None:

- None: No sync was attempted (normal for agents not reporting Placement-backed resources).
- True: The last sync attempt was completely successful.
- False: The last sync attempt was partially or utterly unsuccessful.

In case resources_synced is not True for an agent, neutron-server does try to re-sync on receiving every heartbeat message from that agent. Therefore it should be able to recover from transient errors of Neutron-Placement communication (e.g. Placement being started later than Neutron).

It is important to note that the restart of neutron-server does not trigger any kind of re-sync to Placement (to avoid an update storm).

As mentioned before, the information flow for resources requested and (if proper) allocated is different. It involves a conversation between Nova, Neutron and Placement.

1. Neutron exposes a ports resource needs in terms of resource classes and traits as the admin-only resource_request attribute of that port.
2. Nova reads this and incorporates it as a numbered request group into the cloud servers overall allocation candidate request to Placement.
3. Nova selects (schedules) and allocates one candidate returned by Placement.
4. Nova informs Neutron when binding the port of which physical network interface resource provider had been selected for the ports resource request in the binding:profile.allocation sub-attribute of that port.

For details please see slides 13-15 of a (pre-release) demo that was presented on the Berlin Summit in November 2018.

Sample usage

Physnets and QoS policies (together with their rules) are usually pre-created by a cloud admin:

```
# as admin

$ openstack network create net0 \
    --provider-network-type vlan \
    --provider-physical-network physnet0 \
    --provider-segment 100

$ openstack subnet create subnet0 \
    --network net0 \
    --subnet-range 10.0.4.0/24

$ openstack network qos policy create policy0

$ openstack network qos rule create policy0 \
    --type minimum-bandwidth \
    --min-kbps 1000000 \
    --egress

$ openstack network qos rule create policy0 \
    --type minimum-bandwidth \
    --min-kbps 1000000 \
    --ingress
```

Then a normal user can use the pre-created policy to create ports and boot servers with those ports:

```
# as an unprivileged user

# an ordinary soft-switched port: ``--vnic-type normal`` is the default
$ openstack port create port-normal-qos \
    --network net0 \
    --qos-policy policy0

# alternatively an SR-IOV port, unused in this example
$ openstack port create port-direct-qos \
    --network net0 \
    --vnic-type direct \
    --qos-policy policy0

$ openstack server create server0 \
    --flavor cirros256 \
    --image cirros-0.5.1-x86_64-disk \
    --port port-normal-qos
```

On Healing of Allocations

Since Placement carries a global view of a cloud deployments resources (what is available, what is used) it may in some conditions get out of sync with reality.

One important case is when the data-plane-only Minimum Guaranteed Bandwidth feature was used before Stein (first released in Newton). Since before Stein guarantees were not enforced during server placement the available resources may have become overallocated without notice. In this case Placements view and the reality of resource usage should be made consistent during/after an upgrade to Stein.

Another case stems from OpenStack not having distributed transactions to allocate resources provided by multiple OpenStack components (here Nova and Neutron). There are known race conditions in which Placements view may get out of sync with reality. The design knowingly minimizes the race condition windows, but there are known problems:

- If a QoS policy is modified after Nova read a ports `resource_request` but before the port is bound its state before the modification will be applied.
- If a bound port with a resource allocation is deleted. The ports allocation is leaked. https://bugs.launchpad.net/nova/+bug/1820588

Note: Deleting a bound port has no known use case. Please consider detaching the interface first by `openstack server remove port` instead.

Incorrect allocations may be fixed by:

- Moving the server, which will delete the wrong allocation and create the correct allocation as soon as move operations are implemented (not in Stein unfortunately). Moving servers fixes local overallocations.
- The need for an upgrade-helper allocation healing tool is being tracked in bug 1819923.
- Manually, by using openstack resource provider allocation set /delete.

Debugging

- Are all components running at least the Stein release?
- Is the `placement` service plugin enabled in neutron-server?
- Is `resource_provider_bandwidths` configured for the relevant neutron agent?
- Is `resource_provider_bandwidths` aligned with `bridge_mappings` or `physical_device_mappings`?
- Was the agent restarted since changing the configuration file?
- Is `resource_provider_bandwidths` reaching neutron-server?

```
# as admin
$ openstack network agent show ... | grep configurations
```

Please find an example in section *Propagation of resource information*.

- Did neutron-server successfully sync to Placement?

```
# as admin
$ openstack network agent show ... | grep resources_synced
```

Please find an example in section *Propagation of resource information*.

- Is the resource provider tree correct? Is the root a compute host? One level below the agents? Two levels below the physical network interfaces?

```
$ openstack --os-placement-api-version 1.17 resource provider list
+--------------------------------------+------------------------------------
↪-------+------------+--------------------------------------+-------------
↪-------------------------+
| uuid                                 | name                               ␣
↪       | generation | root_provider_uuid                   | parent_
↪provider_uuid                |
+--------------------------------------+------------------------------------
↪-------+------------+--------------------------------------+-------------
↪-------------------------+
| 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | devstack0                          ␣
↪       |          2 | 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | None        ␣
↪                         |
| 4a8a819d-61f9-5822-8c5c-3e9c7cb942d6 | devstack0:NIC Switch agent         ␣
↪       |          0 | 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | 3b36d91e-
↪bf60-460f-b1f8-3322dee5cdfd |
| 1c7e83f0-108d-5c35-ada7-7ebebbe43aad | devstack0:NIC Switch agent:ens5    ␣
↪       |          2 | 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | 4a8a819d-
↪61f9-5822-8c5c-3e9c7cb942d6 |
| 89ca1421-5117-5348-acab-6d0e2054239c | devstack0:Open vSwitch agent       ␣
↪       |          0 | 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | 3b36d91e-
↪bf60-460f-b1f8-3322dee5cdfd |
| f9c9ce07-679d-5d72-ac5f-31720811629a | devstack0:Open vSwitch agent:br-
↪physnet0 |          2 | 3b36d91e-bf60-460f-b1f8-3322dee5cdfd | 89ca1421-
↪5117-5348-acab-6d0e2054239c |
+--------------------------------------+------------------------------------
↪-------+------------+--------------------------------------+-------------
↪-------------------------+
```

- Does Placement have the expected traits?

```
# as admin
$ openstack --os-placement-api-version 1.17 trait list | awk '/CUSTOM_/ {␣
↪print $2 }' | sort
CUSTOM_PHYSNET_PHYSNET0
CUSTOM_VNIC_TYPE_DIRECT
CUSTOM_VNIC_TYPE_DIRECT_PHYSICAL
CUSTOM_VNIC_TYPE_MACVTAP
CUSTOM_VNIC_TYPE_NORMAL
```

- Do the physical network interface resource providers have the proper trait associations and inventories?

```
# as admin
$ openstack --os-placement-api-version 1.17 resource provider trait list␣
↪RP-UUID
$ openstack --os-placement-api-version 1.17 resource provider inventory␣
↪list RP-UUID
```

- Does the QoS policy have a `minimum-bandwidth` rule?
- Does the port have the proper policy?
- Does the port have a `resource_request`?

```
# as admin
$ openstack port show port-normal-qos | grep resource_request
```

- Was the server booted with a port (as opposed to a network)?
- Did nova allocate resources for the server in Placement?

```
# as admin
$ openstack --os-placement-api-version 1.17 resource provider allocation␣
↪show SERVER-UUID
```

- Does the allocation have a part on the expected physical network interface resource provider?

```
# as admin
$ openstack --os-placement-api-version 1.17 resource provider show --
↪allocations RP-UUID
```

- Did placement manage to produce an allocation candidate list to nova during scheduling?
- Did nova manage to schedule the server?
- Did nova tell neutron which physical network interface resource provider was allocated to satisfy the bandwidth request?

```
# as admin
$ openstack port show port-normal-qos | grep binding.profile.*allocation
```

- Did neutron manage to bind the port?

Links

- Pre-release feature demo presented on the Berlin Summit in November 2018
- Nova documentation on using a port with `resource_request`
  - API Guide
  - Admin Guide
- Neutron spec: QoS minimum bandwidth allocation in Placement API
  - on specs.openstack.org
  - on review.opendev.org
- Nova spec: Network Bandwidth resource provider
  - on specs.openstack.org
  - on review.opendev.org
- Relevant OpenStack Networking API references
  - https://docs.openstack.org/api-ref/network/v2/#agent-resources-synced-extension

  - https://docs.openstack.org/api-ref/network/v2/#port-resource-request
  - https://docs.openstack.org/api-ref/network/v2/#qos-minimum-bandwidth-rules
- Microversion histories
  - Compute 2.72
  - Placement 1.29
- Implementation
  - on review.opendev.org
- Known Bugs
  - Missing tool to heal allocations
  - Bandwidth resource is leaked

8.2.24 Role-Based Access Control (RBAC)

The Role-Based Access Control (RBAC) policy framework enables both operators and users to grant access to resources for specific projects.

Supported objects for sharing with specific projects

Currently, the access that can be granted using this feature is supported by:

- Regular port creation permissions on networks (since Liberty).
- Binding QoS policies permissions to networks or ports (since Mitaka).
- Attaching router gateways to networks (since Mitaka).
- Binding security groups to ports (since Stein).
- Assigning address scopes to subnet pools (since Ussuri).
- Assigning subnet pools to subnets (since Ussuri).

Sharing an object with specific projects

Sharing an object with a specific project is accomplished by creating a policy entry that permits the target project the `access_as_shared` action on that object.

Sharing a network with specific projects

Create a network to share:

```
$ openstack network create secret_network
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
```

(continues on next page)

(continued from previous page)

```
| availability_zones        |                                      |
| created_at                | 2017-01-25T20:16:40Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | f55961b9-3eb8-42eb-ac96-b97038b568de |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| mtu                       | 1450                                 |
| name                      | secret_network                       |
| port_security_enabled     | True                                 |
| project_id                | 61b7eba037fd41f29cfba757c010faff     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 9                                    |
| qos_policy_id             | None                                 |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2017-01-25T20:16:40Z                 |
+---------------------------+--------------------------------------+
```

Create the policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `b87b2fc13e0248a4a031d38e06dc191d`):

```
$ openstack network rbac create --target-project \
b87b2fc13e0248a4a031d38e06dc191d --action access_as_shared \
--type network f55961b9-3eb8-42eb-ac96-b97038b568de
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
| id                | f93efdbf-f1e0-41d2-b093-8328959d469e |
| name              | None                                 |
| object_id         | f55961b9-3eb8-42eb-ac96-b97038b568de |
| object_type       | network                              |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| target_project_id | b87b2fc13e0248a4a031d38e06dc191d     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the network. The `action` parameter specifies what the project is allowed to do. The `type` parameter says that the target object is a network. The final parameter is the ID of the network we are granting access to.

Project `b87b2fc13e0248a4a031d38e06dc191d` will now be able to see the network when running **openstack network list** and **openstack network show** and will also be able to create ports on that network. No other users (other than admins and the owner) will be able to see the network.

Note: Subnets inherit the RBAC policy entries of their network.

To remove access for that project, delete the policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete f93efdbf-f1e0-41d2-b093-8328959d469e
```

If that project has ports on the network, the server will prevent the policy from being deleted until the ports have been deleted:

```
$ openstack network rbac delete f93efdbf-f1e0-41d2-b093-8328959d469e
RBAC policy on object f93efdbf-f1e0-41d2-b093-8328959d469e
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a network with an arbitrary number of projects.

Sharing a QoS policy with specific projects

Create a QoS policy to share:

```
$ openstack network qos policy create secret_policy
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| id                | 1f730d69-1c45-4ade-a8f2-89070ac4f046 |
| name              | secret_policy                        |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| revision_number   | 1                                    |
| rules             | []                                   |
| shared            | False                                |
| tags              | []                                   |
+-------------------+--------------------------------------+
```

Create the RBAC policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `be98b82f8fdf46b696e9e01cebc33fd9`):

```
$ openstack network rbac create --target-project \
be98b82f8fdf46b696e9e01cebc33fd9 --action access_as_shared \
--type qos_policy 1f730d69-1c45-4ade-a8f2-89070ac4f046
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
| id                | 8828e38d-a0df-4c78-963b-e5f215d3d550 |
| name              | None                                 |
| object_id         | 1f730d69-1c45-4ade-a8f2-89070ac4f046 |
| object_type       | qos_policy                           |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| target_project_id | be98b82f8fdf46b696e9e01cebc33fd9     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the QoS policy. The `action` parameter specifies what the project is allowed to do. The `type` parameter says that the target object is a QoS policy. The final parameter is the ID of the QoS policy we are granting access to.

Project `be98b82f8fdf46b696e9e01cebc33fd9` will now be able to see the QoS policy when running **openstack network qos policy list** and **openstack network qos policy show** and will also be able to bind it to its ports or networks. No other users (other than admins and the owner) will be able to see the QoS policy.

To remove access for that project, delete the RBAC policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
```

If that project has ports or networks with the QoS policy applied to them, the server will not delete the RBAC policy until the QoS policy is no longer in use:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
RBAC policy on object 8828e38d-a0df-4c78-963b-e5f215d3d550
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a qos-policy with an arbitrary number of projects.

Sharing a security group with specific projects

Create a security group to share:

```
$ openstack security group create my_security_group
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2019-02-07T06:09:59Z                 |
| description       | my_security_group                    |
| id                | 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24 |
| location          | None                                 |
| name              | my_security_group                    |
| project_id        | 077e8f39d3db4c9e998d842b0503283a     |
| revision_number   | 1                                    |
| rules             | ...                                  |
| tags              | []                                   |
| updated_at        | 2019-02-07T06:09:59Z                 |
+-------------------+--------------------------------------+
```

Create the RBAC policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `32016615de5d43bb88de99e7f2e26a1e`):

```
$ openstack network rbac create --target-project \
32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
--type security_group 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
| id                | 8828e38d-a0df-4c78-963b-e5f215d3d550 |
| name              | None                                 |
| object_id         | 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24 |
| object_type       | security_group                       |
| project_id        | 077e8f39d3db4c9e998d842b0503283a     |
```

(continues on next page)

(continued from previous page)

```
| target_project_id | 32016615de5d43bb88de99e7f2e26a1e     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the security group. The `action` parameter specifies what the project is allowed to do. The `type` parameter says that the target object is a security group. The final parameter is the ID of the security group we are granting access to.

Project `32016615de5d43bb88de99e7f2e26a1e` will now be able to see the security group when running **openstack security group list** and **openstack security group show** and will also be able to bind it to its ports. No other users (other than admins and the owner) will be able to see the security group.

To remove access for that project, delete the RBAC policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
```

If that project has ports with the security group applied to them, the server will not delete the RBAC policy until the security group is no longer in use:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
RBAC policy on object 8828e38d-a0df-4c78-963b-e5f215d3d550
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a security-group with an arbitrary number of projects.

Sharing an address scope with specific projects

Create an address scope to share:

```
$ openstack address scope create my_address_scope
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| id                | c19cb654-3489-4160-9c82-8a3015483643 |
| ip_version        | 4                                    |
| location          | ...                                  |
| name              | my_address_scope                     |
| project_id        | 34304bc4f233470fa4a2448d153b6324     |
| shared            | False                                |
+-------------------+--------------------------------------+
```

Create the RBAC policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `32016615de5d43bb88de99e7f2e26a1e`):

```
$ openstack network rbac create --target-project \
32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
--type address_scope c19cb654-3489-4160-9c82-8a3015483643
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
```

(continues on next page)

(continued from previous page)

```
| id                | d54b1482-98c4-44aa-9115-ede80387ffe0 |
| location          | ...                                  |
| name              | None                                 |
| object_id         | c19cb654-3489-4160-9c82-8a3015483643 |
| object_type       | address_scope                        |
| project_id        | 34304bc4f233470fa4a2448d153b6324     |
| target_project_id | 32016615de5d43bb88de99e7f2e26a1e     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the address scope. The `action` parameter specifies what the project is allowed to do. The `type` parameter says that the target object is an address scope. The final parameter is the ID of the address scope we are granting access to.

Project `32016615de5d43bb88de99e7f2e26a1e` will now be able to see the address scope when running **openstack address scope list** and **openstack address scope show** and will also be able to assign it to its subnet pools. No other users (other than admins and the owner) will be able to see the address scope.

To remove access for that project, delete the RBAC policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete d54b1482-98c4-44aa-9115-ede80387ffe0
```

If that project has subnet pools with the address scope applied to them, the server will not delete the RBAC policy until the address scope is no longer in use:

```
$ openstack network rbac delete d54b1482-98c4-44aa-9115-ede80387ffe0
RBAC policy on object c19cb654-3489-4160-9c82-8a3015483643
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share an address scope with an arbitrary number of projects.

Sharing a subnet pool with specific projects

Create a subnet pool to share:

```
$ openstack subnet pool create my_subnetpool --pool-prefix 203.0.113.0/24
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | None                                 |
| created_at        | 2020-03-16T14:23:01Z                 |
| default_prefixlen | 8                                    |
| default_quota     | None                                 |
| description       |                                      |
| id                | 11f79287-bc17-46b2-bfd0-2562471eb631 |
| ip_version        | 4                                    |
| is_default        | False                                |
| location          | ...                                  |
| max_prefixlen     | 32                                   |
| min_prefixlen     | 8                                    |
| name              | my_subnetpool                        |
| project_id        | 290ccedbcf594ecc8e76eff06f964f7e     |
```

(continues on next page)

(continued from previous page)

```
| revision_number   | 0                                    |
| shared            | False                                |
| tags              |                                      |
| updated_at        | 2020-03-16T14:23:01Z                 |
+-------------------+--------------------------------------+
```

Create the RBAC policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `32016615de5d43bb88de99e7f2e26a1e`):

```
$ openstack network rbac create --target-project \
32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
--type subnetpool 11f79287-bc17-46b2-bfd0-2562471eb631
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
| id                | d54b1482-98c4-44aa-9115-ede80387ffe0 |
| location          | ...                                  |
| name              | None                                 |
| object_id         | 11f79287-bc17-46b2-bfd0-2562471eb631 |
| object_type       | subnetpool                           |
| project_id        | 290ccedbcf594ecc8e76eff06f964f7e     |
| target_project_id | 32016615de5d43bb88de99e7f2e26a1e     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the subnet pool. The `action` parameter specifies what the project is allowed to do. The `type` parameter says that the target object is a subnet pool. The final parameter is the ID of the subnet pool we are granting access to.

Project `32016615de5d43bb88de99e7f2e26a1e` will now be able to see the subnet pool when running **openstack subnet pool list** and **openstack subnet pool show** and will also be able to assign it to its subnets. No other users (other than admins and the owner) will be able to see the subnet pool.

To remove access for that project, delete the RBAC policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete d54b1482-98c4-44aa-9115-ede80387ffe0
```

If that project has subnets with the subnet pool applied to them, the server will not delete the RBAC policy until the subnet pool is no longer in use:

```
$ openstack network rbac delete d54b1482-98c4-44aa-9115-ede80387ffe0
RBAC policy on object 11f79287-bc17-46b2-bfd0-2562471eb631
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a subnet pool with an arbitrary number of projects.

How the shared flag relates to these entries

As introduced in other guide entries, neutron provides a means of making an object (`address-scope`, `network`, `qos-policy`, `security-group`, `subnetpool`) available to every project. This is accomplished using the `shared` flag on the supported object:

```
$ openstack network create global_network --share
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-01-25T20:32:06Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 84a7e627-573b-49da-af66-c9a65244f3ce |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| mtu                       | 1450                                 |
| name                      | global_network                       |
| port_security_enabled     | True                                 |
| project_id                | 61b7eba037fd41f29cfba757c010faff     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 7                                    |
| qos_policy_id             | None                                 |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2017-01-25T20:32:07Z                 |
+---------------------------+--------------------------------------+
```

This is the equivalent of creating a policy on the network that permits every project to perform the action `access_as_shared` on that network. Neutron treats them as the same thing, so the policy entry for that network should be visible using the **`openstack network rbac list`** command:

```
$ openstack network rbac list
+-------------------------------+-------------+--------------------------
↪----+
| ID                            | Object Type | Object ID                ␣
↪    |
+-------------------------------+-------------+--------------------------
↪----+
| 58a5ee31-2ad6-467d-           | qos_policy  | 1f730d69-1c45-4ade-      ␣
↪    |
| 8bb8-8c2ae3dd1382             |             | a8f2-89070ac4f046        ␣
↪    |
| 27efbd79-f384-4d89-9dfc-      | network     | 84a7e627-573b-49da-      ␣
↪    |
| 6c4a606ceec6                  |             | af66-c9a65244f3ce        ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
+-------------------------------+-------------+---------------------------
↪----+
```

Use the **openstack network rbac show** command to see the details:

```
$ openstack network rbac show 27efbd79-f384-4d89-9dfc-6c4a606ceec6
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_shared                     |
| id                | 27efbd79-f384-4d89-9dfc-6c4a606ceec6 |
| name              | None                                 |
| object_id         | 84a7e627-573b-49da-af66-c9a65244f3ce |
| object_type       | network                              |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| target_project_id | *                                    |
+-------------------+--------------------------------------+
```

The output shows that the entry allows the action `access_as_shared` on object `84a7e627-573b-49da-af66-c9a65244f3ce` of type `network` to target_tenant `*`, which is a wildcard that represents all projects.

Currently, the `shared` flag is just a mapping to the underlying RBAC policies for a network. Setting the flag to `True` on a network creates a wildcard RBAC entry. Setting it to `False` removes the wildcard entry.

When you run **openstack network list** or **openstack network show**, the `shared` flag is calculated by the server based on the calling project and the RBAC entries for each network. For QoS objects use **openstack network qos policy list** or **openstack network qos policy show** respectively. If there is a wildcard entry, the `shared` flag is always set to `True`. If there are only entries that share with specific projects, only the projects the object is shared to will see the flag as `True` and the rest will see the flag as `False`.

Allowing a network to be used as an external network

To make a network available as an external network for specific projects rather than all projects, use the `access_as_external` action.

1. Create a network that you want to be available as an external network:

```
$ openstack network create secret_external_network
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-01-25T20:36:59Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 802d4e9e-4649-43e6-9ee2-8d052a880cfb |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
```

(continues on next page)

(continued from previous page)

```
| mtu                       | 1450                                 |
| name                      | secret_external_network              |
| port_security_enabled     | True                                 |
| project_id                | 61b7eba037fd41f29cfba757c010faff     |
| proider:network_type      | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 21                                   |
| qos_policy_id             | None                                 |
| revision_number           | 3                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2017-01-25T20:36:59Z                 |
+---------------------------+--------------------------------------+
```

2. Create a policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is `838030a7bf3c4d04b4b054c0f0b2b17c`):

```
$ openstack network rbac create --target-project \
838030a7bf3c4d04b4b054c0f0b2b17c --action access_as_external \
--type network 802d4e9e-4649-43e6-9ee2-8d052a880cfb
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| action            | access_as_external                   |
| id                | afdd5b8d-b6f5-4a15-9817-5231434057be |
| name              | None                                 |
| object_id         | 802d4e9e-4649-43e6-9ee2-8d052a880cfb |
| object_type       | network                              |
| project_id        | 61b7eba037fd41f29cfba757c010faff     |
| target_project_id | 838030a7bf3c4d04b4b054c0f0b2b17c     |
+-------------------+--------------------------------------+
```

The `target-project` parameter specifies the project that requires access to the network. The `action` parameter specifies what the project is allowed to do. The `type` parameter indicates that the target object is a network. The final parameter is the ID of the network we are granting external access to.

Now project `838030a7bf3c4d04b4b054c0f0b2b17c` is able to see the network when running **openstack network list** and **openstack network show** and can attach router gateway ports to that network. No other users (other than admins and the owner) are able to see the network.

To remove access for that project, delete the policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete afdd5b8d-b6f5-4a15-9817-5231434057be
```

If that project has router gateway ports attached to that network, the server prevents the policy from being deleted until the ports have been deleted:

```
$ openstack network rbac delete afdd5b8d-b6f5-4a15-9817-5231434057be
RBAC policy on object afdd5b8d-b6f5-4a15-9817-5231434057be
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to make a network available as external to an arbitrary number of projects.

If a network is marked as external during creation, it now implicitly creates a wildcard RBAC policy granting everyone access to preserve previous behavior before this feature was added.

```
$ openstack network create global_external_network --external
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-01-25T20:41:44Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 72a257a2-a56e-4ac7-880f-94a4233abec6 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| mtu                       | 1450                                 |
| name                      | global_external_network              |
| port_security_enabled     | True                                 |
| project_id                | 61b7eba037fd41f29cfba757c010faff     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 69                                   |
| qos_policy_id             | None                                 |
| revision_number           | 4                                    |
| router:external           | External                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
| updated_at                | 2017-01-25T20:41:44Z                 |
+---------------------------+--------------------------------------+
```

In the output above the standard `router:external` attribute is `External` as expected. Now a wildcard policy is visible in the RBAC policy listings:

```
$ openstack network rbac list --long -c ID -c Action
+--------------------------------------+--------------------+
| ID                                   | Action             |
+--------------------------------------+--------------------+
| b694e541-bdca-480d-94ec-eda59ab7d71a | access_as_external |
+--------------------------------------+--------------------+
```

You can modify or delete this policy with the same constraints as any other RBAC `access_as_external` policy.

Preventing regular users from sharing objects with each other

The default `policy.json` file will not allow regular users to share objects with every other project using a wildcard; however, it will allow them to share objects with specific project IDs.

If an operator wants to prevent normal users from doing this, the `"create_rbac_policy":` entry in `policy.json` can be adjusted from `""` to `"rule:admin_only"`.

8.2.25 Routed provider networks

Note: Use of this feature requires the OpenStack client version 3.3 or newer.

Before routed provider networks, the Networking service could not present a multi-segment layer-3 network as a single entity. Thus, each operator typically chose one of the following architectures:

- Single large layer-2 network
- Multiple smaller layer-2 networks

Single large layer-2 networks become complex at scale and involve significant failure domains.

Multiple smaller layer-2 networks scale better and shrink failure domains, but leave network selection to the user. Without additional information, users cannot easily differentiate these networks.

A routed provider network enables a single provider network to represent multiple layer-2 networks (broadcast domains) or segments and enables the operator to present one network to users. However, the particular IP addresses available to an instance depend on the segment of the network available on the particular compute node. Neutron port could be associated with only one network segment, but there is an exception for OVN distributed services like OVN Metadata.

Similar to conventional networking, layer-2 (switching) handles transit of traffic between ports on the same segment and layer-3 (routing) handles transit of traffic between segments.

Each segment requires at least one subnet that explicitly belongs to that segment. The association between a segment and a subnet distinguishes a routed provider network from other types of networks. The Networking service enforces that either zero or all subnets on a particular network associate with a segment. For example, attempting to create a subnet without a segment on a network containing subnets with segments generates an error.

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. In the future, implementation of dynamic routing protocols may ease configuration of routed networks.

Prerequisites

Routed provider networks require additional prerequisites over conventional provider networks. We recommend using the following procedure:

1. Begin with segments. The Networking service defines a segment using the following components:

   - Unique physical network name
   - Segmentation type
   - Segmentation ID

   For example, `provider1`, `VLAN`, and `2016`. See the API reference for more information.

   Within a network, use a unique physical network name for each segment which enables reuse of the same segmentation details between subnets. For example, using the same VLAN ID across all segments of a particular provider network. Similar to conventional provider networks, the operator must provision the layer-2 physical network infrastructure accordingly.

2. Implement routing between segments.

   The Networking service does not provision routing among segments. The operator must implement routing among segments of a provider network. Each subnet on a segment must contain the gateway address of the router interface on that particular subnet. For example:

Segment	Version	Addresses	Gateway
segment1	4	203.0.113.0/24	203.0.113.1
segment1	6	fd00:203:0:113::/64	fd00:203:0:113::1
segment2	4	198.51.100.0/24	198.51.100.1
segment2	6	fd00:198:51:100::/64	fd00:198:51:100::1

3. Map segments to compute nodes.

   Routed provider networks imply that compute nodes reside on different segments. The operator must ensure that every compute host that is supposed to participate in a router provider network has direct connectivity to one of its segments.

Host	Rack	Physical Network
compute0001	rack 1	segment 1
compute0002	rack 1	segment 1
compute0101	rack 2	segment 2
compute0102	rack 2	segment 2
compute0102	rack 2	segment 2

4. Deploy DHCP agents.

   Unlike conventional provider networks, a DHCP agent cannot support more than one segment within a network. The operator must deploy at least one DHCP agent per segment. Consider deploying DHCP agents on compute nodes containing the segments rather than one or more network nodes to reduce node count.

Host	Rack	Physical Network
network0001	rack 1	segment 1
network0002	rack 2	segment 2

5. Configure communication of the Networking service with the Compute scheduler.

   An instance with an interface with an IPv4 address in a routed provider network must be placed by the Compute scheduler in a host that has access to a segment with available IPv4 addresses. To make this possible, the Networking service communicates to the Compute scheduler the inventory of IPv4 addresses associated with each segment of a routed provider network. The operator must configure the authentication credentials that the Networking service will use to communicate with the Compute schedulers placement API. Please see below an example configuration.

   Note: Coordination between the Networking service and the Compute scheduler is not necessary for IPv6 subnets as a consequence of their large address spaces.

   Note: The coordination between the Networking service and the Compute scheduler requires the following minimum API micro-versions.

   - Compute service API: 2.41
   - Placement API: 1.1

Example configuration

Controller node

1. Enable the segments service plug-in by appending `segments` to the list of `service_plugins` in the `neutron.conf` file on all nodes running the `neutron-server` service:

```
[DEFAULT]
# ...
service_plugins = ...,segments
```

2. Add a `placement` section to the `neutron.conf` file with authentication credentials for the Compute service placement API:

```
[placement]
www_authenticate_uri = http://192.0.2.72/identity
project_domain_name = Default
project_name = service
user_domain_name = Default
password = apassword
username = nova
auth_url = http://192.0.2.72/identity_admin
auth_type = password
region_name = RegionOne
```

3. Restart the `neutron-server` service.

Network or compute nodes

- Configure the layer-2 agent on each node to map one or more segments to the appropriate physical network bridge or interface and restart the agent.

Create a routed provider network

The following steps create a routed provider network with two segments. Each segment contains one IPv4 subnet and one IPv6 subnet.

1. Source the administrative project credentials.
2. Create a VLAN provider network which includes a default segment. In this example, the network uses the `provider1` physical network with VLAN ID 2016.

```
$ openstack network create --share --provider-physical-network␣
↪provider1 \
  --provider-network-type vlan --provider-segment 2016 multisegment1
+---------------------------+--------------------------------------+
| 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  | 2016                                 |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
+---------------------------+--------------------------------------+
```

3. Rename the default segment to `segment1`.

```
$ openstack network segment list --network multisegment1
+--------------------------------------+----------+--------------------
↪------------------+--------------+---------+
| ID                                   | Name     | Network           ␣
↪                  | Network Type | Segment |
+--------------------------------------+----------+--------------------
↪------------------+--------------+---------+
| 43e16869-ad31-48e4-87ce-acf756709e18 | None     | 6ab19caa-dda9-
↪4b3d-abc4-5b8f435b98d9 | vlan         |    2016 |
+--------------------------------------+----------+--------------------
↪------------------+--------------+---------+
```

```
$ openstack network segment set --name segment1 43e16869-ad31-48e4-
↪87ce-acf756709e18
```

(continues on next page)

(continued from previous page)

Note: This command provides no output.

4. Create a second segment on the provider network. In this example, the segment uses the `provider2` physical network with VLAN ID 2017.

```
$ openstack network segment create --physical-network provider2 \
  --network-type vlan --segment 2017 --network multisegment1 segment2
+------------------+--------------------------------------+
| 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  | 2017                                 |
| tags             | []                                   |
+------------------+--------------------------------------+
```

5. Verify that the network contains the `segment1` and `segment2` segments.

```
$ openstack network segment list --network multisegment1
+--------------------------------------+----------+-------------------
↪-------------------+--------------+---------+
| ID                                   | Name     | Network           ␣
↪                   | Network Type | Segment |
+--------------------------------------+----------+-------------------
↪-------------------+--------------+---------+
| 053b7925-9a89-4489-9992-e164c8cc8763 | segment2 | 6ab19caa-dda9-
↪4b3d-abc4-5b8f435b98d9 | vlan         |    2017 |
| 43e16869-ad31-48e4-87ce-acf756709e18 | segment1 | 6ab19caa-dda9-
↪4b3d-abc4-5b8f435b98d9 | vlan         |    2016 |
+--------------------------------------+----------+-------------------
↪-------------------+--------------+---------+
```

6. Create subnets on the `segment1` segment. In this example, the IPv4 subnet uses 203.0.113.0/24 and the IPv6 subnet uses fd00:203:0:113::/64.

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 4 --subnet-range 203.0.113.0/24 \
  multisegment1-segment1-v4
+-------------------+--------------------------------------+
| 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                          |
```

(continues on next page)

(continued from previous page)

```
| 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              | []                                   |
+-------------------+--------------------------------------+

$ 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
+-------------------+-----------------------------------------------
↪------+
| 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.

7. Create subnets on the `segment2` segment. In this example, the IPv4 subnet uses 198.51.100.0/24 and the IPv6 subnet uses fd00:198:51:100::/64.

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 4 --subnet-range 198.51.100.0/24 \
  multisegment1-segment2-v4
+-------------------+--------------------------------------+
| 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              | []                                   |
+-------------------+--------------------------------------+

$ 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
+-------------------+------------------------------------------------------
↪--------+
| 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              | []                                                   ␣
↪        |
+-------------------+------------------------------------------------------
↪--------+
```

(continues on next page)

(continued from previous page)

8. Verify that each IPv4 subnet associates with at least one DHCP agent.

```
$ openstack network agent list --agent-type dhcp --network␣
↪multisegment1
+--------------------------------------+------------+-------------+---
↪----------------+-------+-------+--------------------+
| 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 |
+--------------------------------------+------------+-------------+---
↪----------------+-------+-------+--------------------+
```

9. Verify that inventories were created for each segment IPv4 subnet in the Compute service placement API (for the sake of brevity, only one of the segments is shown in this example).

```
$ SEGMENT_ID=053b7925-9a89-4489-9992-e164c8cc8763
$ openstack resource provider inventory list $SEGMENT_ID
+----------------+------------------+----------+----------+-----------
↪+----------+-------+
| resource_class | allocation_ratio | max_unit | reserved | step_size␣
↪| min_unit | total |
+----------------+------------------+----------+----------+-----------
↪+----------+-------+
| IPV4_ADDRESS   |              1.0 |        1 |        2 |         1␣
↪|        1 |    30 |
+----------------+------------------+----------+----------+-----------
↪+----------+-------+
```

10. Verify that host aggregates were created for each segment in the Compute service (for the sake of brevity, only one of the segments is shown in this example).

```
$ openstack aggregate list
+----+---------------------------------------------------------+------
↪-------------+
| Id | Name                                                    |␣
↪Availability Zone |
+----+---------------------------------------------------------+------
↪-------------+
| 10 | Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763 | None␣
↪             |
+----+---------------------------------------------------------+------
↪-------------+
```

11. 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:

```
$ openstack port create --network multisegment1 port1
+-----------------------+--------------------------------------+
| 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                  | []                                   |
+-----------------------+--------------------------------------+
```

Migrating non-routed networks to routed

Migration of existing non-routed networks is only possible if there is only one segment and one subnet on the network. To migrate a candidate network, update the subnet and set `id` of the existing network segment as `segment_id`.

Note: In the case where there are multiple subnets or segments it is not possible to safely migrate. The reason for this is that in non-routed networks addresses from the subnets allocation pools are assigned to ports without considering to which network segment the port is bound.

Example

The following steps migrate an existing non-routed network with one subnet and one segment to a routed one.

1. Source the administrative project credentials.
2. Get the `id` of the current network segment on the network that is being migrated.

```
$ openstack network segment list --network my_network
+--------------------------------------+------+--------------------------------------
↪---------------+--------------+---------+
| ID                                   | Name | Network                              ␣
↪               | Network Type | Segment |
+--------------------------------------+------+--------------------------------------
↪---------------+--------------+---------+
| 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 | None | 45e84575-2918-471c-
↪95c0-018b961a2984 | flat         | None    |
+--------------------------------------+------+--------------------------------------
↪---------------+--------------+---------+
```

(continues on next page)

(continued from previous page)

3. Get the `id` or `name` of the current subnet on the network.

```
$ openstack subnet list --network my_network
+--------------------------------------+-----------+------------------
↪-------------------+---------------+
| ID                                   | Name      | Network          ␣
↪                   | Subnet        |
+--------------------------------------+-----------+------------------
↪-------------------+---------------+
| 71d931d2-0328-46ae-93bc-126caf794307 | my_subnet | 45e84575-2918-
↪471c-95c0-018b961a2984 | 172.24.4.0/24 |
+--------------------------------------+-----------+------------------
↪-------------------+---------------+
```

4. Verify the current `segment_id` of the subnet is `None`.

```
$ openstack subnet show my_subnet --c segment_id
+------------+-------+
| Field      | Value |
+------------+-------+
| segment_id | None  |
+------------+-------+
```

5. Update the `segment_id` of the subnet.

```
$ openstack subnet set --network-segment 81e5453d-4c9f-43a5-8ddf-
↪feaf3937e8c7 my_subnet
```

6. Verify that the subnet is now associated with the desired network segment.

```
$ openstack subnet show my_subnet --c segment_id
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| segment_id | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 |
+------------+--------------------------------------+
```

8.2.26 Service function chaining

Service function chain (SFC) essentially refers to the software-defined networking (SDN) version of policy-based routing (PBR). In many cases, SFC involves security, although it can include a variety of other features.

Fundamentally, SFC routes packets through one or more service functions instead of conventional routing that routes packets using destination IP address. Service functions essentially emulate a series of physical network devices with cables linking them together.

A basic example of SFC involves routing packets from one location to another through a firewall that lacks a next hop IP address from a conventional routing perspective. A more complex example involves an ordered series of service functions, each implemented using multiple instances (VMs). Packets must flow through one instance and a hashing algorithm distributes flows across multiple instances at each hop.

Architecture

All OpenStack Networking services and OpenStack Compute instances connect to a virtual network via ports making it possible to create a traffic steering model for service chaining using only ports. Including these ports in a port chain enables steering of traffic through one or more instances providing service functions.

A port chain, or service function path, consists of the following:

- A set of ports that define the sequence of service functions.
- A set of flow classifiers that specify the classified traffic flows entering the chain.

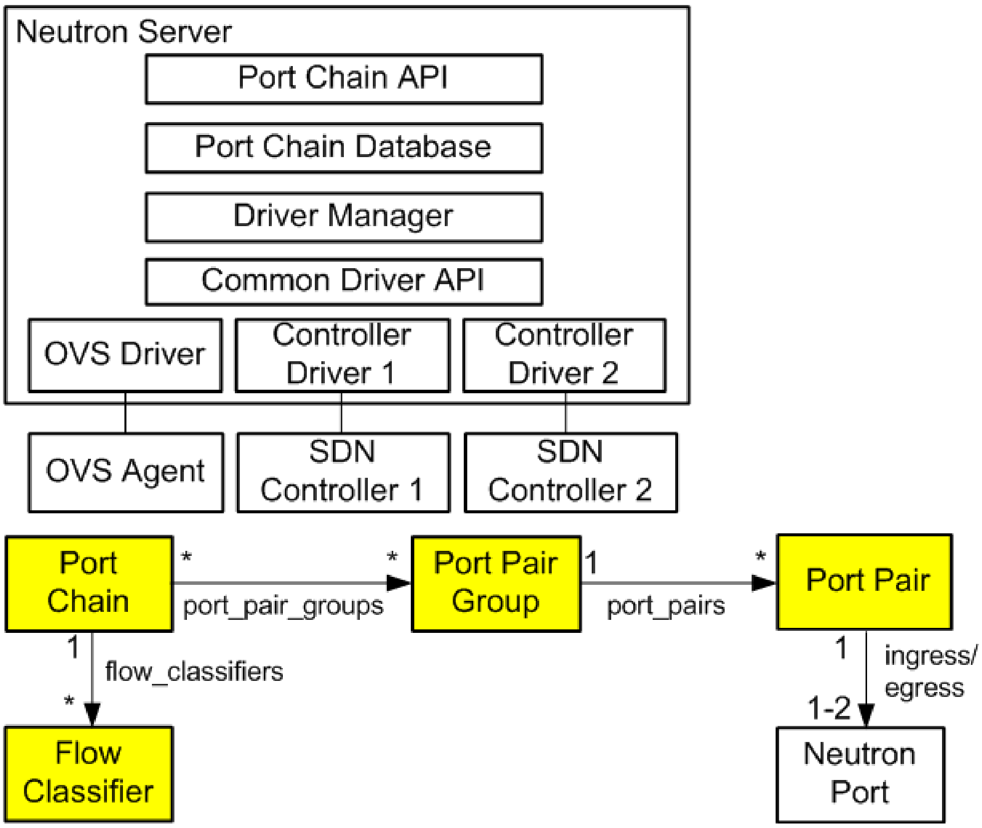
If a service function involves a pair of ports, the first port acts as the ingress port of the service function and the second port acts as the egress port. If both ports use the same value, they function as a single virtual bidirectional port.

A port chain is a unidirectional service chain. The first port acts as the head of the service function chain and the second port acts as the tail of the service function chain. A bidirectional service function chain consists of two unidirectional port chains.

A flow classifier can only belong to one port chain to prevent ambiguity as to which chain should handle packets in the flow. A check prevents such ambiguity. However, you can associate multiple flow classifiers with a port chain because multiple flows can request the same service function path.

Currently, SFC lacks support for multi-project service functions.

The port chain plug-in supports backing service providers including the OVS driver and a variety of SDN controller drivers. The common driver API enables different drivers to provide different implementations for the service chain path rendering.



See the networking-sfc documentation for more information.

Resources

Port chain

- `id` - Port chain ID
- `project_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `port_pair_groups` - List of port pair group IDs
- `flow_classifiers` - List of flow classifier IDs
- `chain_parameters` - Dictionary of chain parameters

A port chain consists of a sequence of port pair groups. Each port pair group is a hop in the port chain. A group of port pairs represents service functions providing equivalent functionality. For example, a group of firewall service functions.

A flow classifier identifies a flow. A port chain can contain multiple flow classifiers. Omitting the flow classifier effectively prevents steering of traffic through the port chain.

The `chain_parameters` attribute contains one or more parameters for the port chain. Currently, it only supports a correlation parameter that defaults to `mpls` for consistency with Open vSwitch (OVS) capabilities. Future values for the correlation parameter may include the network service header (NSH).

Port pair group

- `id` - Port pair group ID
- `project_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `port_pairs` - List of service function port pairs

A port pair group may contain one or more port pairs. Multiple port pairs enable load balancing/distribution over a set of functionally equivalent service functions.

Port pair

- `id` - Port pair ID
- `project_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `ingress` - Ingress port
- `egress` - Egress port
- `service_function_parameters` - Dictionary of service function parameters

A port pair represents a service function instance that includes an ingress and egress port. A service function containing a bidirectional port uses the same ingress and egress port.

The `service_function_parameters` attribute includes one or more parameters for the service function. Currently, it only supports a correlation parameter that determines association of a packet with a chain. This parameter defaults to `none` for legacy service functions that lack support for correlation such as the NSH. If set to `none`, the data plane implementation must provide service function proxy functionality.

Flow classifier

- `id` - Flow classifier ID
- `project_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `ethertype` - Ethertype (IPv4/IPv6)
- `protocol` - IP protocol
- `source_port_range_min` - Minimum source protocol port
- `source_port_range_max` - Maximum source protocol port
- `destination_port_range_min` - Minimum destination protocol port
- `destination_port_range_max` - Maximum destination protocol port
- `source_ip_prefix` - Source IP address or prefix
- `destination_ip_prefix` - Destination IP address or prefix
- `logical_source_port` - Source port
- `logical_destination_port` - Destination port
- `l7_parameters` - Dictionary of L7 parameters

A combination of the source attributes defines the source of the flow. A combination of the destination attributes defines the destination of the flow. The `l7_parameters` attribute is a place holder that may be used to support flow classification using layer 7 fields, such as a URL. If unspecified, the `logical_source_port` and `logical_destination_port` attributes default to `none`, the `ethertype` attribute defaults to `IPv4`, and all other attributes default to a wildcard value.

Operations

Create a port chain

The following example uses the `openstack` command-line interface (CLI) to create a port chain consisting of three service function instances to handle HTTP (TCP) traffic flows from 192.0.2.11:1000 to 198.51.100.11:80.

- Instance 1
  - Name: vm1

- Function: Firewall
- Port pair: [p1, p2]

- Instance 2
  - Name: vm2
  - Function: Firewall
  - Port pair: [p3, p4]
- Instance 3
  - Name: vm3
  - Function: Intrusion detection system (IDS)
  - Port pair: [p5, p6]

Note: The example network `net1` must exist before creating ports on it.

1. Source the credentials of the project that owns the `net1` network.
2. Create ports on network `net1` and record the UUID values.

```
$ openstack port create p1 --network net1
$ openstack port create p2 --network net1
$ openstack port create p3 --network net1
$ openstack port create p4 --network net1
$ openstack port create p5 --network net1
$ openstack port create p6 --network net1
```

3. Launch service function instance `vm1` using ports `p1` and `p2`, `vm2` using ports `p3` and `p4`, and `vm3` using ports `p5` and `p6`.

```
$ openstack server create --nic port-id=P1_ID --nic port-id=P2_ID vm1
$ openstack server create --nic port-id=P3_ID --nic port-id=P4_ID vm2
$ openstack server create --nic port-id=P5_ID --nic port-id=P6_ID vm3
```

Replace `P1_ID`, `P2_ID`, `P3_ID`, `P4_ID`, `P5_ID`, and `P6_ID` with the UUIDs of the respective ports.

Note: This command requires additional options to successfully launch an instance. See the CLI reference for more information.

Alternatively, you can launch each instance with one network interface and attach additional ports later.

4. Create flow classifier `FC1` that matches the appropriate packet headers.

```
$ openstack sfc flow classifier create \
  --description "HTTP traffic from 192.0.2.11 to 198.51.100.11" \
  --ethertype IPv4 \
  --source-ip-prefix 192.0.2.11/32 \
  --destination-ip-prefix 198.51.100.11/32 \
```

(continues on next page)

(continued from previous page)

```
  --protocol tcp \
  --source-port 1000:1000 \
  --destination-port 80:80 FC1
```

Note: When using the (default) OVS driver, the `--logical-source-port` parameter is also required

5. Create port pair `PP1` with ports `p1` and `p2`, `PP2` with ports `p3` and `p4`, and `PP3` with ports `p5` and `p6`.

```
$ openstack sfc port pair create \
  --description "Firewall SF instance 1" \
  --ingress p1 \
  --egress p2 PP1

$ openstack sfc port pair create \
  --description "Firewall SF instance 2" \
  --ingress p3 \
  --egress p4 PP2

$ openstack sfc port pair create \
  --description "IDS SF instance" \
  --ingress p5 \
  --egress p6 PP3
```

6. Create port pair group `PPG1` with port pair `PP1` and `PP2` and `PPG2` with port pair `PP3`.

```
$ openstack sfc port pair group create \
  --port-pair PP1 --port-pair PP2 PPG1
$ openstack sfc port pair group create \
  --port-pair PP3 PPG2
```

Note: You can repeat the `--port-pair` option for multiple port pairs of functionally equivalent service functions.

7. Create port chain `PC1` with port pair groups `PPG1` and `PPG2` and flow classifier `FC1`.

```
$ openstack sfc port chain create \
  --port-pair-group PPG1 --port-pair-group PPG2 \
  --flow-classifier FC1 PC1
```

Note: You can repeat the `--port-pair-group` option to specify additional port pair groups in the port chain. A port chain must contain at least one port pair group.

You can repeat the `--flow-classifier` option to specify multiple flow classifiers for a port chain. Each flow classifier identifies a flow.

Update a port chain or port pair group

- Use the **openstack sfc port chain set** command to dynamically add or remove port pair groups or flow classifiers on a port chain.
  - For example, add port pair group PPG3 to port chain PC1:

    ```
    $ openstack sfc port chain set \
      --port-pair-group PPG1 --port-pair-group PPG2 --port-pair-group␣
    ↪PPG3 \
      --flow-classifier FC1 PC1
    ```

  - For example, add flow classifier FC2 to port chain PC1:

    ```
    $ openstack sfc port chain set \
      --port-pair-group PPG1 --port-pair-group PPG2 \
      --flow-classifier FC1 --flow-classifier FC2 PC1
    ```

    SFC steers traffic matching the additional flow classifier to the port pair groups in the port chain.

- Use the **openstack sfc port pair group set** command to perform dynamic scale-out or scale-in operations by adding or removing port pairs on a port pair group.

  ```
  $ openstack sfc port pair group set \
    --port-pair PP1 --port-pair PP2 --port-pair PP4 PPG1
  ```

  SFC performs load balancing/distribution over the additional service functions in the port pair group.

8.2.27 SR-IOV

The purpose of this page is to describe how to enable SR-IOV functionality available in OpenStack (using OpenStack Networking). This functionality was first introduced in the OpenStack Juno release. This page intends to serve as a guide for how to configure OpenStack Networking and OpenStack Compute to create SR-IOV ports.

The basics

PCI-SIG Single Root I/O Virtualization and Sharing (SR-IOV) functionality is available in OpenStack since the Juno release. The SR-IOV specification defines a standardized mechanism to virtualize PCIe devices. This mechanism can virtualize a single PCIe Ethernet controller to appear as multiple PCIe devices. Each device can be directly assigned to an instance, bypassing the hypervisor and virtual switch layer. As a result, users are able to achieve low latency and near-line wire speed.

The following terms are used throughout this document:

Term	Definition
PF	Physical Function. The physical Ethernet controller that supports SR-IOV.
VF	Virtual Function. The virtual PCIe device created from a physical Ethernet controller.

SR-IOV agent

The SR-IOV agent allows you to set the admin state of ports, configure port security (enable and disable spoof checking), and configure QoS rate limiting and minimum bandwidth. You must include the SR-IOV agent on each compute node using SR-IOV ports.

Note: The SR-IOV agent was optional before Mitaka, and was not enabled by default before Liberty.

Note: The ability to control port security and QoS rate limit settings was added in Liberty.

Supported Ethernet controllers

The following manufacturers are known to work:

- Intel
- Mellanox
- QLogic
- Broadcom

For information on **Mellanox SR-IOV Ethernet ConnectX cards**, see:

- Mellanox: How To Configure SR-IOV VFs on ConnectX-4 or newer.
- Mellanox: How To Configure SR-IOV VFs on ConnectX-3/ConnectX-3 Pro.

For information on **QLogic SR-IOV Ethernet cards**, see:

- Users Guide OpenStack Deployment with SR-IOV Configuration.

For information on **Broadcom NetXtreme-E Series Ethernet cards**, see the Broadcom NetXtreme-C/NetXtreme-E User Guide.

For information on **Broadcom NetXtreme-S Series Ethernet cards**, see the Broadcom NetXtreme-S Product Page.

Using SR-IOV interfaces

In order to enable SR-IOV, the following steps are required:

1. Create Virtual Functions (Compute)
2. Configure allow list for PCI devices in nova-compute (Compute)
3. Configure neutron-server (Controller)
4. Configure nova-scheduler (Controller)
5. Enable neutron sriov-agent (Compute)

We recommend using VLAN provider networks for segregation. This way you can combine instances without SR-IOV ports and instances with SR-IOV ports on a single network.

Note: Throughout this guide, eth3 is used as the PF and physnet2 is used as the provider network configured as a VLAN range. These ports may vary in different environments.

Create Virtual Functions (Compute)

Create the VFs for the network interface that will be used for SR-IOV. We use eth3 as PF, which is also used as the interface for the VLAN provider network and has access to the private networks of all machines.

Note: The steps detail how to create VFs using Mellanox ConnectX-4 and newer/Intel SR-IOV Ethernet cards on an Intel system. Steps may differ for different hardware configurations.

1. Ensure SR-IOV and VT-d are enabled in BIOS.
2. Enable IOMMU in Linux by adding intel_iommu=on to the kernel parameters, for example, using GRUB.
3. On each compute node, create the VFs via the PCI SYS interface:

```
# echo '8' > /sys/class/net/eth3/device/sriov_numvfs
```

Note: On some PCI devices, observe that when changing the amount of VFs you receive the error Device or resource busy. In this case, you must first set sriov_numvfs to 0, then set it to your new value.

Note: A network interface could be used both for PCI passthrough, using the PF, and SR-IOV, using the VFs. If the PF is used, the VF number stored in the sriov_numvfs file is lost. If the PF is attached again to the operating system, the number of VFs assigned to this interface will be zero. To keep the number of VFs always assigned to this interface, modify the interfaces configuration file adding an ifup script command.

On Ubuntu, modify the /etc/network/interfaces file:

```
auto eth3
iface eth3 inet dhcp
pre-up echo '4' > /sys/class/net/eth3/device/sriov_numvfs
```

On RHEL and derivatives, modify the /sbin/ifup-local file:

```
#!/bin/sh
if [[ "$1" == "eth3" ]]
then
    echo '4' > /sys/class/net/eth3/device/sriov_numvfs
fi
```

> **Warning:** Alternatively, you can create VFs by passing the `max_vfs` to the kernel module of your network interface. However, the `max_vfs` parameter has been deprecated, so the PCI SYS interface is the preferred method.

You can determine the maximum number of VFs a PF can support:

```
# cat /sys/class/net/eth3/device/sriov_totalvfs
63
```

4. Verify that the VFs have been created and are in `up` state. For example:

```
# lspci | grep Ethernet
82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/
↪SFP+ Network Connection (rev 01)
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/
↪SFP+ Network Connection (rev 01)
82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:10.4 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:10.6 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:11.0 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:11.2 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:11.4 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
82:11.6 Ethernet controller: Intel Corporation 82599 Ethernet␣
↪Controller Virtual Function (rev 01)
```

```
# ip link show eth3
8: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP␣
↪mode DEFAULT qlen 1000
   link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
   vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 4 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 5 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 6 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
   vf 7 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

If the interfaces are down, set them to `up` before launching a guest, otherwise the instance will fail to spawn:

```
# ip link set eth3 up
```

5. Persist created VFs on reboot:

```
# echo "echo '7' > /sys/class/net/eth3/device/sriov_numvfs" >> /etc/
↪rc.local
```

Note: The suggested way of making PCI SYS settings persistent is through the `sysfsutils` tool. However, this is not available by default on many major distributions.

Configuring allow list for PCI devices nova-compute (Compute)

1. Configure which PCI devices the `nova-compute` service may use. Edit the `nova.conf` file:

```
[pci]
passthrough_whitelist = { "devname": "eth3", "physical_network":
↪"physnet2"}
```

This tells the Compute service that all VFs belonging to `eth3` are allowed to be passed through to instances and belong to the provider network `physnet2`.

Alternatively the `[pci] passthrough_whitelist` parameter also supports allowing devices by:

- PCI address: The address uses the same syntax as in `lspci` and an asterisk (`*`) can be used to match anything.

```
[pci]
passthrough_whitelist = { "address": "[[[[<domain>]:]<bus>]:][
↪<slot>][.[<function>]]", "physical_network": "physnet2" }
```

  For example, to match any domain, bus `0a`, slot `00`, and all functions:

```
[pci]
passthrough_whitelist = { "address": "*:0a:00.*", "physical_
↪network": "physnet2" }
```

- PCI `vendor_id` and `product_id` as displayed by the Linux utility `lspci`.

```
[pci]
passthrough_whitelist = { "vendor_id": "<id>", "product_id": "<id>
↪", "physical_network": "physnet2" }
```

If the device defined by the PCI address or `devname` corresponds to an SR-IOV PF, all VFs under the PF will match the entry. Multiple `[pci] passthrough_whitelist` entries per host are supported.

In order to enable SR-IOV to request trusted mode, the `[pci] passthrough_whitelist` parameter also supports a `trusted` tag.

Note: This capability is only supported starting with version 18.0.0 (Rocky) release of the compute service configured to use the libvirt driver.

Important: There are security implications of enabling trusted ports. The trusted VFs can be set into VF promiscuous mode which will enable it to receive unmatched and multicast traffic sent to the physical function.

For example, to allow users to request SR-IOV devices with trusted capabilities on device eth3:

```
[pci]
passthrough_whitelist = { "devname": "eth3", "physical_network":
↪"physnet2", "trusted":"true" }
```

The ports will have to be created with a binding profile to match the `trusted` tag, see *Launching instances with SR-IOV ports*.

2. Restart the `nova-compute` service for the changes to go into effect.

Configure neutron-server (Controller)

1. Add `sriovnicswitch` as mechanism driver. Edit the `ml2_conf.ini` file on each controller:

```
[ml2]
mechanism_drivers = openvswitch,sriovnicswitch
```

2. Ensure your physnet is configured for the chosen network type. Edit the `ml2_conf.ini` file on each controller:

```
[ml2_type_vlan]
network_vlan_ranges = physnet2
```

3. Add the `plugin.ini` file as a parameter to the `neutron-server` service. Edit the appropriate initialization script to configure the `neutron-server` service to load the plugin configuration file:

```
--config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugin.ini
```

4. Restart the `neutron-server` service.

Configure nova-scheduler (Controller)

1. On every controller node running the `nova-scheduler` service, add `PciPassthroughFilter` to `[filter_scheduler] enabled_filters` to enable this filter. Ensure `[filter_scheduler] available_filters` is set to the default of `nova.scheduler.filters.all_filters`:

```
[filter_scheduler]
enabled_filters = AvailabilityZoneFilter, ComputeFilter,␣
↪ComputeCapabilitiesFilter, ImagePropertiesFilter,␣
↪ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter,␣
↪PciPassthroughFilter
available_filters = nova.scheduler.filters.all_filters
```

2. Restart the `nova-scheduler` service.

Enable neutron-sriov-nic-agent (Compute)

1. Install the SR-IOV agent, if necessary.
2. Edit the `sriov_agent.ini` file on each compute node. For example:

```
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver

[sriov_nic]
physical_device_mappings = physnet2:eth3
exclude_devices =
```

Note: The `physical_device_mappings` parameter is not limited to be a 1-1 mapping between physical networks and NICs. This enables you to map the same physical network to more than one NIC. For example, if `physnet2` is connected to `eth3` and `eth4`, then `physnet2:eth3,physnet2:eth4` is a valid option.

The `exclude_devices` parameter is empty, therefore, all the VFs associated with eth3 may be configured by the agent. To exclude specific VFs, add them to the `exclude_devices` parameter as follows:

```
exclude_devices = eth1:0000:07:00.2;0000:07:00.3,eth2:0000:05:00.1;
↪0000:05:00.2
```

3. Ensure the SR-IOV agent runs successfully:

```
# neutron-sriov-nic-agent \
  --config-file /etc/neutron/neutron.conf \
  --config-file /etc/neutron/plugins/ml2/sriov_agent.ini
```

4. Enable the neutron SR-IOV agent service.

   If installing from source, you must configure a daemon file for the init system manually.

(Optional) FDB L2 agent extension

Forwarding DataBase (FDB) population is an L2 agent extension to OVS agent or Linux bridge. Its objective is to update the FDB table for existing instance using normal port. This enables communication between SR-IOV instances and normal instances. The use cases of the FDB population extension are:

- Direct port and normal port instances reside on the same compute node.
- Direct port instance that uses floating IP address and network node are located on the same host.

For additional information describing the problem, refer to: Virtual switching technologies and Linux bridge.

1. Edit the `ovs_agent.ini` or `linuxbridge_agent.ini` file on each compute node. For example:

```
[agent]
extensions = fdb
```

2. Add the FDB section and the `shared_physical_device_mappings` parameter. This parameter maps each physical port to its physical network name. Each physical network can be mapped to several ports:

```
[FDB]
shared_physical_device_mappings = physnet1:p1p1, physnet1:p1p2
```

Launching instances with SR-IOV ports

Once configuration is complete, you can launch instances with SR-IOV ports.

1. If it does not already exist, create a network and subnet for the chosen physnet. This is the network to which SR-IOV ports will be attached. For example:

```
$ openstack network create --provider-physical-network physnet2 \
    --provider-network-type vlan --provider-segment 1000 \
    sriov-net

$ openstack subnet create --network sriov-net \
    --subnet-pool shared-default-subnetpool-v4 \
    sriov-subnet
```

2. Get the `id` of the network where you want the SR-IOV port to be created:

```
$ net_id=$(openstack network show sriov-net -c id -f value)
```

3. Create the SR-IOV port. `vnic-type=direct` is used here, but other options include `normal`, `direct-physical`, and `macvtap`:

```
$ openstack port create --network $net_id --vnic-type direct \
    sriov-port
```

Alternatively, to request that the SR-IOV port accept trusted capabilities, the binding profile should be enhanced with the `trusted` tag.

```
$ openstack port create --network $net_id --vnic-type direct \
    --binding-profile trusted=true \
    sriov-port
```

4. Get the `id` of the created port:

```
$ port_id=$(openstack port show sriov-port -c id -f value)
```

5. Create the instance. Specify the SR-IOV port created in step two for the NIC:

```
$ openstack server create --flavor m1.large --image ubuntu_18.04 \
    --nic port-id=$port_id \
    test-sriov
```

Note: There are two ways to attach VFs to an instance. You can create an SR-IOV port or use the `pci_alias` in the Compute service. For more information about using `pci_alias`, refer to nova-api configuration.

SR-IOV with ConnectX-3/ConnectX-3 Pro Dual Port Ethernet

In contrast to Mellanox newer generation NICs, ConnectX-3 family network adapters expose a single PCI device (PF) in the system regardless of the number of physical ports. When the device is **dual port** and SR-IOV is enabled and configured we can observe some inconsistencies in linux networking subsystem.

Note: In the example below `enp4s0` represents PF net device associated with physical port 1 and `enp4s0d1` represents PF net device associated with physical port 2.

Example: A system with ConnectX-3 dual port device and a total of four VFs configured, two VFs assigned to port one and two VFs assigned to port two.

```
$ lspci | grep Mellanox
04:00.0 Network controller: Mellanox Technologies MT27520 Family [ConnectX-
↪3 Pro]
04:00.1 Network controller: Mellanox Technologies MT27500/MT27520 Family␣
↪[ConnectX-3/ConnectX-3 Pro Virtual Function]
04:00.2 Network controller: Mellanox Technologies MT27500/MT27520 Family␣
↪[ConnectX-3/ConnectX-3 Pro Virtual Function]
04:00.3 Network controller: Mellanox Technologies MT27500/MT27520 Family␣
↪[ConnectX-3/ConnectX-3 Pro Virtual Function]
04:00.4 Network controller: Mellanox Technologies MT27500/MT27520 Family␣
↪[ConnectX-3/ConnectX-3 Pro Virtual Function]
```

Four VFs are available in the system, however,

```
$ ip link show
31: enp4s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop master ovs-system␣
↪state DOWN mode DEFAULT group default qlen 1000
    link/ether f4:52:14:01:d9:e1 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 1 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 2 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 3 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
32: enp4s0d1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode␣
↪DEFAULT group default qlen 1000
    link/ether f4:52:14:01:d9:e2 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 1 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 2 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
    vf 3 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state␣
↪auto
```

ip command identifies each PF associated net device as having four VFs *each*.

Note: Mellanox `mlx4` driver allows *ip* commands to perform configuration of *all* VFs from either PF associated network devices.

To allow neutron SR-IOV agent to properly identify the VFs that belong to the correct PF network device (thus to the correct network port) Admin is required to provide the `exclude_devices` configuration option in `sriov_agent.ini`

Step 1: derive the VF to Port mapping from mlx4 driver configuration file: `/etc/modprobe.d/mlnx.conf` or `/etc/modprobe.d/mlx4.conf`

```
$ cat /etc/modprobe.d/mlnx.conf | grep "options mlx4_core"
options mlx4_core port_type_array=2,2 num_vfs=2,2,0 probe_vf=2,2,0 log_num_
↪mgm_entry_size=-1
```

Where:

`num_vfs=n1,n2,n3` - The driver will enable `n1` VFs on physical port 1, `n2` VFs on physical port 2 and `n3` dual port VFs (applies only to dual port HCA when all ports are Ethernet ports).

`probe_vfs=m1,m2,m3` - the driver probes `m1` single port VFs on physical port 1, `m2` single port VFs on physical port 2 (applies only if such a port exist) `m3` dual port VFs. Those VFs are attached to the hypervisor. (applies only if all ports are configured as Ethernet).

The VFs will be enumerated in the following order:

1. port 1 VFs
2. port 2 VFs
3. dual port VFs

In our example:

04:00.0 : PF associated to **both** ports.
04:00.1 : VF associated to port **1**
04:00.2 : VF associated to port **1**
04:00.3 : VF associated to port **2**
04:00.4 : VF associated to port **2**

Step 2: Update `exclude_devices` configuration option in `sriov_agent.ini` with the correct mapping

Each PF associated net device shall exclude the **other** ports VFs

```
[sriov_nic]
physical_device_mappings = physnet1:enp4s0,physnet2:enp4s0d1
exclude_devices = enp4s0:0000:04:00.3;0000:04:00.4,enp4s0d1:0000:04:00.1;
↪0000:04:00.2
```

SR-IOV with InfiniBand

The support for SR-IOV with InfiniBand allows a Virtual PCI device (VF) to be directly mapped to the guest, allowing higher performance and advanced features such as RDMA (remote direct memory access). To use this feature, you must:

1. Use InfiniBand enabled network adapters.
2. Run InfiniBand subnet managers to enable InfiniBand fabric.

   All InfiniBand networks must have a subnet manager running for the network to function. This is true even when doing a simple network of two machines with no switch and the cards are plugged in back-to-back. A subnet manager is required for the link on the cards to come up. It is possible to have more than one subnet manager. In this case, one of them will act as the primary, and any other will act as a backup that will take over when the primary subnet manager fails.
3. Install the `ebrctl` utility on the compute nodes.

   Check that `ebrctl` is listed somewhere in `/etc/nova/rootwrap.d/*`:

   ```
   $ grep 'ebrctl' /etc/nova/rootwrap.d/*
   ```

   If `ebrctl` does not appear in any of the rootwrap files, add this to the `/etc/nova/rootwrap.d/compute.filters` file in the `[Filters]` section.

   ```
   [Filters]
   ebrctl: CommandFilter, ebrctl, root
   ```

Known limitations

- When using Quality of Service (QoS), `max_burst_kbps` (burst over `max_kbps`) is not supported. In addition, `max_kbps` is rounded to Mbps.
- Security groups are not supported when using SR-IOV, thus, the firewall driver must be disabled. This can be done in the `neutron.conf` file.

  ```
  [securitygroup]
  firewall_driver = neutron.agent.firewall.NoopFirewallDriver
  ```

- SR-IOV is not integrated into the OpenStack Dashboard (horizon). Users must use the CLI or API to configure SR-IOV interfaces.
- Live migration support has been added to the Libvirt Nova virt-driver in the Train release for instances with neutron SR-IOV ports. Indirect mode SR-IOV interfaces (vnic-type: macvtap or virtio-forwarder) can now be migrated transparently to the guest. Direct mode SR-IOV interfaces (vnic-type: direct or direct-physical) are detached before the migration and reattached after the migration so this is not transparent to the guest. To avoid loss of network connectivy when live migrating with direct mode sriov the user should create a failover bond in the guest with a transparently live migration port type e.g. vnic-type normal or indirect mode SR-IOV.

  Note: SR-IOV features may require a specific NIC driver version, depending on the vendor. Intel NICs, for example, require ixgbe version 4.4.6 or greater, and ixgbevf version 3.2.2 or greater.

- Attaching SR-IOV ports to existing servers is supported starting with the Victoria release.

8.2.28 Subnet pools

Subnet pools have been made available since the Kilo release. It is a simple feature that has the potential to improve your workflow considerably. It also provides a building block from which other new features will be built in to OpenStack Networking.

To see if your cloud has this feature available, you can check that it is listed in the supported aliases. You can do this with the OpenStack client.

```
$ openstack extension list | grep subnet_allocation
| Subnet Allocation | subnet_allocation | Enables allocation of subnets
from a subnet pool                                                        ␣
↪                                                           |
```

Why you need them

Before Kilo, Networking had no automation around the addresses used to create a subnet. To create one, you had to come up with the addresses on your own without any help from the system. There are valid use cases for this but if you are interested in the following capabilities, then subnet pools might be for you.

First, would not it be nice if you could turn your pool of addresses over to Neutron to take care of? When you need to create a subnet, you just ask for addresses to be allocated from the pool. You do not have to worry about what you have already used and what addresses are in your pool. Subnet pools can do this.

Second, subnet pools can manage addresses across projects. The addresses are guaranteed not to overlap. If the addresses come from an externally routable pool then you know that all of the projects have addresses which are *routable* and unique. This can be useful in the following scenarios.

1. IPv6 since OpenStack Networking has no IPv6 floating IPs.
2. Routing directly to a project network from an external network.

How they work

A subnet pool manages a pool of addresses from which subnets can be allocated. It ensures that there is no overlap between any two subnets allocated from the same pool.

As a regular project in an OpenStack cloud, you can create a subnet pool of your own and use it to manage your own pool of addresses. This does not require any admin privileges. Your pool will not be visible to any other project.

If you are an admin, you can create a pool which can be accessed by any regular project. Being a shared resource, there is a quota mechanism to arbitrate access.

Quotas

Subnet pools have a quota system which is a little bit different than other quotas in Neutron. Other quotas in Neutron count discrete instances of an object against a quota. Each time you create something like a router, network, or a port, it uses one from your total quota.

With subnets, the resource is the IP address space. Some subnets take more of it than others. For example, 203.0.113.0/24 uses 256 addresses in one subnet but 198.51.100.224/28 uses only 16. If address space is limited, the quota system can encourage efficient use of the space.

With IPv4, the default_quota can be set to the number of absolute addresses any given project is allowed to consume from the pool. For example, with a quota of 128, I might get 203.0.113.128/26, 203.0.113.224/28, and still have room to allocate 48 more addresses in the future.

With IPv6 it is a little different. It is not practical to count individual addresses. To avoid ridiculously large numbers, the quota is expressed in the number of /64 subnets which can be allocated. For example, with a default_quota of 3, I might get 2001:db8:c18e:c05a::/64, 2001:db8:221c:8ef3::/64, and still have room to allocate one more prefix in the future.

Default subnet pools

Beginning with Mitaka, a subnet pool can be marked as the default. This is handled with a new extension.

```
$ openstack extension list | grep default-subnetpools
| Default Subnetpools | default-subnetpools | Provides ability to mark and use a subnetpool as the default |
```

An administrator can mark a pool as default. Only one pool from each address family can be marked default.

```
$ openstack subnet pool set --default 74348864-f8bf-4fc0-ab03-81229d189467
```

If there is a default, it can be requested by passing `--use-default-subnetpool` instead of `--subnet-pool SUBNETPOOL`.

Demo

If you have access to an OpenStack Kilo or later based neutron, you can play with this feature now. Give it a try. All of the following commands work equally as well with IPv6 addresses.

First, as admin, create a shared subnet pool:

```
$ openstack subnet pool create --share --pool-prefix 203.0.113.0/24 \
--default-prefix-length 26 demo-subnetpool4
+-------------------+--------------------------------+
| Field             | Value                          |
+-------------------+--------------------------------+
| address_scope_id  | None                           |
| created_at        | 2016-12-14T07:21:26Z           |
| default_prefixlen | 26                             |
| default_quota     | None                           |
| description       |                                |
```

(continues on next page)

(continued from previous page)

```
| headers           |                                 |
| id                | d3aefb76-2527-43d4-bc21-0ec253 |
|                   | 908545                          |
| ip_version        | 4                               |
| is_default        | False                           |
| max_prefixlen     | 32                              |
| min_prefixlen     | 8                               |
| name              | demo-subnetpool4                |
| prefixes          | 203.0.113.0/24                  |
| project_id        | cfd1889ac7d64ad891d4f20aef9f8d |
|                   | 7c                              |
| revision_number   | 1                               |
| shared            | True                            |
| tags              | []                              |
| updated_at        | 2016-12-14T07:21:26Z            |
+-------------------+---------------------------------+
```

The `default_prefix_length` defines the subnet size you will get if you do not specify `--prefix-length` when creating a subnet.

Do essentially the same thing for IPv6 and there are now two subnet pools. Regular projects can see them. (the output is trimmed a bit for display)

```
$ openstack subnet pool list
+------------------+------------------+--------------------+
| ID               | Name             | Prefixes           |
+------------------+------------------+--------------------+
| 2b7cc19f-0114-4e | demo-subnetpool  | 2001:db8:a583::/48 |
| f4-ad86-c1bb91fc |                  |                    |
| d1f9             |                  |                    |
| d3aefb76-2527-43 | demo-subnetpool4 | 203.0.113.0/24     |
| d4-bc21-0ec25390 |                  |                    |
| 8545             |                  |                    |
+------------------+------------------+--------------------+
```

Now, use them. It is easy to create a subnet from a pool:

```
$ openstack subnet create --ip-version 4 --subnet-pool \
demo-subnetpool4 --network demo-network1 demo-subnet1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.194-203.0.113.254          |
| cidr              | 203.0.113.192/26                     |
| created_at        | 2016-12-14T07:33:13Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 203.0.113.193                        |
| headers           |                                      |
| host_routes       |                                      |
| id                | 8d4fbae3-076c-4c08-b2dd-2d6175115a5e |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | demo-subnet1                         |
```

(continues on next page)

(continued from previous page)

```
| network_id        | 6b377f77-ce00-4ff6-8676-82343817470d |
| project_id        | cfd1889ac7d64ad891d4f20aef9f8d7c     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | d3aefb76-2527-43d4-bc21-0ec253908545 |
| tags              | []                                   |
| updated_at        | 2016-12-14T07:33:13Z                 |
+-------------------+--------------------------------------+
```

You can request a specific subnet from the pool. You need to specify a subnet that falls within the pools prefixes. If the subnet is not already allocated, the request succeeds. You can leave off the IP version because it is deduced from the subnet pool.

```
$ openstack subnet create --subnet-pool demo-subnetpool4 \
--network demo-network1 --subnet-range 203.0.113.128/26 subnet2
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 203.0.113.130-203.0.113.190          |
| cidr              | 203.0.113.128/26                     |
| created_at        | 2016-12-14T07:27:40Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 203.0.113.129                        |
| headers           |                                      |
| host_routes       |                                      |
| id                | d32814e3-cf46-4371-80dd-498a80badfba |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | subnet2                              |
| network_id        | 6b377f77-ce00-4ff6-8676-82343817470d |
| project_id        | cfd1889ac7d64ad891d4f20aef9f8d7c     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | d3aefb76-2527-43d4-bc21-0ec253908545 |
| tags              | []                                   |
| updated_at        | 2016-12-14T07:27:40Z                 |
+-------------------+--------------------------------------+
```

If the pool becomes exhausted, load some more prefixes:

```
$ openstack subnet pool set --pool-prefix \
198.51.100.0/24 demo-subnetpool4
$ openstack subnet pool show demo-subnetpool4
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| address_scope_id  | None                                 |
| created_at        | 2016-12-14T07:21:26Z                 |
| default_prefixlen | 26                                   |
| default_quota     | None                                 |
| description       |                                      |
| id                | d3aefb76-2527-43d4-bc21-0ec253908545 |
```

(continues on next page)

(continued from previous page)

```
| ip_version          | 4                                    |
| is_default          | False                                |
| max_prefixlen       | 32                                   |
| min_prefixlen       | 8                                    |
| name                | demo-subnetpool4                     |
| prefixes            | 198.51.100.0/24, 203.0.113.0/24      |
| project_id          | cfd1889ac7d64ad891d4f20aef9f8d7c     |
| revision_number     | 2                                    |
| shared              | True                                 |
| tags                | []                                   |
| updated_at          | 2016-12-14T07:30:32Z                 |
+---------------------+--------------------------------------+
```

8.2.29 Subnet onboard

The subnet onboard feature allows you to take existing subnets that have been created outside of a subnet pool and move them into an existing subnet pool. This enables you to begin using subnet pools and address scopes if you havent allocated existing subnets from subnet pools. It also allows you to move individual subnets between subnet pools, and by extension, move them between address scopes.

How it works

One of the fundamental constraints of subnet pools is that all subnets of the same address family (IPv4, IPv6) on a network must be allocated from the same subnet pool. Because of this constraint, subnets must be moved, or onboarded, into a subnet pool as a group at the network level rather than being handled individually. As such, the onboarding of subnets requires users to supply the UUID of the network the subnet(s) to onboard are associated with, and the UUID of the target subnet pool to perform the operation.

Does my environment support subnet onboard?

To test that subnet onboard is supported in your environment, execute the following command:

```
$ openstack extension list --network -c Alias -c Description | grep subnet_
↪onboard
| subnet_onboard | Provides support for onboarding subnets into subnet␣
↪pools
```

Support for subnet onboard exists in the ML2 plugin as of the Stein release. If you require subnet onboard but your current environment does not support it, consider upgrading to a release that supports subnet onboard. When using third-party plugins with neutron, check with the supplier of the plugin regarding support for subnet onboard.

Demo

Suppose an administrator has an existing provider network in their environment that was created without allocating its subnets from a subnet pool.

```
$ openstack network list
+--------------------------------------+----------------+------------------
↪--------------------+
| ID                                   | Name           | Subnets          ␣
↪                    |
+--------------------------------------+----------------+------------------
↪--------------------+
| f643a4f5-f8d3-4325-b1fe-6061a9af0f07 | provider-net-1 | 5153cab7-7ab6-
↪4956-8466-39aa85dccc9a |
+--------------------------------------+----------------+------------------
↪--------------------+

$ openstack subnet show 5153cab7-7ab6-4956-8466-39aa85dccc9a
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.0.2-192.168.7.254            |
| cidr              | 192.168.0.0/21                       |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.0.1                          |
| host_routes       |                                      |
| id                | 5153cab7-7ab6-4956-8466-39aa85dccc9a |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| network_id        | f643a4f5-f8d3-4325-b1fe-6061a9af0f07 |
| prefix_length     | None                                 |
| project_id        | 7b80998e5e044cee91c1cdb2e9c63afd     |
| revision_number   | 0                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| tags              |                                      |
| updated_at        | 2019-03-13T18:24:37Z                 |
+-------------------+--------------------------------------+
```

The administrator has created a subnet pool named `routable-prefixes` and wants to onboard the subnets associated with network `provider-net-1`. The administrator now wants to manage the address space for provider networks using a subnet pool, but doesnt have the prefixes used by these provider networks under the management of a subnet pool or address scope.

```
$ openstack subnet pool list
+--------------------------------------+-------------------+--------------+
| ID                                   | Name              | Prefixes     |
+--------------------------------------+-------------------+--------------+
| d05e9f61-248c-43f2-98f4-5142570127e1 | routable-prefixes | 10.10.0.0/16 |
+--------------------------------------+-------------------+--------------+
```

```
$ openstack subnet pool show routable-prefixes
+-------------------+--------------------------------+
| Field             | Value                          |
+-------------------+--------------------------------+
| address_scope_id  | None                           |
| created_at        | 2019-03-102T05:45:01Z          |
| default_prefixlen | 26                             |
| default_quota     | None                           |
| description       | Routable prefixes for projects |
| headers           |                                |
| id                | d3aefb76-2527-43d4-bc21-0ec253 |
|                   | 908545                         |
| ip_version        | 4                              |
| is_default        | False                          |
| max_prefixlen     | 32                             |
| min_prefixlen     | 8                              |
| name              | routable-prefixes              |
| prefixes          | 10.10.0.0/16                   |
| project_id        | cfd1889ac7d64ad891d4f20aef9f8d |
|                   | 7c                             |
| revision_number   | 1                              |
| shared            | True                           |
| tags              | []                             |
| updated_at        | 2019-03-10T05:45:01Z           |
+-------------------+--------------------------------+
```

The administrator can use the following command to bring these subnets under the management of a subnet pool:

```
$ openstack network onboard subnets provider-net-1 routable-prefixes
```

The subnets on `provider-net-1` should now all have their subnetpool_id updated to match the UUID of the `routable-prefixes` subnet pool:

```
$ openstack subnet show 5153cab7-7ab6-4956-8466-39aa85dccc9a
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.0.2-192.168.7.254            |
| cidr              | 192.168.0.0/21                       |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.0.1                          |
| host_routes       |                                      |
| id                | 5153cab7-7ab6-4956-8466-39aa85dccc9a |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| network_id        | f643a4f5-f8d3-4325-b1fe-6061a9af0f07 |
| prefix_length     | None                                 |
| project_id        | 7b80998e5e044cee91c1cdb2e9c63afd     |
| revision_number   | 0                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | d3aefb76-2527-43d4-bc21-0ec253908545 |
```

(continues on next page)

(continued from previous page)

```
| updated_at         | 2019-03-13T18:24:37Z                 |
+-------------------+--------------------------------------+
```

The subnet pool will also now show the onboarded prefix(es) in its prefix list:

```
$ openstack subnet pool show routable-prefixes
+-------------------+--------------------------------+
| Field             | Value                          |
+-------------------+--------------------------------+
| address_scope_id  | None                           |
| created_at        | 2019-03-102T05:45:01Z          |
| default_prefixlen | 26                             |
| default_quota     | None                           |
| description       | Routable prefixes for projects |
| headers           |                                |
| id                | d3aefb76-2527-43d4-bc21-0ec253 |
|                   | 908545                         |
| ip_version        | 4                              |
| is_default        | False                          |
| max_prefixlen     | 32                             |
| min_prefixlen     | 8                              |
| name              | routable-prefixes              |
| prefixes          | 10.10.0.0/16, 192.168.0.0/21   |
| project_id        | cfd1889ac7d64ad891d4f20aef9f8d |
|                   | 7c                             |
| revision_number   | 1                              |
| shared            | True                           |
| tags              | []                             |
| updated_at        | 2019-03-12T13:11:037Z          |
+-------------------+--------------------------------+
```

8.2.30 Service subnets

Service subnets enable operators to define valid port types for each subnet on a network without limiting networks to one subnet or manually creating ports with a specific subnet ID. Using this feature, operators can ensure that ports for instances and router interfaces, for example, always use different subnets.

Operation

Define one or more service types for one or more subnets on a particular network. Each service type must correspond to a valid device owner within the port model in order for it to be used.

During IP allocation, the *IPAM* driver returns an address from a subnet with a service type matching the port device owner. If no subnets match, or all matching subnets lack available IP addresses, the IPAM driver attempts to use a subnet without any service types to preserve compatibility. If all subnets on a network have a service type, the IPAM driver cannot preserve compatibility. However, this feature enables strict IP allocation from subnets with a matching device owner. If multiple subnets contain the same service type, or a subnet without a service type exists, the IPAM driver selects the first subnet with a matching service type. For example, a floating IP agent gateway port uses the following selection process:

- `network:floatingip_agent_gateway`

- None

Note: Ports with the device owner `network:dhcp` are exempt from the above IPAM logic for subnets with `dhcp_enabled` set to `True`. This preserves the existing automatic DHCP port creation behaviour for DHCP-enabled subnets.

Creating or updating a port with a specific subnet skips this selection process and explicitly uses the given subnet.

Usage

Note: Creating a subnet with a service type requires administrative privileges.

Example 1 - Proof-of-concept

This following example is not typical of an actual deployment. It is shown to allow users to experiment with configuring service subnets.

1. Create a network.

```
$ openstack network create demo-net1
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| description               |                                      |
| headers                   |                                      |
| id                        | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| mtu                       | 1450                                 |
| name                      | demo-net1                            |
| port_security_enabled     | True                                 |
| project_id                | a3db43cd0f224242a847ab84d091217d     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 110                                  |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      | []                                   |
+---------------------------+--------------------------------------+
```

2. Create a subnet on the network with one or more service types. For example, the `compute:nova` service type enables instances to use this subnet.

```
$ openstack subnet create demo-subnet1 --subnet-range 192.0.2.0/24 \
  --service-type 'compute:nova' --network demo-net1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| id                | 6e38b23f-0b27-4e3c-8e69-fd23a3df1935 |
| ip_version        | 4                                    |
| cidr              | 192.0.2.0/24                         |
| name              | demo-subnet1                         |
| network_id        | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| revision_number   | 1                                    |
| service_types     | ['compute:nova']                     |
| tags              | []                                   |
| tenant_id         | a8b3054cc1214f18b1186b291525650f     |
+-------------------+--------------------------------------+
```

3. Optionally, create another subnet on the network with a different service type. For example, the `compute:foo` arbitrary service type.

```
$ openstack subnet create demo-subnet2 --subnet-range 198.51.100.0/24␣
↪\
  --service-type 'compute:foo' --network demo-net1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| id                | ea139dcd-17a3-4f0a-8cca-dff8b4e03f8a |
| ip_version        | 4                                    |
| cidr              | 198.51.100.0/24                      |
| name              | demo-subnet2                         |
| network_id        | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| revision_number   | 1                                    |
| service_types     | ['compute:foo']                      |
| tags              | []                                   |
| tenant_id         | a8b3054cc1214f18b1186b291525650f     |
+-------------------+--------------------------------------+
```

4. Launch an instance using the network. For example, using the `cirros` image and `m1.tiny` flavor.

```
$ openstack server create demo-instance1 --flavor m1.tiny \
  --image cirros --nic net-id=b5b729d8-31cc-4d2c-8284-72b3291fec02
+--------------------------------------+-------------------------------
↪-----------------+
| Field                                | Value                        ␣
↪                 |
+--------------------------------------+-------------------------------
↪-----------------+
| OS-DCF:diskConfig                    | MANUAL                       ␣
↪                 |
| OS-EXT-AZ:availability_zone          |                              ␣
↪                 |
| OS-EXT-SRV-ATTR:host                 | None                         ␣
↪                 |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | None                         ␣
↪                 |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000009            ␣
↪                 |
```

(continues on next page)

(continued from previous page)

```
| OS-EXT-STS:power_state               | 0                                ␣
↪                  |
| OS-EXT-STS:task_state                | scheduling                       ␣
↪                  |
| OS-EXT-STS:vm_state                  | building                         ␣
↪                  |
| OS-SRV-USG:launched_at               | None                             ␣
↪                  |
| OS-SRV-USG:terminated_at             | None                             ␣
↪                  |
| accessIPv4                           |                                  ␣
↪                  |
| accessIPv6                           |                                  ␣
↪                  |
| addresses                            |                                  ␣
↪                  |
| adminPass                            | Fn85skabdxBL                     ␣
↪                  |
| config_drive                         |                                  ␣
↪                  |
| created                              | 2016-09-19T15:07:42Z             ␣
↪                  |
| flavor                               | m1.tiny (1)                      ␣
↪                  |
| hostId                               |                                  ␣
↪                  |
| id                                   | 04222b73-1a6e-4c2a-9af4-
↪ef3d17d521ff           |
| image                                | cirros (4aaec87d-c655-4856-
↪8618-b2dada3a2b11) |
| key_name                             | None                             ␣
↪                  |
| name                                 | demo-instance1                   ␣
↪                  |
| os-extended-volumes:volumes_attached | []                               ␣
↪                  |
| progress                             | 0                                ␣
↪                  |
| project_id                           |␣
↪d44c19e056674381b86430575184b167         |
| properties                           |                                  ␣
↪                  |
| security_groups                      | [{u'name': u'default'}]          ␣
↪                  |
| status                               | BUILD                            ␣
↪                  |
| updated                              | 2016-09-19T15:07:42Z             ␣
↪                  |
| user_id                              |␣
↪331afbeb322d4c559a181e19051ae362         |
+--------------------------------------+-----------------------------
↪-----------------+
```

5. Check the instance status. The `Networks` field contains an IP address from the subnet having the `compute:nova` service type.

```
$ openstack server list
+--------------------------------------+-----------------+---------+--
↪-------------------+--------+---------+
| ID                                   | Name            | Status  |␣
↪Networks           | Image  | Flavor  |
+--------------------------------------+-----------------+---------+--
↪-------------------+--------+---------+
| 20181f46-5cd2-4af8-9af0-f4cf5c983008 | demo-instance1  | ACTIVE  |␣
↪demo-net1=192.0.2.3 | cirros | m1.tiny |
+--------------------------------------+-----------------+---------+--
↪-------------------+--------+---------+
```

Example 2 - DVR configuration

The following example outlines how you can configure service subnets in a DVR-enabled deployment, with the goal of minimizing public IP address consumption. This example uses three subnets on the same external network:

- 192.0.2.0/24 for instance floating IP addresses
- 198.51.100.0/24 for floating IP agent gateway IPs configured on compute nodes
- 203.0.113.0/25 for all other IP allocations on the external network

This example uses again the private network, `demo-net1` (b5b729d8-31cc-4d2c-8284-72b3291fec02) which was created in *Example 1 - Proof-of-concept*.

1. Create an external network:

```
$ openstack network create --external demo-ext-net
```

2. Create a subnet on the external network for the instance floating IP addresses. This uses the `network:floatingip` service type.

```
$ openstack subnet create demo-floating-ip-subnet \
  --subnet-range 192.0.2.0/24 --no-dhcp \
  --service-type 'network:floatingip' --network demo-ext-net
```

3. Create a subnet on the external network for the floating IP agent gateway IP addresses, which are configured by DVR on compute nodes. This will use the `network:floatingip_agent_gateway` service type.

```
$ openstack subnet create demo-floating-ip-agent-gateway-subnet \
  --subnet-range 198.51.100.0/24 --no-dhcp \
  --service-type 'network:floatingip_agent_gateway' \
  --network demo-ext-net
```

4. Create a subnet on the external network for all other IP addresses allocated on the external network. This will not use any service type. It acts as a fall back for allocations that do not match either of the above two service subnets.

```
$ openstack subnet create demo-other-subnet \
  --subnet-range 203.0.113.0/25 --no-dhcp \
  --network demo-ext-net
```

5. Create a router:

```
$ openstack router create demo-router
```

6. Add an interface to the router on demo-subnet1:

```
$ openstack router add subnet demo-router demo-subnet1
```

7. Set the external gateway for the router, which will create an interface and allocate an IP address on demo-ext-net:

```
$ openstack router set --external-gateway demo-ext-net demo-router
```

8. Launch an instance on a private network and retrieve the neutron port ID that was allocated. As above, use the `cirros` image and `m1.tiny` flavor:

```
$ openstack server create demo-instance1 --flavor m1.tiny \
  --image cirros --nic net-id=b5b729d8-31cc-4d2c-8284-72b3291fec02
$ openstack port list --server demo-instance1
+--------------------------------------+------+-------------------+---
↪--------------------------------------------------+--------+
| ID                                   | Name | MAC Address       |␣
↪Fixed IP Addresses                                | Status |
+--------------------------------------+------+-------------------+---
↪--------------------------------------------------+--------+
| a752bb24-9bf2-4d37-b9d6-07da69c86f19 |      | fa:16:3e:99:54:32 |␣
↪ip_address='203.0.113.130',                       | ACTIVE |
|                                      |      |                   |␣
↪subnet_id='6e38b23f-0b27-4e3c-8e69-fd23a3df1935' |        |
+--------------------------------------+------+-------------------+---
↪--------------------------------------------------+--------+
```

9. Associate a floating IP with the instance port and verify it was allocated an IP address from the correct subnet:

```
$ openstack floating ip create --port \
  a752bb24-9bf2-4d37-b9d6-07da69c86f19 demo-ext-net
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    | 203.0.113.130                        |
| floating_ip_address | 192.0.2.12                           |
| floating_network_id | 02d236d5-dad9-4082-bb6b-5245f9f84d13 |
| id                  | f15cae7f-5e05-4b19-bd25-4bb71edcf3de |
| port_id             | a752bb24-9bf2-4d37-b9d6-07da69c86f19 |
| project_id          | d44c19e056674381b86430575184b167     |
| revision_number     | 1                                    |
| router_id           | 5a8ca19f-3703-4f81-bc29-db6bc2f528d6 |
| status              | ACTIVE                               |
| tags                | []                                   |
+---------------------+--------------------------------------+
```

10. As the *admin* user, verify the neutron routers are allocated IP addresses from their correct subnets. Use `openstack port list` to find ports associated with the routers.

    First, the router gateway external port:

```
$ openstack port show f148ffeb-3c26-4067-bc5f-5c3dfddae2f5
+-----------------------+-----------------------------------------------
↪-----------------------------+
| Field                 | Value                                         ␣
↪                             |
+-----------------------+-----------------------------------------------
↪-----------------------------+
| admin_state_up        | UP                                            ␣
↪                             |
| device_id             | 5a8ca19f-3703-4f81-bc29-db6bc2f528d6          ␣
↪                             |
| device_owner          | network:router_gateway                        ␣
↪                             |
| extra_dhcp_opts       |                                               ␣
↪                             |
| fixed_ips             | ip_address='203.0.113.11',                    ␣
↪                             |
|                       | subnet_id='67c251d9-2b7a-4200-99f6-
↪e13785b0334d'                         |
| id                    | f148ffeb-3c26-4067-bc5f-5c3dfddae2f5          ␣
↪                             |
| mac_address           | fa:16:3e:2c:0f:69                             ␣
↪                             |
| network_id            | 02d236d5-dad9-4082-bb6b-5245f9f84d13          ␣
↪                             |
| revision_number       | 1                                             ␣
↪                             |
| project_id            |                                               ␣
↪                             |
| status                | ACTIVE                                        ␣
↪                             |
| tags                  | []                                            ␣
↪                             |
+-----------------------+-----------------------------------------------
↪-----------------------------+
```

Second, the router floating IP agent gateway external port:

```
$ openstack port show a2d1e756-8ae1-4f96-9aa1-e7ea16a6a68a
+-----------------------+-----------------------------------------------
↪-----------------------------+
| Field                 | Value                                         ␣
↪                             |
+-----------------------+-----------------------------------------------
↪-----------------------------+
| admin_state_up        | UP                                            ␣
↪                             |
| device_id             | 3d0c98eb-bca3-45cc-8aa4-90ae3deb0844          ␣
↪                             |
| device_owner          | network:floatingip_agent_gateway              ␣
↪                             |
| extra_dhcp_opts       |                                               ␣
↪                             |
| fixed_ips             | ip_address='198.51.100.10',                   ␣
↪                             |
|                       | subnet_id='67c251d9-2b7a-4200-99f6-
↪e13785b0334d'                         |
```

(continues on next page)

(continued from previous page)

```
| id                    | a2d1e756-8ae1-4f96-9aa1-e7ea16a6a68a                 ␣
↪                |
| mac_address           | fa:16:3e:f4:5d:fa                                    ␣
↪                |
| network_id            | 02d236d5-dad9-4082-bb6b-5245f9f84d13                 ␣
↪                |
| project_id            |                                                      ␣
↪                |
| revision_number       | 1                                                    ␣
↪                |
| status                | ACTIVE                                               ␣
↪                |
| tags                  | []                                                   ␣
↪                |
+-----------------------+-------------------------------------------------------
↪----------------+
```

8.2.31 Trunking

The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting it to a single port.

Operation

Network trunking consists of a service plug-in and a set of drivers that manage trunks on different layer-2 mechanism drivers. Users can create a port, associate it with a trunk, and launch an instance on that port. Users can dynamically attach and detach additional networks without disrupting operation of the instance.

Every trunk has a parent port and can have any number of subports. The parent port is the port that the trunk is associated with. Users create instances and specify the parent port of the trunk when launching instances attached to a trunk.

The network presented by the subport is the network of the associated port. When creating a subport, a `segmentation-id` may be required by the driver. `segmentation-id` defines the segmentation ID on which the subport network is presented to the instance. `segmentation-type` may be required by certain drivers like OVS. At this time the following `segmentation-type` values are supported:

- `vlan` uses VLAN for segmentation.
- `inherit` uses the `segmentation-type` from the network the subport is connected to if no `segmentation-type` is specified for the subport. Note that using the `inherit` type requires the `provider` extension to be enabled and only works when the connected networks `segmentation-type` is `vlan`.

Note: The `segmentation-type` and `segmentation-id` parameters are optional in the Networking API. However, all drivers as of the Newton release require both to be provided when adding a subport to a trunk. Future drivers may be implemented without this requirement.

The `segmentation-type` and `segmentation-id` specified by the user on the subports is intentionally decoupled from the `segmentation-type` and ID of the networks. For example, it is possible to configure the Networking service with `tenant_network_types = vxlan` and still create subports with `segmentation_type = vlan`. The Networking service performs remapping as necessary.

Example configuration

The ML2 plug-in supports trunking with the following mechanism drivers:

- Open vSwitch (OVS)
- Linux bridge
- Open Virtual Network (OVN)

When using a `segmentation-type` of `vlan`, the OVS and Linux bridge drivers present the network of the parent port as the untagged VLAN and all subports as tagged VLANs.

Controller node

- In the `neutron.conf` file, enable the trunk service plug-in:

```
[DEFAULT]
service_plugins = trunk
```

Verify service operation

1. Source the administrative project credentials and list the enabled extensions.
2. Use the command **`openstack extension list --network`** to verify that the `Trunk Extension` and `Trunk port details` extensions are enabled.

Workflow

At a high level, the basic steps to launching an instance on a trunk are the following:

1. Create networks and subnets for the trunk and subports
2. Create the trunk
3. Add subports to the trunk
4. Launch an instance on the trunk

Create networks and subnets for the trunk and subports

Create the appropriate networks for the trunk and subports that will be added to the trunk. Create subnets on these networks to ensure the desired layer-3 connectivity over the trunk.

Create the trunk

- Create a parent port for the trunk.

```
$ openstack port create --network project-net-A trunk-parent
+-------------------+-----------------------------------------------------------------------+
| Field             | Value                                                                 |
+-------------------+-----------------------------------------------------------------------+
| admin_state_up    | UP                                                                    |
| binding_vif_type  | unbound                                                               |
| binding_vnic_type | normal                                                                |
| fixed_ips         | ip_address='192.0.2.7',subnet_id='8b957198-d3cf-4953-8449-ad4e4dd712cc' |
| id                | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38                                  |
| mac_address       | fa:16:3e:dd:c4:d1                                                     |
| name              | trunk-parent                                                          |
| network_id        | 1b47d3e7-cda5-48e4-b0c8-d20bd7e35f55                                  |
| revision_number   | 1                                                                     |
| tags              | []                                                                    |
+-------------------+-----------------------------------------------------------------------+
```

- Create the trunk using `--parent-port` to reference the port from the previous step:

```
$ openstack network trunk create --parent-port trunk-parent trunk1
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| id              | fdf02fcb-1844-45f1-9d9b-e4c2f522c164 |
| name            | trunk1                               |
| port_id         | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38 |
| revision_number | 1                                    |
| sub_ports       |                                      |
+-----------------+--------------------------------------+
```

Add subports to the trunk

Subports can be added to a trunk in two ways: creating the trunk with subports or adding subports to an existing trunk.

- Create trunk with subports:

  This method entails creating the trunk with subports specified at trunk creation.

```
$ openstack port create --network project-net-A trunk-parent
+-------------------+-------------------------------------------------
↪-----------------------+
| Field             | Value                                           ␣
↪                       |
+-------------------+-------------------------------------------------
↪-----------------------+
| admin_state_up    | UP                                              ␣
↪                       |
| binding_vif_type  | unbound                                         ␣
↪                       |
| binding_vnic_type | normal                                          ␣
↪                       |
| fixed_ips         | ip_address='192.0.2.7',subnet_id='8b957198-d3cf-
↪4953-8449-ad4e4dd712cc' |
| id                | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38            ␣
↪                       |
| mac_address       | fa:16:3e:dd:c4:d1                               ␣
↪                       |
| name              | trunk-parent                                    ␣
↪                       |
| network_id        | 1b47d3e7-cda5-48e4-b0c8-d20bd7e35f55            ␣
↪                       |
| revision_number   | 1                                               ␣
↪                       |
| tags              | []                                              ␣
↪                       |
+-------------------+-------------------------------------------------
↪-----------------------+

$ openstack port create --network trunked-net subport1
+-------------------+-------------------------------------------------
↪---------------------------+
| Field             | Value                                           ␣
↪                           |
+-------------------+-------------------------------------------------
↪---------------------------+
| admin_state_up    | UP                                              ␣
↪                           |
| binding_vif_type  | unbound                                         ␣
↪                           |
| binding_vnic_type | normal                                          ␣
↪                           |
| fixed_ips         | ip_address='198.51.100.8',subnet_id='2a860e2c-
↪922b-437b-a149-b269a8c9b120' |
| id                | 91f9dde8-80a4-4506-b5da-c287feb8f5d8            ␣
↪                           |
| mac_address       | fa:16:3e:ba:f0:4d                               ␣
↪                           |
```

(continues on next page)

(continued from previous page)

```
| name              | subport1
↪                         |
| network_id        | aef78ec5-16e3-4445-b82d-b2b98c6a86d9
↪                         |
| revision_number   | 1
↪                         |
| tags              | []
↪                         |
+-------------------+-----------------------------------------------------
↪---------------------------+

$ openstack network trunk create \
  --parent-port trunk-parent \
  --subport port=subport1,segmentation-type=vlan,segmentation-id=100 \
  trunk1
+-----------------+-----------------------------------------------------
↪-----------------------------------------------+
| Field           | Value
↪                                                 |
+-----------------+-----------------------------------------------------
↪-----------------------------------------------+
| admin_state_up  | UP
↪                                                 |
| id              | 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
↪                                                 |
| name            | trunk1
↪                                                 |
| port_id         | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
↪                                                 |
| revision_number| 1
↪                                                 |
| sub_ports       | port_id='73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38',
↪segmentation_id='100', segmentation_type='vlan' |
| tags            | []
↪                                                 |
+-----------------+-----------------------------------------------------
↪-----------------------------------------------+
```

- Add subports to an existing trunk:

  This method entails creating a trunk, then adding subports to the trunk after it has already been created.

```
$ openstack network trunk set --subport \
  port=subport1,segmentation-type=vlan,segmentation-id=100 \
  trunk1
```

Note: The command provides no output.

```
$ openstack network trunk show trunk1
+-----------------+-----------------------------------------------------
↪-----------------------------------------------+
| Field           | Value
↪                                                 |
```

(continues on next page)

(continued from previous page)

```
+-----------------+-----------------------------------------------------------
↪-----------------------------------------------+
| admin_state_up  | UP                                                        ␣
↪                                               |
| id              | 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3                      ␣
↪                                               |
| name            | trunk1                                                    ␣
↪                                               |
| port_id         | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38                      ␣
↪                                               |
| revision_number| 1                                                          ␣
↪                                               |
| sub_ports       | port_id='73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38',␣
↪segmentation_id='100', segmentation_type='vlan' |
| tags            | []                                                        ␣
↪                                               |
+-----------------+-----------------------------------------------------------
↪-----------------------------------------------+
```

- When using the OVN driver, additional logical switch port information is available using the following commands:

```
$ ovn-nbctl lsp-get-parent 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38

$ ovn-nbctl lsp-get-tag 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
```

Launch an instance on the trunk

- Show trunk details to get the `port_id` of the trunk.

```
$ openstack network trunk show trunk1
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| id              | 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3 |
| name            | trunk                                |
| port_id         | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38 |
| revision_number| 1                                     |
| sub_ports       |                                      |
| tags            | []                                   |
+-----------------+--------------------------------------+
```

- Launch the instance by specifying `port-id` using the value of `port_id` from the trunk details. Launching an instance on a subport is not supported.

Using trunks and subports inside an instance

When configuring instances to use a subport, ensure that the interface on the instance is set to use the MAC address assigned to the port by the Networking service. Instances are not made aware of changes made to the trunk after they are active. For example, when a subport with a `segmentation-type` of `vlan` is added to a trunk, any operations specific to the instance operating system that allow the instance to send and receive traffic on the new VLAN must be handled outside of the Networking service.

When creating subports, the MAC address of the trunk parent port can be set on the subport. This will allow VLAN subinterfaces inside an instance launched on a trunk to be configured without explicitly setting a MAC address. Although unique MAC addresses can be used for subports, this can present issues with ARP spoof protections and the native OVS firewall driver. If the native OVS firewall driver is to be used, we recommend that the MAC address of the parent port be re-used on all subports.

Trunk states

- `ACTIVE`

  The trunk is `ACTIVE` when both the logical and physical resources have been created. This means that all operations within the Networking and Compute services have completed and the trunk is ready for use.

- `DOWN`

  A trunk is `DOWN` when it is first created without an instance launched on it, or when the instance associated with the trunk has been deleted.

- `DEGRADED`

  A trunk can be in a `DEGRADED` state when a temporary failure during the provisioning process is encountered. This includes situations where a subport add or remove operation fails. When in a degraded state, the trunk is still usable and some subports may be usable as well. Operations that cause the trunk to go into a `DEGRADED` state can be retried to fix temporary failures and move the trunk into an `ACTIVE` state.

- `ERROR`

  A trunk is in `ERROR` state if the request leads to a conflict or an error that cannot be fixed by retrying the request. The `ERROR` status can be encountered if the network is not compatible with the trunk configuration or the binding process leads to a persistent failure. When a trunk is in `ERROR` state, it must be brought to a sane state (`ACTIVE`), or else requests to add subports will be rejected.

- `BUILD`

  A trunk is in `BUILD` state while the resources associated with the trunk are in the process of being provisioned. Once the trunk and all of the subports have been provisioned successfully, the trunk transitions to `ACTIVE`. If there was a partial failure, the trunk transitions to `DEGRADED`.

  When `admin_state` is set to `DOWN`, the user is blocked from performing operations on the trunk. `admin_state` is set by the user and should not be used to monitor the health of the trunk.

Limitations and issues

- In `neutron-ovs-agent` the use of `iptables_hybrid` firewall driver and trunk ports are not compatible with each other. The `iptables_hybrid` firewall is not going to filter the traffic of subports. Instead use other firewall drivers like `openvswitch`.
- See bugs for more information.

8.2.32 Installing Neutron API via WSGI

This document is a guide to deploying neutron using WSGI. There are two ways to deploy using WSGI: `uwsgi` and Apache `mod_wsgi`.

Please note that if you intend to use mode uwsgi, you should install the `mode_proxy_uwsgi` module. For example on deb-based system:

```
# sudo apt-get install libapache2-mod-proxy-uwsgi
# sudo a2enmod proxy
# sudo a2enmod proxy_uwsgi
```

WSGI Application

The function `neutron.server.get_application` will setup a WSGI application to run behind uwsgi and mod_wsgi.

Neutron API behind uwsgi

Create a `/etc/neutron/neutron-api-uwsgi.ini` file with the content below:

```
[uwsgi]
chmod-socket = 666
socket = /var/run/uwsgi/neutron-api.socket
lazy-apps = true
add-header = Connection: close
buffer-size = 65535
hook-master-start = unix_signal:15 gracefully_kill_them_all
thunder-lock = true
plugins = python
enable-threads = true
worker-reload-mercy = 90
exit-on-reload = false
die-on-term = true
master = true
processes = 2
wsgi-file = <path-to-neutron-bin-dir>/neutron-api
```

Start neutron-api:

```
# uwsgi --procname-prefix neutron-api --ini /etc/neutron/neutron-api-uwsgi.
↪ini
```

Neutron API behind mod_wsgi

Create /etc/apache2/neutron.conf with content below:

```
Listen 9696
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"
↪%D(us)" neutron_combined

<Directory /usr/local/bin>
    Require all granted
</Directory>

<VirtualHost *:9696>
    WSGIDaemonProcess neutron-server processes=1 threads=1 user=stack␣
↪display-name=%{GROUP}
    WSGIProcessGroup neutron-server
    WSGIScriptAlias / <path-to-neutron-bin-dir>/neutron-api
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    ErrorLogFormat "%M"
    ErrorLog /var/log/neutron/neutron.log
    CustomLog /var/log/neutron/neutron_access.log neutron_combined
</VirtualHost>

Alias /networking <path-to-neutron-bin-dir>/neutron-api
<Location /networking>
    SetHandler wsgi-script
    Options +ExecCGI
    WSGIProcessGroup neutron-server
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
</Location>

WSGISocketPrefix /var/run/apache2
```

For deb-based systems copy or symlink the file to /etc/apache2/sites-available. Then enable the neutron site:

```
# a2ensite neutron
# systemctl reload apache2.service
```

For rpm-based systems copy the file to /etc/httpd/conf.d. Then enable the neutron site:

```
# systemctl reload httpd.service
```

Start Neutron RPC server

When Neutron API is served by a web server (like Apache2) it is difficult to start an rpc listener thread. So start the Neutron RPC server process to serve this job:

```
# /usr/bin/neutron-rpc-server --config-file /etc/neutron/neutron.conf --
↪config-file /etc/neutron/plugins/ml2/ml2_conf.ini
```

Neutron Worker Processes

Neutron will attempt to spawn a number of child processes for handling API and RPC requests. The number of API workers is set to the number of CPU cores, further limited by available memory, and the number of RPC workers is set to half that number.

It is strongly recommended that all deployers set these values themselves, via the api_workers and rpc_workers configuration parameters.

For a cloud with a high load to a relatively small number of objects, a smaller value for api_workers will provide better performance than many (somewhere around 4-8.) For a cloud with a high load to lots of different objects, then the more the better. Budget neutron-server using about 2GB of RAM in steady-state.

For rpc_workers, there needs to be enough to keep up with incoming events from the various neutron agents. Signs that there are too few can be agent heartbeats arriving late, nova vif bindings timing out on the hypervisors, or rpc message timeout exceptions in agent logs.

Note: For general configuration, see the Configuration Reference.

8.3 Deployment examples

The following deployment examples provide building blocks of increasing architectural complexity using the Networking service reference architecture which implements the Modular Layer 2 (ML2) plug-in and either the Open vSwitch (OVS) or Linux bridge mechanism drivers. Both mechanism drivers support the same basic features such as provider networks, self-service networks, and routers. However, more complex features often require a particular mechanism driver. Thus, you should consider the requirements (or goals) of your cloud before choosing a mechanism driver.

After choosing a *mechanism driver*, the deployment examples generally include the following building blocks:

1. Provider (public/external) networks using IPv4 and IPv6
2. Self-service (project/private/internal) networks including routers using IPv4 and IPv6
3. High-availability features
4. Other features such as BGP dynamic routing

8.3.1 Prerequisites

Prerequisites, typically hardware requirements, generally increase with each building block. Each building block depends on proper deployment and operation of prior building blocks. For example, the first building block (provider networks) only requires one controller and two compute nodes, the second building block (self-service networks) adds a network node, and the high-availability building blocks typically add a second network node for a total of five nodes. Each building block could also require additional infrastructure or changes to existing infrastructure such as networks.

For basic configuration of prerequisites, see the latest Install Tutorials and Guides.

Note: Example commands using the `openstack` client assume version 3.2.0 or higher.

Nodes

The deployment examples refer one or more of the following nodes:

- Controller: Contains control plane components of OpenStack services and their dependencies.
  - Two network interfaces: management and provider.
  - Operational SQL server with databases necessary for each OpenStack service.
  - Operational message queue service.
  - Operational OpenStack Identity (keystone) service.
  - Operational OpenStack Image Service (glance).
  - Operational management components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
  - OpenStack Networking (neutron) server service and ML2 plug-in.
- Network: Contains the OpenStack Networking service layer-3 (routing) component. High availability options may include additional components.
  - Three network interfaces: management, overlay, and provider.
  - OpenStack Networking layer-2 (switching) agent, layer-3 agent, and any dependencies.
- Compute: Contains the hypervisor component of the OpenStack Compute service and the OpenStack Networking layer-2, DHCP, and metadata components. High-availability options may include additional components.
  - Two network interfaces: management and provider.
  - Operational hypervisor components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
  - OpenStack Networking layer-2 agent, DHCP agent, metadata agent, and any dependencies.

Each building block defines the quantity and types of nodes including the components on each node.

Note: You can virtualize these nodes for demonstration, training, or proof-of-concept purposes. However, you must use physical hosts for evaluation of performance or scaling.

Networks and network interfaces

The deployment examples refer to one or more of the following networks and network interfaces:

- Management: Handles API requests from clients and control plane traffic for OpenStack services including their dependencies.
- Overlay: Handles self-service networks using an overlay protocol such as VXLAN or GRE.
- Provider: Connects virtual and physical networks at layer-2. Typically uses physical network infrastructure for switching/routing traffic to external networks such as the Internet.

Note: For best performance, 10+ Gbps physical network infrastructure should support jumbo frames.

For illustration purposes, the configuration examples typically reference the following IP address ranges:

- Provider network 1:
  - IPv4: 203.0.113.0/24
  - IPv6: fd00:203:0:113::/64
- Provider network 2:
  - IPv4: 192.0.2.0/24
  - IPv6: fd00:192:0:2::/64
- Self-service networks:
  - IPv4: 198.51.100.0/24 in /24 segments
  - IPv6: fd00:198:51::/48 in /64 segments

You may change them to work with your particular network infrastructure.

8.3.2 Mechanism drivers

Linux bridge mechanism driver

The Linux bridge mechanism driver uses only Linux bridges and `veth` pairs as interconnection devices. A layer-2 agent manages Linux bridges on each compute node and any other node that provides layer-3 (routing), DHCP, metadata, or other network services.

Linux bridge: Provider networks

The provider networks architecture example provides layer-2 connectivity between instances and the physical network infrastructure using VLAN (802.1q) tagging. It supports one untagged (flat) network and up to 4095 tagged (VLAN) networks. The actual quantity of VLAN networks depends on the physical network infrastructure. For more information on provider networks, see *Provider networks*.

Prerequisites

One controller node with the following components:

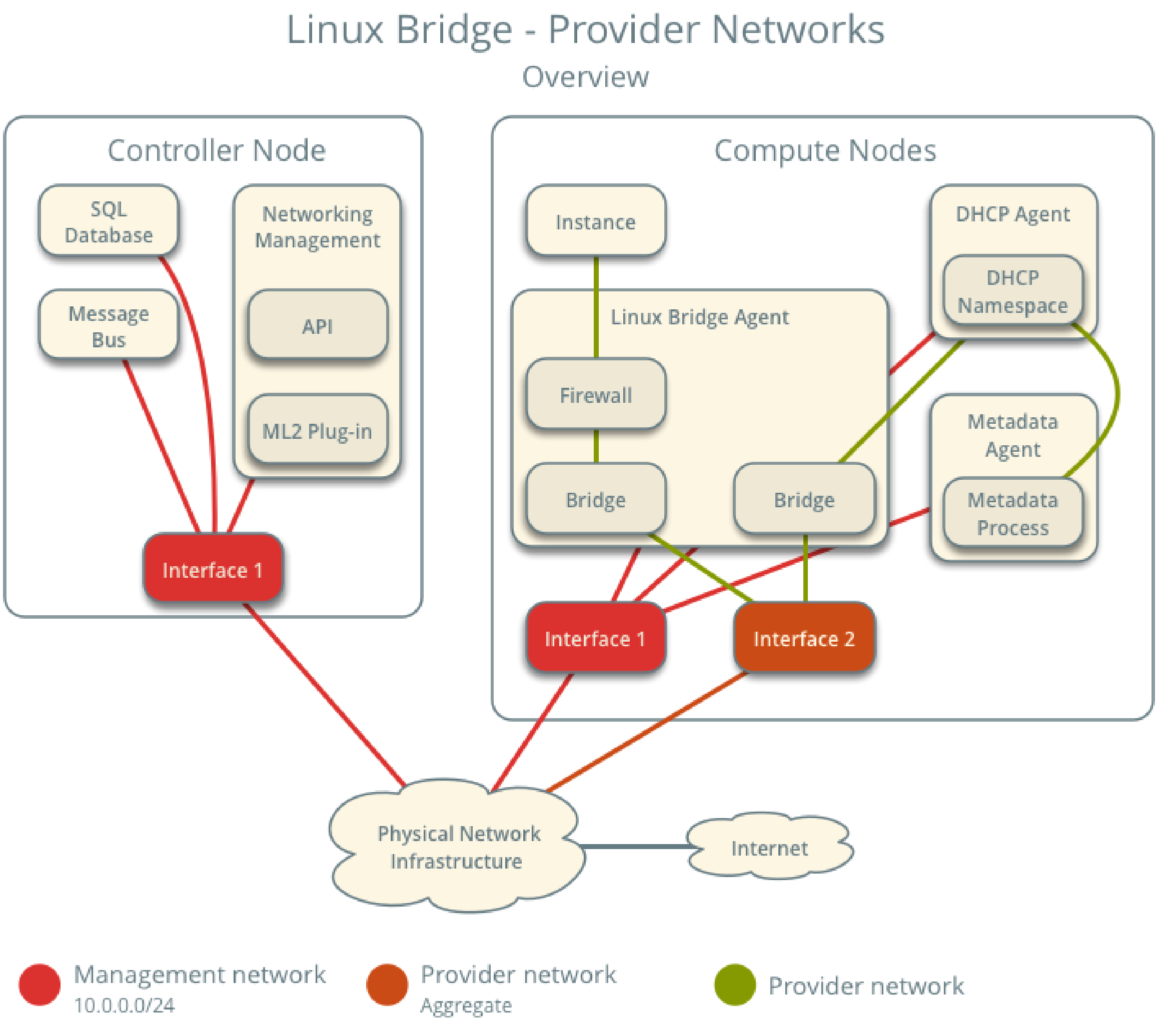
- Two network interfaces: management and provider.
- OpenStack Networking server service and ML2 plug-in.

Two compute nodes with the following components:

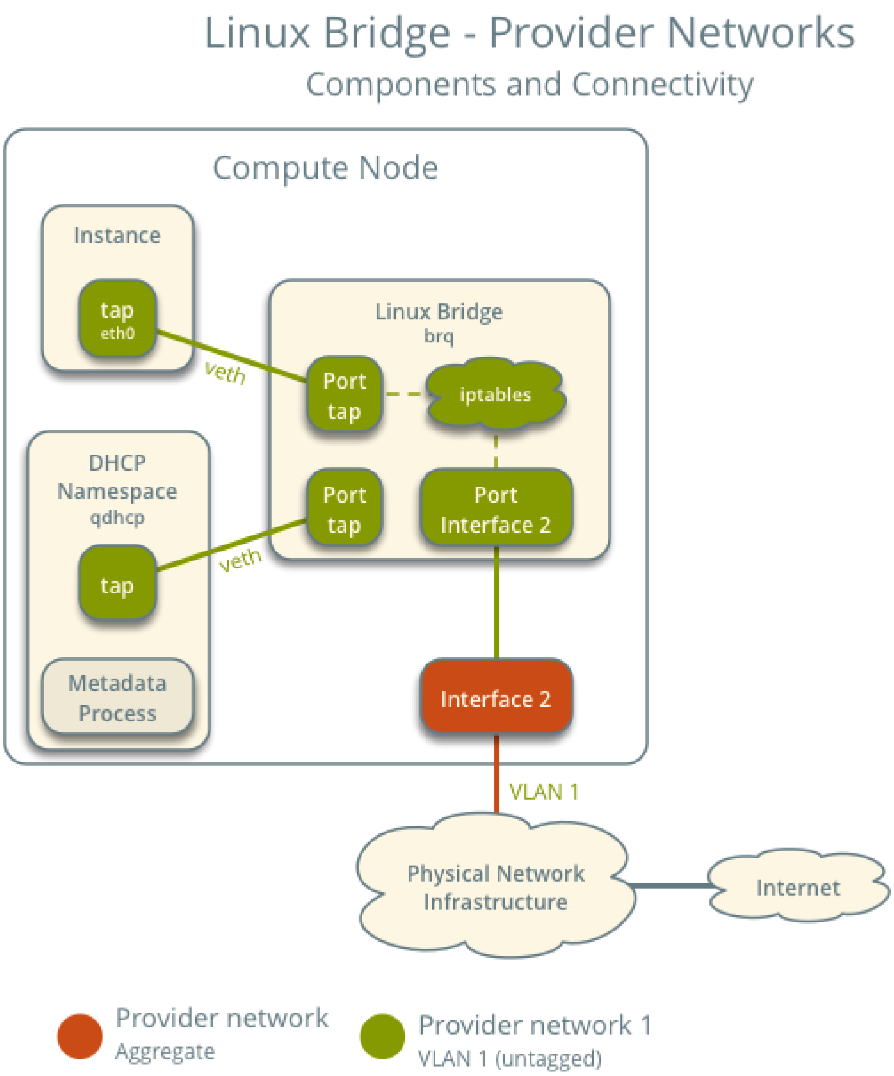
- Two network interfaces: management and provider.
- OpenStack Networking Linux bridge layer-2 agent, DHCP agent, metadata agent, and any dependencies.

Note: Larger deployments typically deploy the DHCP and metadata agents on a subset of compute nodes to increase performance and redundancy. However, too many agents can overwhelm the message bus. Also, to further simplify any deployment, you can omit the metadata agent and use a configuration drive to provide metadata to instances.

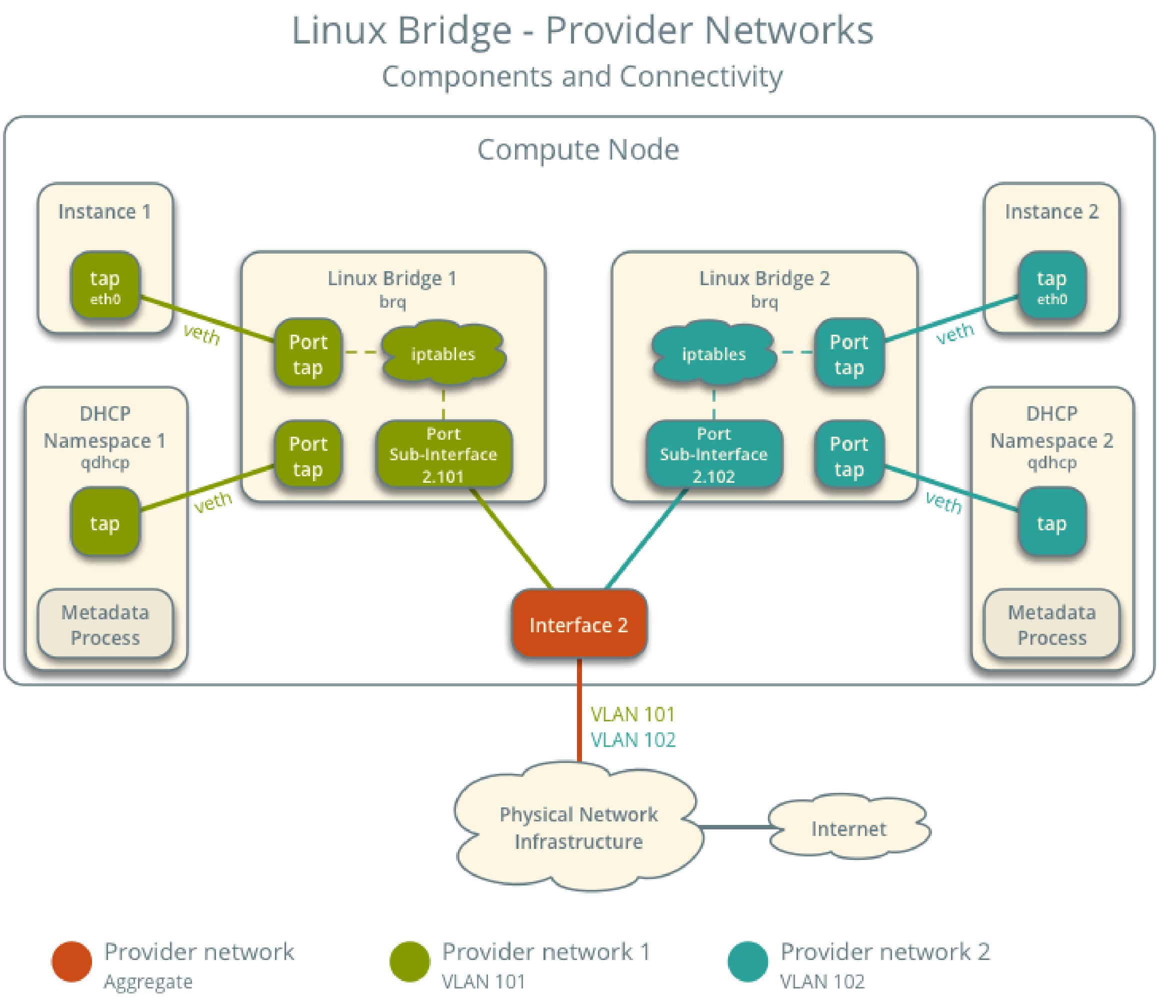
Architecture



The following figure shows components and connectivity for one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the provider physical network interface.



The following figure describes virtual connectivity among components for two tagged (VLAN) networks. Essentially, each network uses a separate bridge that contains a port on the VLAN sub-interface on the provider physical network interface. Similar to the single untagged network case, the DHCP agent may reside on a different compute node.



Note: These figures omit the controller node because it does not handle instance network traffic.

Example configuration

Use the following example configuration as a template to deploy provider networks in your environment.

Controller node

1. Install the Networking service components that provides the `neutron-server` service and ML2 plug-in.
2. In the `neutron.conf` file:
   - Configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone
```

(continues on next page)

(continued from previous page)

```
[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

- Disable service plug-ins because provider networks do not require any. However, this breaks portions of the dashboard that manage the Networking service. See the latest Install Tutorials and Guides for more information.

```
[DEFAULT]
service_plugins =
```

- Enable two DHCP agents per network so both compute nodes can provide DHCP service provider networks.

```
[DEFAULT]
dhcp_agents_per_network = 2
```

- If necessary, *configure MTU*.

3. In the `ml2_conf.ini` file:

- Configure drivers and network types:

```
[ml2]
type_drivers = flat,vlan
tenant_network_types =
mechanism_drivers = linuxbridge
extension_drivers = port_security
```

- Configure network mappings:

```
[ml2_type_flat]
flat_networks = provider

[ml2_type_vlan]
network_vlan_ranges = provider
```

Note: The `tenant_network_types` option contains no value because the architecture does not support self-service networks.

Note: The `provider` value in the `network_vlan_ranges` option lacks VLAN ID ranges to support use of arbitrary VLAN IDs.

4. Populate the database.

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/
↪neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head"␣
↪neutron
```

5. Start the following services:

   - Server

Compute nodes

1. Install the Networking service Linux bridge layer-2 agent.

2. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

   See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

3. In the `linuxbridge_agent.ini` file, configure the Linux bridge agent:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE

[vxlan]
enable_vxlan = False

[securitygroup]
firewall_driver = iptables
```

   Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

4. In the `dhcp_agent.ini` file, configure the DHCP agent:

```
[DEFAULT]
interface_driver = linuxbridge
enable_isolated_metadata = True
force_metadata = True
```

Note: The `force_metadata` option forces the DHCP agent to provide a host route to the metadata service on `169.254.169.254` regardless of whether the subnet contains an interface on a router, thus maintaining similar and predictable metadata behavior among subnets.

5. In the `metadata_agent.ini` file, configure the metadata agent:

```
[DEFAULT]
nova_metadata_host = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

   The value of `METADATA_SECRET` must match the value of the same option in the `[neutron]` section of the `nova.conf` file.

6. Start the following services:

   - Linux bridge agent
   - DHCP agent
   - Metadata agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents:

```
$ openstack network agent list
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host     ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent |␣
↪compute2 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
| 188945d1-9e70-4803-a276-df924e0788a4 | Linux bridge agent |␣
↪compute1 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
| e76c440d-d5f6-4316-a674-d689630b629e | DHCP agent         |␣
↪compute1 | nova              | True  | UP    | neutron-dhcp-agent     ␣
↪      |
| e67367de-6657-11e6-86a4-931cd04404bb | DHCP agent         |␣
↪compute2 | nova              | True  | UP    | neutron-dhcp-agent     ␣
↪      |
| e8174cae-6657-11e6-89f0-534ac6d0cb5c | Metadata agent     |␣
↪compute1 | None              | True  | UP    | neutron-metadata-
↪agent     |
```

(continues on next page)

(continued from previous page)

```
| ece49ec6-6657-11e6-bafb-c7560f19197d | Metadata agent     |␣
↪compute2 | None                 | True  | UP    | neutron-metadata-
↪agent    |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
```

Create initial networks

The configuration supports one flat or multiple VLAN provider networks. For simplicity, the following procedure creates one flat provider network.

1. Source the administrative project credentials.
2. Create a flat network.

```
$ openstack network create --share --provider-physical-network␣
↪provider \
  --provider-network-type flat provider1
+---------------------------+-----------+-
| Field                     | Value     |
+---------------------------+-----------+
| admin_state_up            | UP        |
| mtu                       | 1500      |
| name                      | provider1 |
| port_security_enabled     | True      |
| provider:network_type     | flat      |
| provider:physical_network | provider  |
| provider:segmentation_id  | None      |
| router:external           | Internal  |
| shared                    | True      |
| status                    | ACTIVE    |
+---------------------------+-----------+
```

Note: The `share` option allows any project to use this network. To limit access to provider networks, see *Role-Based Access Control (RBAC)*.

Note: To create a VLAN network instead of a flat network, change `--provider-network-type flat` to `--provider-network-type vlan` and add `--provider-segment` with a value referencing the VLAN ID.

3. Create a IPv4 subnet on the provider network.

```
$ openstack subnet create --subnet-range 203.0.113.0/24 --gateway 203.
↪0.113.1 \
  --network provider1 --allocation-pool start=203.0.113.11,end=203.0.
↪113.250 \
  --dns-nameserver 8.8.4.4 provider1-v4
+-------------------+---------------------------+
| Field             | Value                     |
+-------------------+---------------------------+
```

(continues on next page)

(continued from previous page)

```
| allocation_pools  | 203.0.113.11-203.0.113.250 |
| cidr              | 203.0.113.0/24             |
| dns_nameservers   | 8.8.4.4                    |
| enable_dhcp       | True                       |
| gateway_ip        | 203.0.113.1                |
| ip_version        | 4                          |
| name              | provider1-v4               |
+-------------------+----------------------------+
```

Important: Enabling DHCP causes the Networking service to provide DHCP which can interfere with existing DHCP services on the physical network infrastructure. Use the `--no-dhcp` option to have the subnet managed by existing DHCP services.

4. Create a IPv6 subnet on the provider network.

```
$ openstack subnet create --subnet-range fd00:203:0:113::/64 --
↪gateway fd00:203:0:113::1 \
  --ip-version 6 --ipv6-address-mode slaac --network provider1 \
  --dns-nameserver 2001:4860:4860::8844 provider1-v6
+-------------------+------------------------------------------------
↪------+
| Field             | Value                                          ␣
↪      |
+-------------------+------------------------------------------------
↪------+
| allocation_pools  | fd00:203:0:113::2-
↪fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr              | fd00:203:0:113::/64                            ␣
↪      |
| dns_nameservers   | 2001:4860:4860::8844                           ␣
↪      |
| enable_dhcp       | True                                           ␣
↪      |
| gateway_ip        | fd00:203:0:113::1                              ␣
↪      |
| ip_version        | 6                                              ␣
↪      |
| ipv6_address_mode | slaac                                          ␣
↪      |
| ipv6_ra_mode      | None                                           ␣
↪      |
| name              | provider1-v6                                   ␣
↪      |
+-------------------+------------------------------------------------
↪------+
```

Note: The Networking service uses the layer-3 agent to provide router advertisement. Provider networks rely on physical network infrastructure for layer-3 services rather than the layer-3 agent. Thus, the physical network infrastructure must provide router advertisement on provider networks for proper operation of IPv6.

Verify network operation

1. On each compute node, verify creation of the qdhcp namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
```

2. Source a regular (non-administrative) project credentials.

3. Create the appropriate security group rules to allow ping and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| protocol         | icmp      |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-
↪icmp default
+-----------+-----------+
| Field     | Value     |
+-----------+-----------+
| direction | ingress   |
| ethertype | IPv6      |
| protocol  | ipv6-icmp |
+-----------+-----------+

$ openstack security group rule create --proto tcp --dst-port 22␣
↪default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto tcp --
↪dst-port 22 default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv6      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
+------------------+-----------+
```

4. Launch an instance with an interface on the provider network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros \
  --nic net-id=NETWORK_ID provider-instance1
```

Replace NETWORK_ID with the ID of the provider network.

5. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+--------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
| ID                                   | Name               | Status␣
↪| Networks                                                   |␣
↪Image   | Flavor   |
+--------------------------------------+--------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
| 018e0ae2-b43c-4271-a78d-62653dd03285 | provider-instance1 | ACTIVE␣
↪| provider1=203.0.113.13, fd00:203:0:113:f816:3eff:fe58:be4e |␣
↪cirros | m1.tiny |
+--------------------------------------+--------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
```

6. On the controller node or any host with access to the provider network, ping the IPv4 and IPv6 addresses of the instance.

```
$ ping -c 4 203.0.113.13
PING 203.0.113.13 (203.0.113.13) 56(84) bytes of data.
64 bytes from 203.0.113.13: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.13: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.13: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.13: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.13 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms

$ ping6 -c 4 fd00:203:0:113:f816:3eff:fe58:be4e
PING␣
↪fd00:203:0:113:f816:3eff:fe58:be4e(fd00:203:0:113:f816:3eff:fe58:be4e)␣
↪56 data bytes
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=1 ttl=64␣
↪time=1.25 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=2 ttl=64␣
↪time=0.683 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=3 ttl=64␣
↪time=0.762 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=4 ttl=64␣
↪time=0.486 ms

--- fd00:203:0:113:f816:3eff:fe58:be4e ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.486/0.796/1.253/0.282 ms
```

7. Obtain access to the instance.
8. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network 1 (VLAN)
  - VLAN ID 101 (tagged)
  - IP address ranges 203.0.113.0/24 and fd00:203:0:113::/64
  - Gateway (via physical network infrastructure)
    - IP addresses 203.0.113.1 and fd00:203:0:113:0::1
- Provider network 2 (VLAN)
  - VLAN ID 102 (tagged)
  - IP address range 192.0.2.0/24 and fd00:192:0:2::/64
  - Gateway
    - IP addresses 192.0.2.1 and fd00:192:0:2::1
- Instance 1
  - IP addresses 203.0.113.101 and fd00:203:0:113:0::101
- Instance 2
  - IP addresses 192.0.2.101 and fd00:192:0:2:0::101

North-south scenario: Instance with a fixed IP address

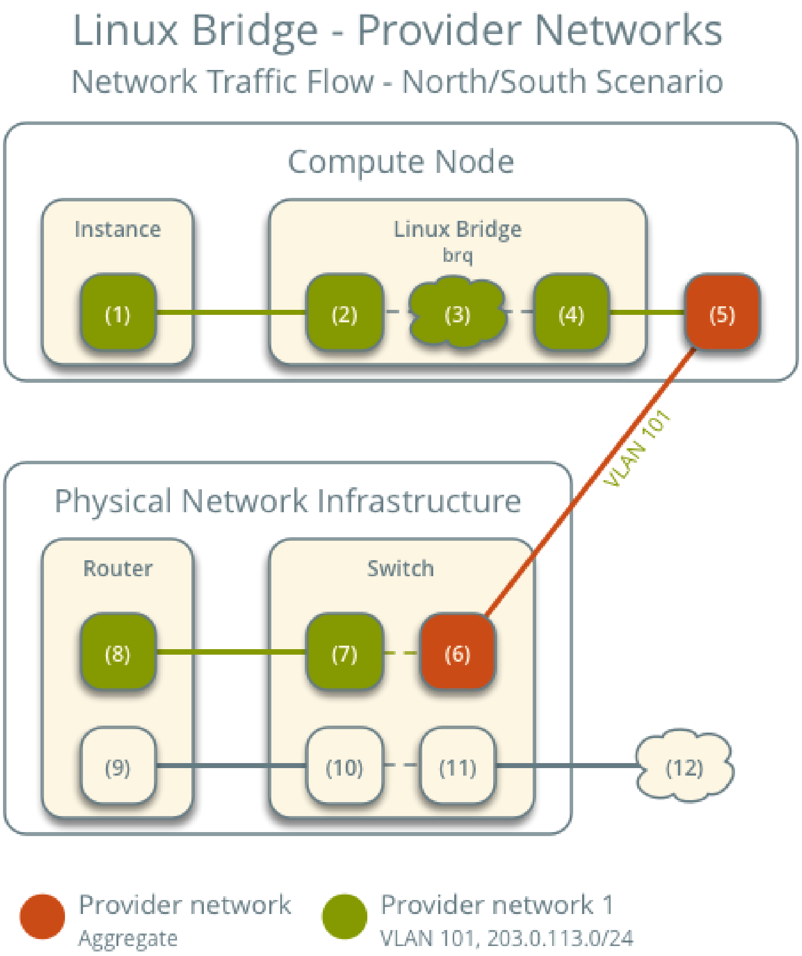
- The instance resides on compute node 1 and uses provider network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1.

1. The instance interface (1) forwards the packet to the provider bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (7).
2. The router routes the packet from the provider network (8) to the external network (9) and forwards the packet to the switch (10).
3. The switch forwards the packet to the external network (11).
4. The external network (12) receives the packet.



Note: Return traffic follows similar steps in reverse.

East-west scenario 1: Instances on the same network

Instances on the same network communicate directly between compute nodes containing those instances.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 2 and uses provider network 1.
- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

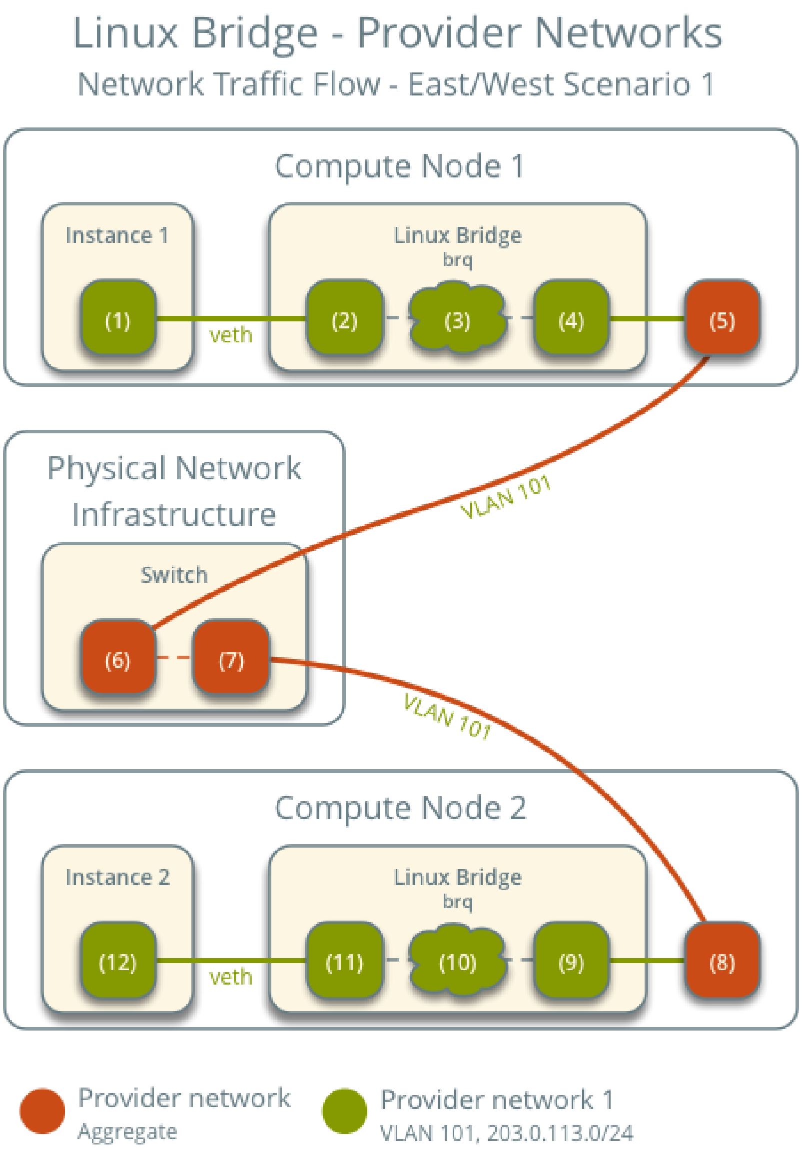
1. The instance 1 interface (1) forwards the packet to the provider bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

1. The switch forwards the packet from compute node 1 to compute node 2 (7).

The following steps involve compute node 2:

1. The physical network interface (8) removes VLAN tag 101 from the packet and forwards it to the VLAN sub-interface port (9) on the provider bridge.
2. Security group rules (10) on the provider bridge handle firewalling and connection tracking for the packet.
3. The provider bridge instance port (11) forwards the packet to the instance 2 interface (12) via `veth` pair.



Note: Return traffic follows similar steps in reverse.

East-west scenario 2: Instances on different networks

Instances communicate via router on the physical network infrastructure.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 1 and uses provider network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VLAN tagging enables multiple logical layer-2 networks to use the same physical layer-2 network.

The following steps involve the compute node:

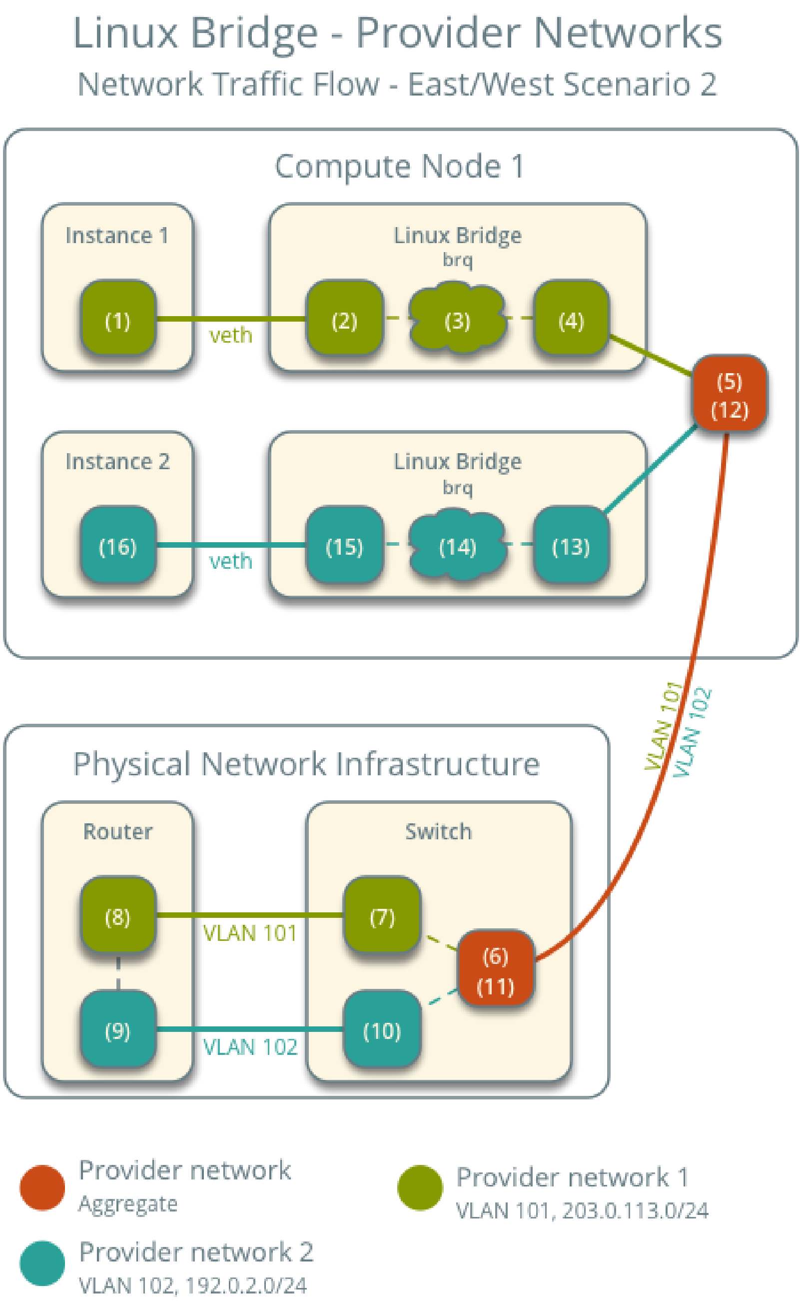
1. The instance 1 interface (1) forwards the packet to the provider bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (7).
2. The router routes the packet from provider network 1 (8) to provider network 2 (9).
3. The router forwards the packet to the switch (10).
4. The switch adds VLAN tag 102 to the packet and forwards it to compute node 1 (11).

The following steps involve the compute node:

1. The physical network interface (12) removes VLAN tag 102 from the packet and forwards it to the VLAN sub-interface port (13) on the provider bridge.
2. Security group rules (14) on the provider bridge handle firewalling and connection tracking for the packet.
3. The provider bridge instance port (15) forwards the packet to the instance 2 interface (16) via `veth` pair.



Note: Return traffic follows similar steps in reverse.

Linux bridge: Self-service networks

This architecture example augments *Linux bridge: Provider networks* to support a nearly limitless quantity of entirely virtual networks. Although the Networking service supports VLAN self-service networks, this example focuses on VXLAN self-service networks. For more information on self-service networks, see *Self-service networks*.

Note: The Linux bridge agent lacks support for other overlay protocols such as GRE and Geneve.

Prerequisites

Add one network node with the following components:

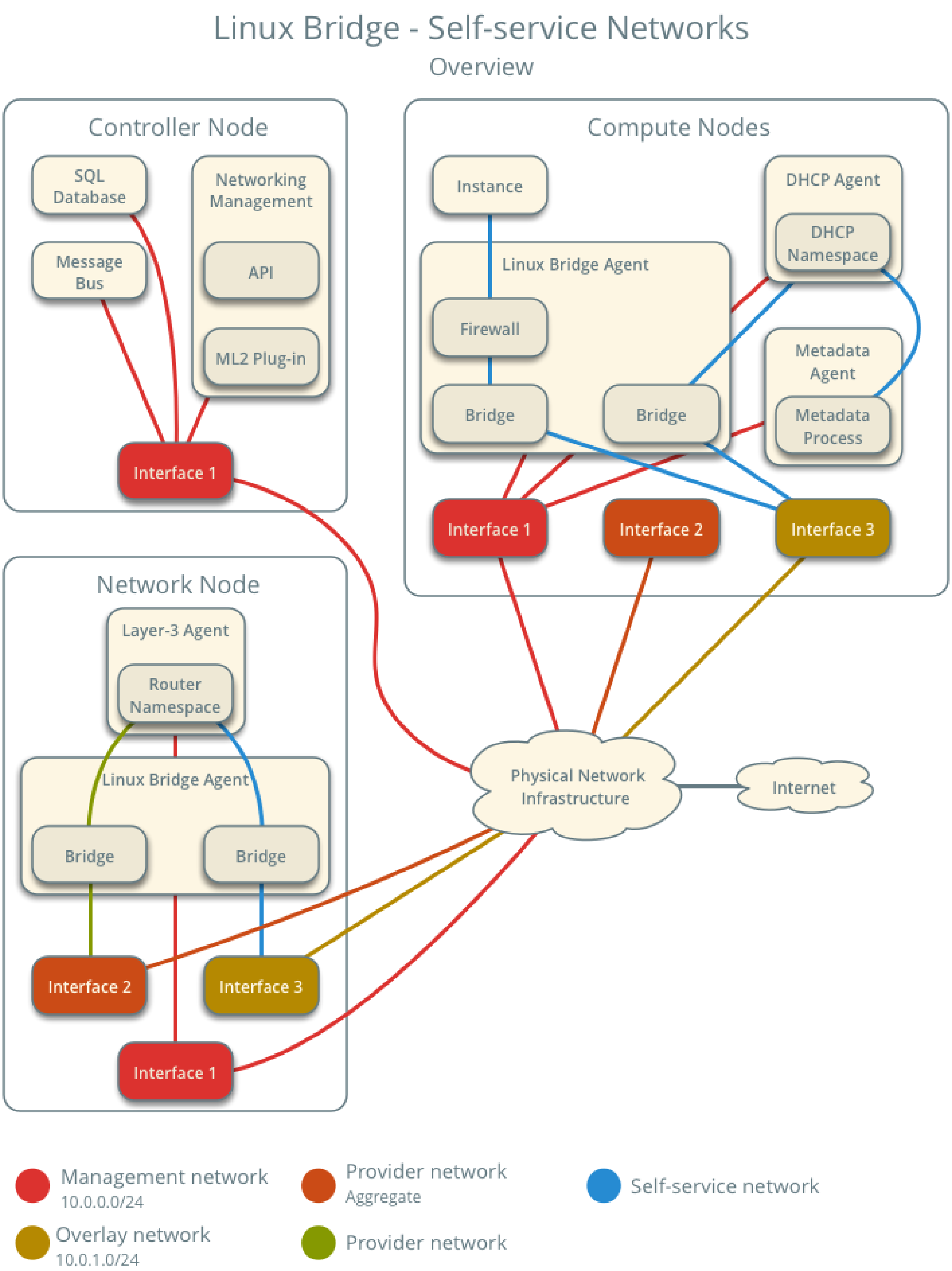
- Three network interfaces: management, provider, and overlay.
- **OpenStack Networking Linux bridge layer-2 agent, layer-3 agent, and any** dependencies.

Modify the compute nodes with the following components:

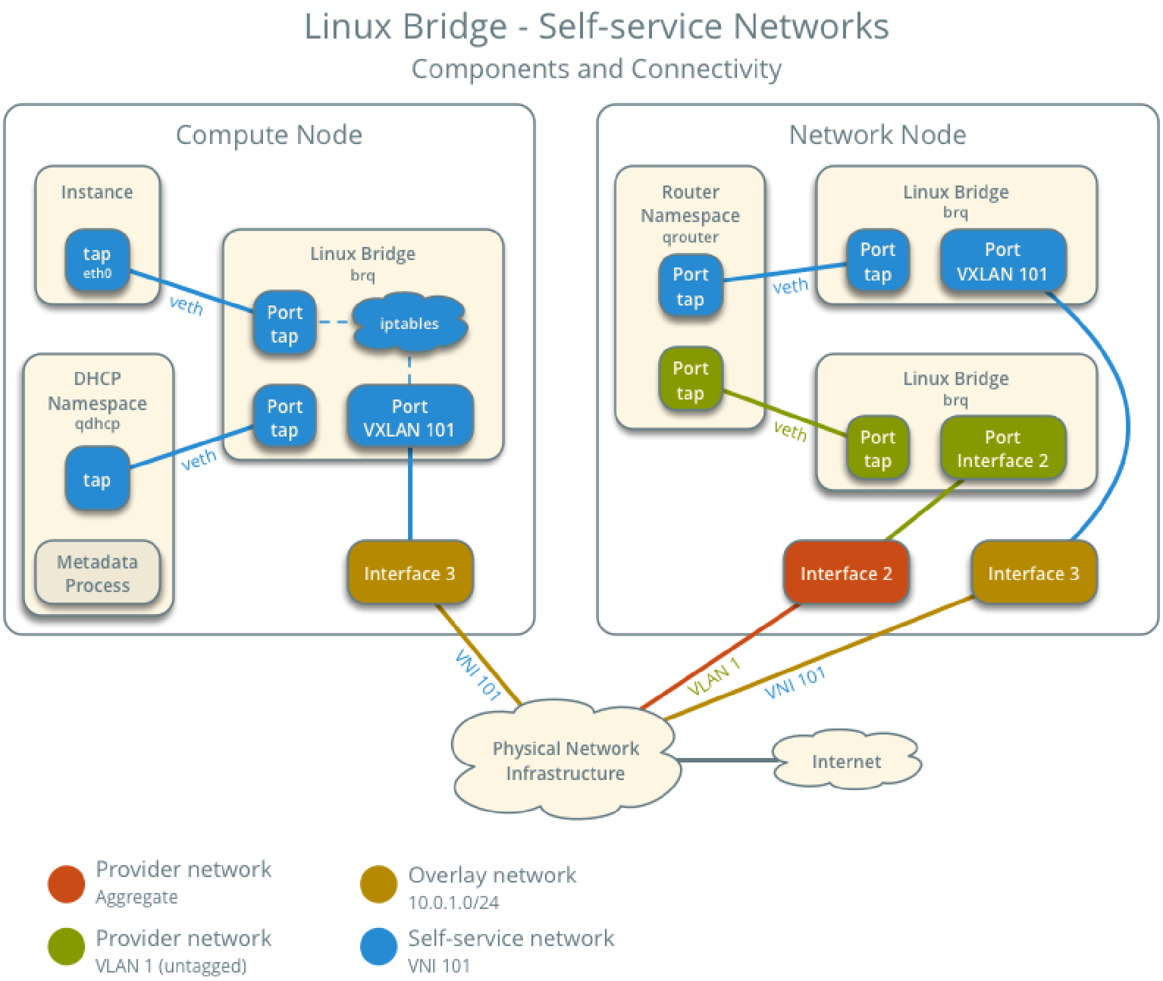
- Add one network interface: overlay.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network node.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) provider network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the overlay physical network interface.



Example configuration

Use the following example configuration as a template to add support for self-service networks to an existing operational environment that supports provider networks.

Controller node

1. In the neutron.conf file:

   - Enable routing and allow overlapping IP address ranges.

```
[DEFAULT]
service_plugins = router
allow_overlapping_ips = True
```

2. In the ml2_conf.ini file:

   - Add vxlan to type drivers and project network types.

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
```

- Enable the layer-2 population mechanism driver.

```
[ml2]
mechanism_drivers = linuxbridge,l2population
```

- Configure the VXLAN network ID (VNI) range.

```
[ml2_type_vxlan]
vni_ranges = VNI_START:VNI_END
```

  Replace `VNI_START` and `VNI_END` with appropriate numerical values.

3. Restart the following services:
   - Server

Network node

1. Install the Networking service layer-3 agent.
2. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

3. In the `linuxbridge_agent.ini` file, configure the layer-2 agent.

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE

[vxlan]
enable_vxlan = True
l2_population = True
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
```

(continues on next page)

(continued from previous page)

```
[securitygroup]
firewall_driver = iptables
```

Warning: By default, Linux uses UDP port `8472` for VXLAN tunnel traffic. This default value doesnt follow the IANA standard, which assigned UDP port `4789` for VXLAN communication. As a consequence, if this node is part of a mixed deployment, where nodes with both OVS and Linux bridge must communicate over VXLAN tunnels, it is recommended that a line containing `udp_dstport = 4789` be added to the [vxlan] section of all the Linux bridge agents. OVS follows the IANA standard.

Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

4. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = linuxbridge
```

5. Start the following services:

   - Linux bridge agent
   - Layer-3 agent

Compute nodes

1. In the `linuxbridge_agent.ini` file, enable VXLAN support including layer-2 population.

```
[vxlan]
enable_vxlan = True
l2_population = True
local_ip = OVERLAY_INTERFACE_IP_ADDRESS
```

Warning: By default, Linux uses UDP port `8472` for VXLAN tunnel traffic. This default value doesnt follow the IANA standard, which assigned UDP port `4789` for VXLAN communication. As a consequence, if this node is part of a mixed deployment, where nodes with both OVS and Linux bridge must communicate over VXLAN tunnels, it is recommended that a line containing `udp_dstport = 4789` be added to the [vxlan] section of all the Linux bridge agents. OVS follows the IANA standard.

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

2. Restart the following services:

   - Linux bridge agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host    ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent |␣
↪compute2 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
| 188945d1-9e70-4803-a276-df924e0788a4 | Linux bridge agent |␣
↪compute1 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
| e76c440d-d5f6-4316-a674-d689630b629e | DHCP agent         |␣
↪compute1 | nova              | True  | UP    | neutron-dhcp-agent  ␣
↪      |
| e67367de-6657-11e6-86a4-931cd04404bb | DHCP agent         |␣
↪compute2 | nova              | True  | UP    | neutron-dhcp-agent  ␣
↪      |
| e8174cae-6657-11e6-89f0-534ac6d0cb5c | Metadata agent     |␣
↪compute1 | None              | True  | UP    | neutron-metadata-
↪agent    |
| ece49ec6-6657-11e6-bafb-c7560f19197d | Metadata agent     |␣
↪compute2 | None              | True  | UP    | neutron-metadata-
↪agent    |
| 598f6357-4331-4da5-a420-0f5be000bec9 | L3 agent           |␣
↪network1 | nova              | True  | UP    | neutron-l3-agent    ␣
↪      |
| f4734e0f-bcd5-4922-a19d-e31d56b0a7ae | Linux bridge agent |␣
↪network1 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
```

Create initial networks

The configuration supports multiple VXLAN self-service networks. For simplicity, the following procedure creates one self-service network and a router with a gateway on the flat provider network. The router uses NAT for IPv4 network traffic and directly routes IPv6 network traffic.

Note: IPv6 connectivity with self-service networks often requires addition of static routes to nodes and physical network infrastructure.

1. Source the administrative project credentials.
2. Update the provider network to support external connectivity for self-service networks.

```
$ openstack network set --external provider1
```

Note: This command provides no output.

3. Source a regular (non-administrative) project credentials.
4. Create a self-service network.

```
$ openstack network create selfservice1
+-------------------------+--------------+
| Field                   | Value        |
+-------------------------+--------------+
| admin_state_up          | UP           |
| mtu                     | 1450         |
| name                    | selfservice1 |
| port_security_enabled   | True         |
| router:external         | Internal     |
| shared                  | False        |
| status                  | ACTIVE       |
+-------------------------+--------------+
```

5. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
  --network selfservice1 --dns-nameserver 8.8.4.4 selfservice1-v4
+-------------------+---------------------------+
| Field             | Value                     |
+-------------------+---------------------------+
| allocation_pools  | 192.0.2.2-192.0.2.254     |
| cidr              | 192.0.2.0/24              |
| dns_nameservers   | 8.8.4.4                   |
| enable_dhcp       | True                      |
| gateway_ip        | 192.0.2.1                 |
| ip_version        | 4                         |
| name              | selfservice1-v4           |
+-------------------+---------------------------+
```

6. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-
↪version 6 \
  --ipv6-ra-mode slaac --ipv6-address-mode slaac --network␣
↪selfservice1 \
  --dns-nameserver 2001:4860:4860::8844 selfservice1-v6
+-------------------+-------------------------------------------------------
↪------+
| Field             | Value                                                ␣
↪      |
+-------------------+-------------------------------------------------------
↪------+
| allocation_pools  | fd00:192:0:2::2-
↪fd00:192:0:2:ffff:ffff:ffff:ffff      |
| cidr              | fd00:192:0:2::/64                                    ␣
↪      |
| dns_nameservers   | 2001:4860:4860::8844                                 ␣
↪      |
```

(continues on next page)

(continued from previous page)

```
| enable_dhcp       | True                                          ␣
↪     |
| gateway_ip        | fd00:192:0:2::1                               ␣
↪     |
| ip_version        | 6                                             ␣
↪     |
| ipv6_address_mode | slaac                                         ␣
↪     |
| ipv6_ra_mode      | slaac                                         ␣
↪     |
| name              | selfservice1-v6                               ␣
↪     |
+-------------------+-----------------------------------------------
↪-----+
```

7. Create a router.

```
$ openstack router create router1
+-----------------------+---------+
| Field                 | Value   |
+-----------------------+---------+
| admin_state_up        | UP      |
| name                  | router1 |
| status                | ACTIVE  |
+-----------------------+---------+
```

8. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router1 selfservice1-v4
$ openstack router add subnet router1 selfservice1-v6
```

Note: These commands provide no output.

9. Add the provider network as the gateway on the router.

```
$ openstack router set --external-gateway provider1 router1
```

Verify network operation

1. On each compute node, verify creation of a second qdhcp namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
qdhcp-8fbc13ca-cfe0-4b8a-993b-e33f37ba66d1
```

2. On the network node, verify creation of the qrouter namespace.

```
# ip netns
qrouter-17db2a15-e024-46d0-9250-4cd4d336a2cc
```

3. Source a regular (non-administrative) project credentials.

4. Create the appropriate security group rules to allow `ping` and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| protocol         | icmp      |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-
↪icmp default
+-----------+-----------+
| Field     | Value     |
+-----------+-----------+
| direction | ingress   |
| ethertype | IPv6      |
| protocol  | ipv6-icmp |
+-----------+-----------+

$ openstack security group rule create --proto tcp --dst-port 22␣
↪default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto tcp --
↪dst-port 22 default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv6      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
+------------------+-----------+
```

5. Launch an instance with an interface on the self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-
↪id=NETWORK_ID selfservice-instance1
```

Replace `NETWORK_ID` with the ID of the self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+-----------------------+------
↪--+--------------------------------------------------------------+------
↪--+---------+
| ID                                   | Name                  |␣
↪Status | Networks                                                     |␣
↪Image  | Flavor  |
+--------------------------------------+-----------------------+------
↪--+--------------------------------------------------------------+------
↪--+---------+
| c055cdb0-ebb4-4d65-957c-35cbdbd59306 | selfservice-instance1 |␣
↪ACTIVE | selfservice1=192.0.2.4, fd00:192:0:2:f816:3eff:fe30:9cb0 |␣
↪cirros | m1.tiny |
+--------------------------------------+-----------------------+------
↪--+--------------------------------------------------------------+------
↪--+---------+
```

> **Warning:** The IPv4 address resides in a private IP address range (RFC1918). Thus, the Networking service performs source network address translation (SNAT) for the instance to access external networks such as the Internet. Access from external networks such as the Internet to the instance requires a floating IPv4 address. The Networking service performs destination network address translation (DNAT) from the floating IPv4 address to the instance IPv4 address on the self-service network. On the other hand, the Networking service architecture for IPv6 lacks support for NAT due to the significantly larger address space and complexity of NAT. Thus, floating IP addresses do not exist for IPv6 and the Networking service only performs routing for IPv6 subnets on self-service networks. In other words, you cannot rely on NAT to hide instances with IPv4 and IPv6 addresses or only IPv6 addresses and must properly implement security groups to restrict access.

7. On the controller node or any host with access to the provider network, `ping` the IPv6 address of the instance.

```
$ ping6 -c 4 fd00:192:0:2:f816:3eff:fe30:9cb0
PING␣
↪fd00:192:0:2:f816:3eff:fe30:9cb0(fd00:192:0:2:f816:3eff:fe30:9cb0)␣
↪56 data bytes
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=1 ttl=63␣
↪time=2.08 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=2 ttl=63␣
↪time=1.88 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=3 ttl=63␣
↪time=1.55 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=4 ttl=63␣
↪time=1.62 ms

--- fd00:192:0:2:f816:3eff:fe30:9cb0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.557/1.788/2.085/0.217 ms
```

8. Optionally, enable IPv4 access from external networks such as the Internet to the instance.

    1. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| fixed_ip    | None                                 |
| id          | 22a1b088-5c9b-43b4-97f3-970ce5df77f2 |
| instance_id | None                                 |
| ip          | 203.0.113.16                         |
| pool        | provider1                            |
+-------------+--------------------------------------+
```

2. Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance1 203.0.
↪113.16
```

Note: This command provides no output.

3. On the controller node or any host with access to the provider network, `ping` the floating IPv4 address of the instance.

```
$ ping -c 4 203.0.113.16
PING 203.0.113.16 (203.0.113.16) 56(84) bytes of data.
64 bytes from 203.0.113.16: icmp_seq=1 ttl=63 time=3.41 ms
64 bytes from 203.0.113.16: icmp_seq=2 ttl=63 time=1.67 ms
64 bytes from 203.0.113.16: icmp_seq=3 ttl=63 time=1.47 ms
64 bytes from 203.0.113.16: icmp_seq=4 ttl=63 time=1.59 ms

--- 203.0.113.16 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.473/2.040/3.414/0.798 ms
```

9. Obtain access to the instance.

10. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
  - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
  - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
  - VXLAN ID (VNI) 102

- Self-service router
  - Gateway on the provider network
  - Interface on self-service network 1
  - Interface on self-service network 2
- Instance 1
- Instance 2

North-south scenario 1: Instance with a fixed IP address

For instances with a fixed IPv4 address, the network node performs SNAT on north-south traffic passing from self-service to external networks such as the Internet. For instances with a fixed IPv6 address, the network node performs conventional routing of traffic between self-service and external networks.

- The instance resides on compute node 1 and uses self-service network 1.
- The instance sends a packet to a host on the Internet.

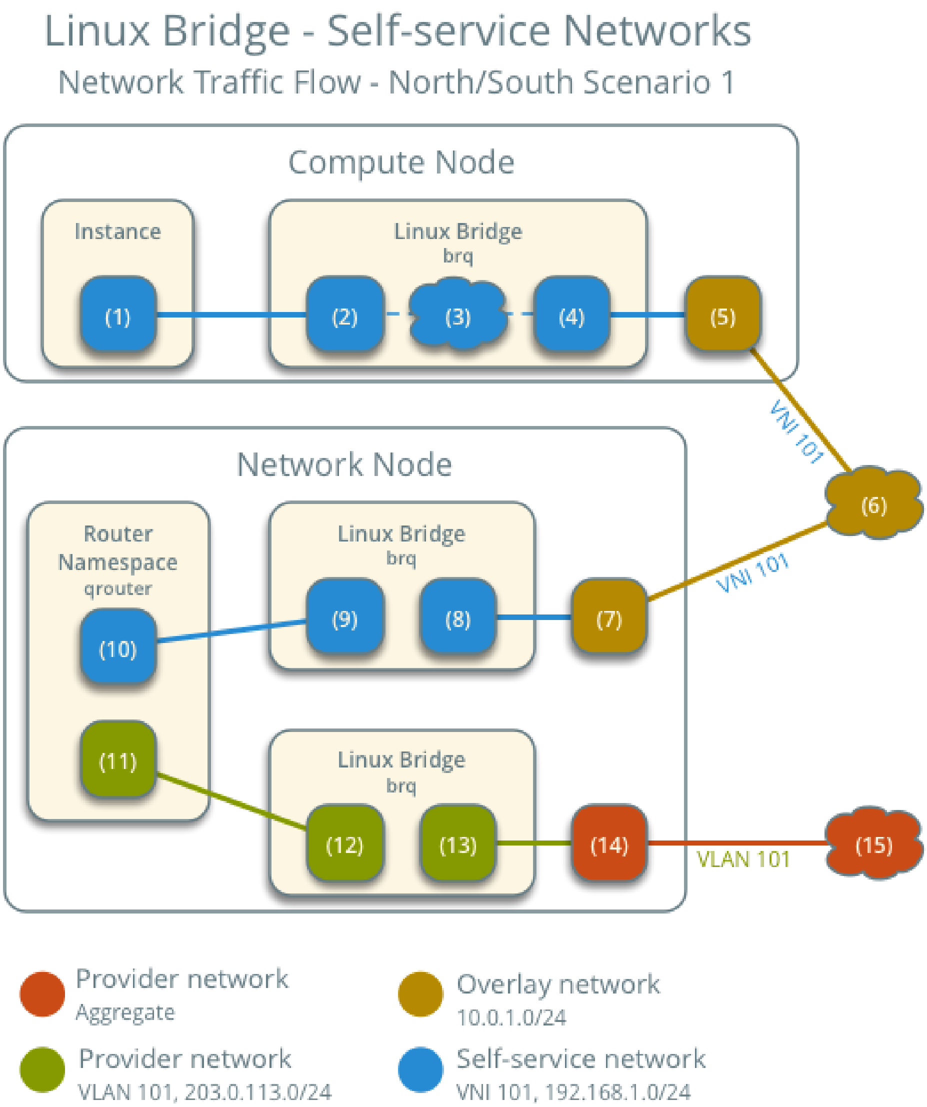
The following steps involve compute node 1:

1. The instance interface (1) forwards the packet to the self-service bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to the network node via the overlay network (6).

The following steps involve the network node:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. The self-service bridge router port (9) forwards the packet to the self-service network interface (10) in the router namespace.
   - For IPv4, the router performs SNAT on the packet which changes the source IP address to the router IP address on the provider network and sends it to the gateway IP address on the provider network via the gateway interface on the provider network (11).
   - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the provider network, via the provider gateway interface (11).
3. The router forwards the packet to the provider bridge router port (12).
4. The VLAN sub-interface port (13) on the provider bridge forwards the packet to the provider physical network interface (14).
5. The provider physical network interface (14) adds VLAN tag 101 to the packet and forwards it to the Internet via physical network infrastructure (15).

Note: Return traffic follows similar steps in reverse. However, without a floating IPv4 address, hosts on the provider or external networks cannot originate connections to instances on the self-service network.



North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address, the network node performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario.

- The instance resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

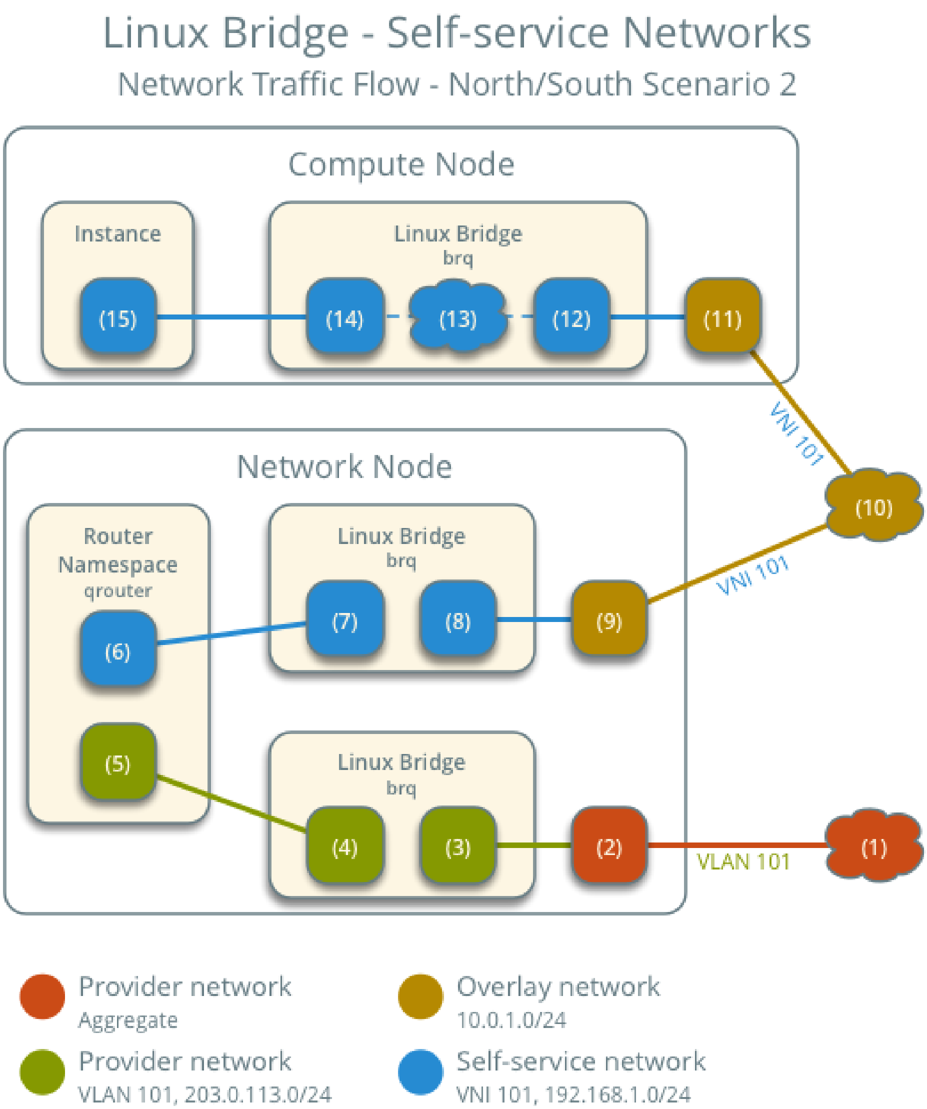
The following steps involve the network node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface (2).
2. The provider physical network interface removes VLAN tag 101 and forwards the packet to the VLAN sub-interface on the provider bridge.
3. The provider bridge forwards the packet to the self-service router gateway port on the provider network (5).
   - For IPv4, the router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network and sends it to the gateway IP address on the self-service network via the self-service interface (6).
   - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the self-service network, via the self-service interface (6).
4. The router forwards the packet to the self-service bridge router port (7).
5. The self-service bridge forwards the packet to the VXLAN interface (8) which wraps the packet using VNI 101.
6. The underlying physical interface (9) for the VXLAN interface forwards the packet to the network node via the overlay network (10).

The following steps involve the compute node:

1. The underlying physical interface (11) for the VXLAN interface forwards the packet to the VXLAN interface (12) which unwraps the packet.
2. Security group rules (13) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (14) forwards the packet to the instance interface (15) via `veth` pair.

Note: Egress instance traffic flows similar to north-south scenario 1, except SNAT changes the source IP address of the packet to the floating IPv4 address rather than the router IP address on the provider network.



East-west scenario 1: Instances on the same network

Instances with a fixed IPv4/IPv6 or floating IPv4 address on the same network communicate directly between compute nodes containing those instances.

By default, the VXLAN protocol lacks knowledge of target location and uses multicast to discover it. After discovery, it stores the location in the local forwarding database. In large deployments, the discovery process can generate a significant amount of network that all nodes must process. To eliminate the latter and generally increase efficiency, the Networking service includes the layer-2 population mechanism driver that automatically populates the forwarding database for VXLAN interfaces. The example configuration enables this driver. For more information, see *ML2 plug-in*.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 2 and uses self-service network 1.
- Instance 1 sends a packet to instance 2.

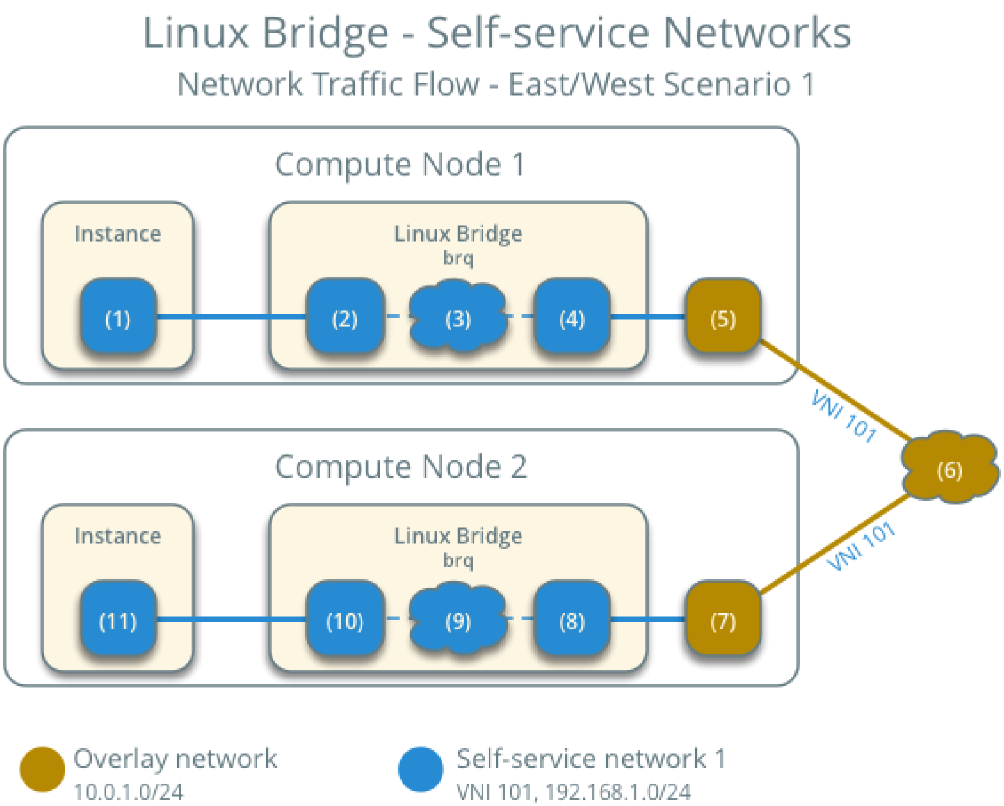
The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the self-service bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to compute node 2 via the overlay network (6).

The following steps involve compute node 2:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. Security group rules (9) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (10) forwards the packet to the instance 1 interface (11) via `veth` pair.

Note: Return traffic follows similar steps in reverse.



East-west scenario 2: Instances on different networks

Instances using a fixed IPv4/IPv6 address or floating IPv4 address communicate via router on the network node. The self-service networks must reside on the same router.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 1 and uses self-service network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VXLAN enables multiple overlays to use the same layer-3 network.

The following steps involve the compute node:

1. The instance 1 interface (1) forwards the packet to the self-service bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to the network node via the overlay network (6).

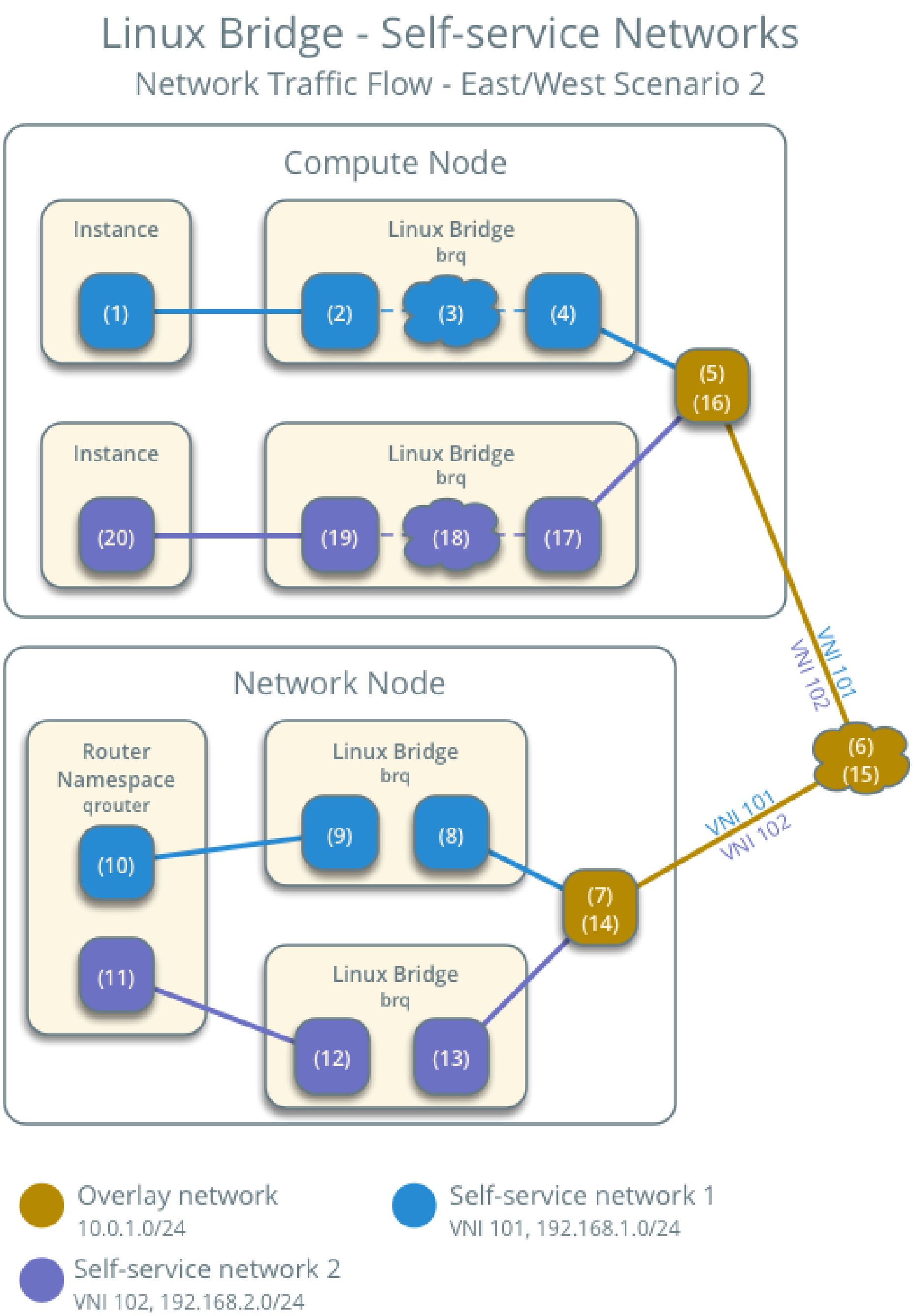
The following steps involve the network node:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. The self-service bridge router port (9) forwards the packet to the self-service network 1 interface (10) in the router namespace.
3. The router sends the packet to the next-hop IP address, typically the gateway IP address on self-service network 2, via the self-service network 2 interface (11).
4. The router forwards the packet to the self-service network 2 bridge router port (12).
5. The self-service network 2 bridge forwards the packet to the VXLAN interface (13) which wraps the packet using VNI 102.
6. The physical network interface (14) for the VXLAN interface sends the packet to the compute node via the overlay network (15).

The following steps involve the compute node:

1. The underlying physical interface (16) for the VXLAN interface sends the packet to the VXLAN interface (17) which unwraps the packet.
2. Security group rules (18) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (19) forwards the packet to the instance 2 interface (20) via `veth` pair.

Note: Return traffic follows similar steps in reverse.



Linux bridge: High availability using VRRP

This architecture example augments the self-service deployment example with a high-availability mechanism using the Virtual Router Redundancy Protocol (VRRP) via `keepalived` and provides failover of routing for self-service networks. It requires a minimum of two network nodes because VRRP creates one master (active) instance and at least one backup instance of each router.

During normal operation, `keepalived` on the master router periodically transmits *heartbeat* packets over a hidden network that connects all VRRP routers for a particular project. Each project with VRRP routers uses a separate hidden network. By default this network uses the first value in the `tenant_network_types` option in the `ml2_conf.ini` file. For additional control, you can specify the self-service network type and physical network name for the hidden network using the `l3_ha_network_type` and `l3_ha_network_name` options in the `neutron.conf` file.

If `keepalived` on the backup router stops receiving *heartbeat* packets, it assumes failure of the master router and promotes the backup router to master router by configuring IP addresses on the interfaces in the `qrouter` namespace. In environments with more than one backup router, `keepalived` on the backup router with the next highest priority promotes that backup router to master router.

Note: This high-availability mechanism configures VRRP using the same priority for all routers. Therefore, VRRP promotes the backup router with the highest IP address to the master router.

Warning: There is a known bug with `keepalived` v1.2.15 and earlier which can cause packet loss when `max_l3_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to `keepalived` v1.2.16 or greater when using this feature.

Interruption of VRRP *heartbeat* traffic between network nodes, typically due to a network interface or physical network infrastructure failure, triggers a failover. Restarting the layer-3 agent, or failure of it, does not trigger a failover providing `keepalived` continues to operate.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Instance network traffic on self-service networks using a particular router only traverses the master instance of that router. Thus, resource limitations of a particular network node can impact all master instances of routers on that network node without triggering failover to another network node. However, you can configure the scheduler to distribute the master instance of each router uniformly across a pool of network nodes to reduce the chance of resource contention on any particular network node.
- Only supports self-service networks using a router. Provider networks operate at layer-2 and rely on physical network infrastructure for redundancy.
- For instances with a floating IPv4 address, maintains state of network connections during failover as a side effect of 1:1 static NAT. The mechanism does not actually implement connection tracking.

For production deployments, we recommend at least three network nodes with sufficient resources to handle network traffic for the entire environment if one network node fails. Also, the remaining two nodes can continue to provide redundancy.

Warning: This high-availability mechanism is not compatible with the layer-2 population mechanism. You must disable layer-2 population in the `linuxbridge_agent.ini` file and restart the Linux bridge agent on all existing network and compute nodes prior to deploying the example configuration.

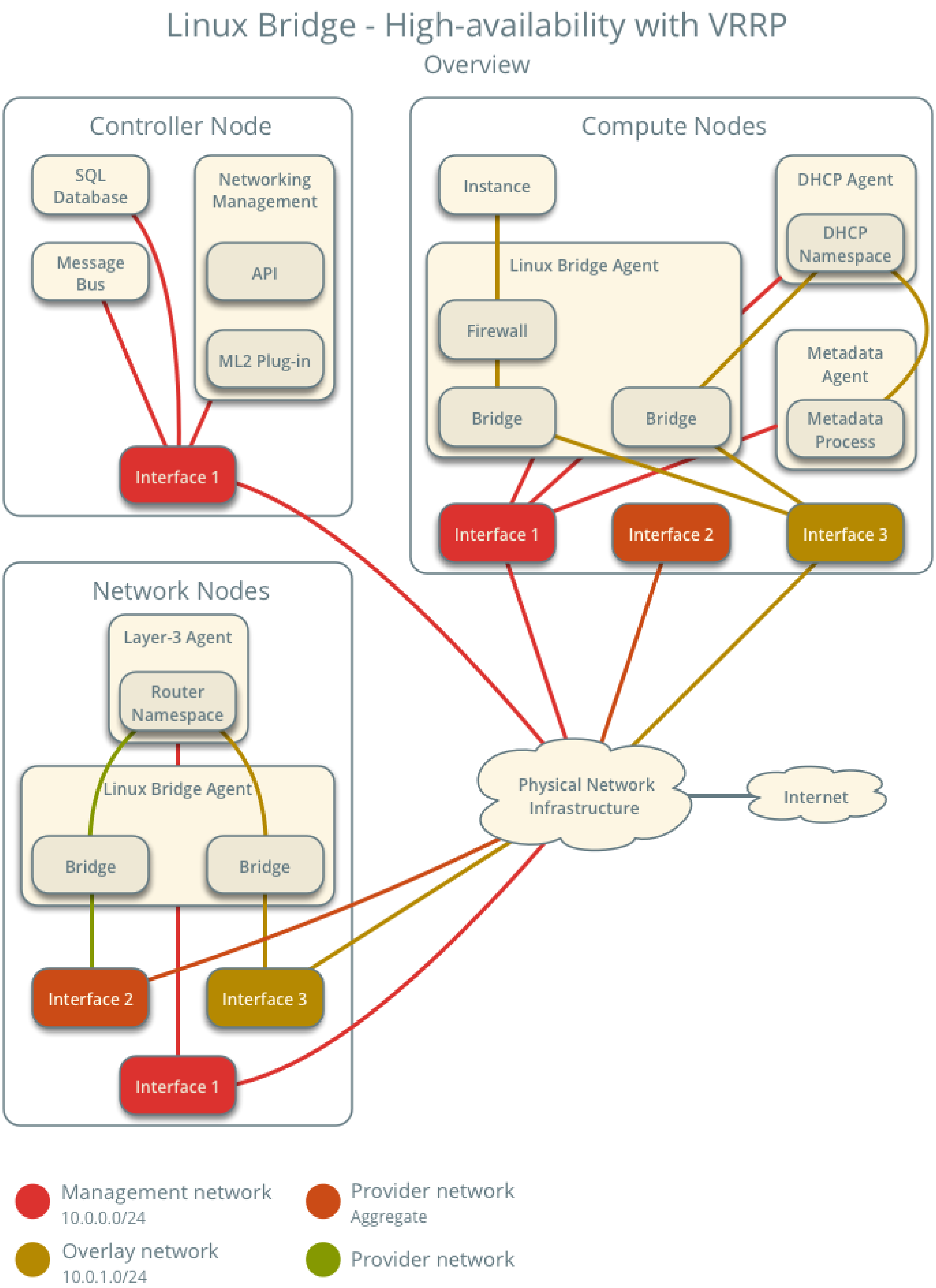
Prerequisites

Add one network node with the following components:

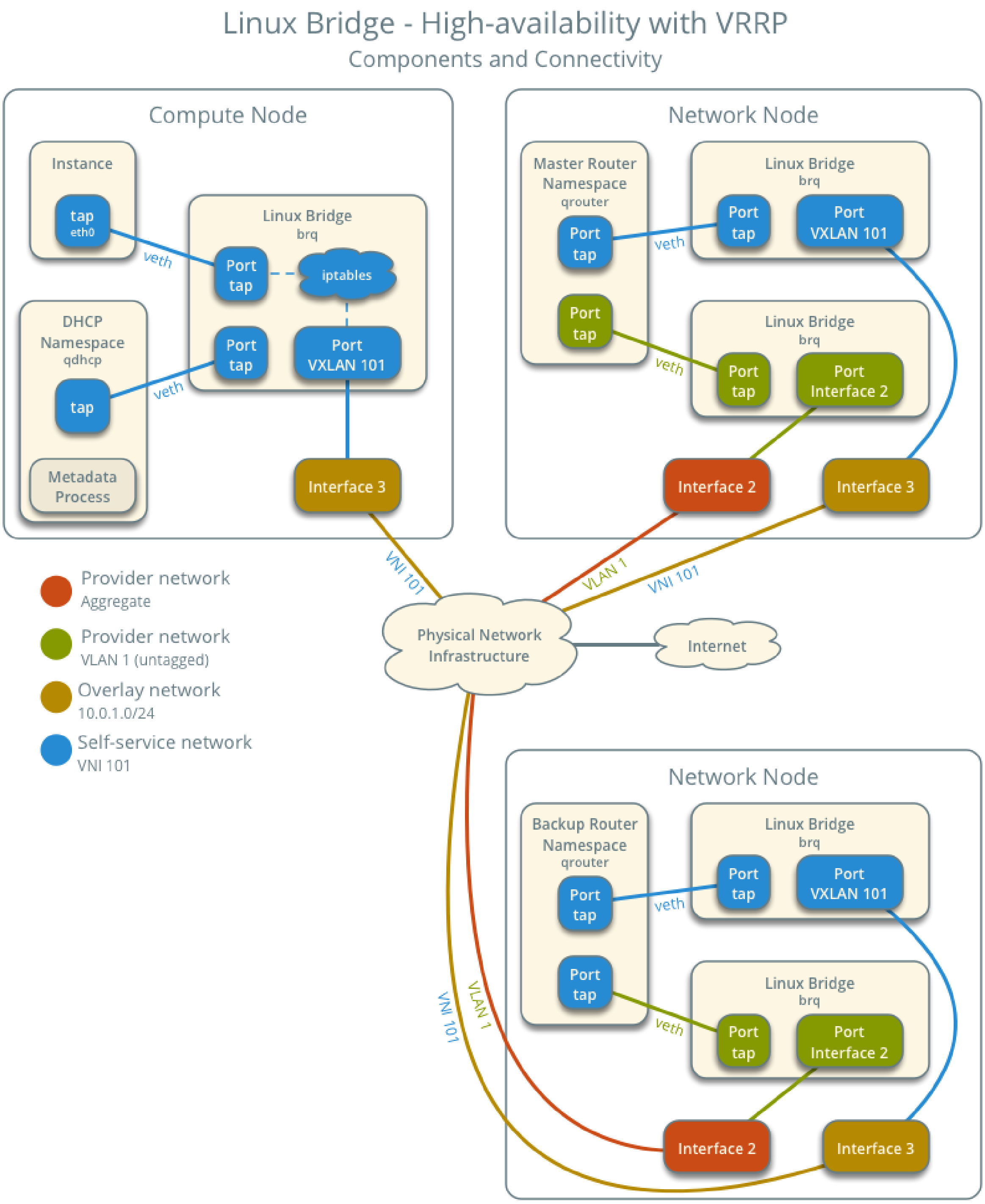
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking layer-2 agent, layer-3 agent, and any dependencies.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network nodes.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) network. The master router resides on network node 1. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the overlay physical network interface.



Example configuration

Use the following example configuration as a template to add support for high-availability using VRRP to an existing operational environment that supports self-service networks.

Controller node

1. In the neutron.conf file:

   - Enable VRRP.

   ```
   [DEFAULT]
   l3_ha = True
   ```

2. Restart the following services:

   - Server

Network node 1

No changes.

Network node 2

1. Install the Networking service Linux bridge layer-2 agent and layer-3 agent.

2. In the neutron.conf file, configure common options:

   ```
   [DEFAULT]
   core_plugin = ml2
   auth_strategy = keystone

   [database]
   # ...

   [keystone_authtoken]
   # ...

   [nova]
   # ...

   [agent]
   # ...
   ```

   See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_authtoken], [nova], and [agent] sections.

3. In the linuxbridge_agent.ini file, configure the layer-2 agent.

   ```
   [linux_bridge]
   physical_interface_mappings = provider:PROVIDER_INTERFACE
   ```

(continues on next page)

(continued from previous page)

```
[vxlan]
enable_vxlan = True
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[securitygroup]
firewall_driver = iptables
```

Warning: By default, Linux uses UDP port `8472` for VXLAN tunnel traffic. This default value doesnt follow the IANA standard, which assigned UDP port `4789` for VXLAN communication. As a consequence, if this node is part of a mixed deployment, where nodes with both OVS and Linux bridge must communicate over VXLAN tunnels, it is recommended that a line containing `udp_dstport = 4789` be added to the [vxlan] section of all the Linux bridge agents. OVS follows the IANA standard.

Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

4. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = linuxbridge
```

5. Start the following services:
   - Linux bridge agent
   - Layer-3 agent

Compute nodes

No changes.

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host    ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent |␣
↪compute2 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
```

(continues on next page)

(continued from previous page)

```
| 188945d1-9e70-4803-a276-df924e0788a4 | Linux bridge agent |␣
↪compute1 | None                 | True   | UP    | neutron-linuxbridge-
↪agent |
| e76c440d-d5f6-4316-a674-d689630b629e | DHCP agent         |␣
↪compute1 | nova                 | True   | UP    | neutron-dhcp-agent   ␣
↪      |
| e67367de-6657-11e6-86a4-931cd04404bb | DHCP agent         |␣
↪compute2 | nova                 | True   | UP    | neutron-dhcp-agent   ␣
↪      |
| e8174cae-6657-11e6-89f0-534ac6d0cb5c | Metadata agent     |␣
↪compute1 | None                 | True   | UP    | neutron-metadata-
↪agent    |
| ece49ec6-6657-11e6-bafb-c7560f19197d | Metadata agent     |␣
↪compute2 | None                 | True   | UP    | neutron-metadata-
↪agent    |
| 598f6357-4331-4da5-a420-0f5be000bec9 | L3 agent           |␣
↪network1 | nova                 | True   | UP    | neutron-l3-agent     ␣
↪      |
| f4734e0f-bcd5-4922-a19d-e31d56b0a7ae | Linux bridge agent |␣
↪network1 | None                 | True   | UP    | neutron-linuxbridge-
↪agent |
| 670e5805-340b-4182-9825-fa8319c99f23 | Linux bridge agent |␣
↪network2 | None                 | True   | UP    | neutron-linuxbridge-
↪agent |
| 96224e89-7c15-42e9-89c4-8caac7abdd54 | L3 agent           |␣
↪network2 | nova                 | True   | UP    | neutron-l3-agent     ␣
↪      |
+--------------------------------------+--------------------+---------
↪-+----------------------+-------+-------+---------------------------+
```

Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use VRRP. The following procedure creates an additional self-service network and router. The Networking service also supports adding high-availability to existing routers. However, the procedure requires administratively disabling and enabling each router which temporarily interrupts network connectivity for self-service networks with interfaces on that router.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

```
$ openstack network create selfservice2
+---------------------------+--------------+
| Field                     | Value        |
+---------------------------+--------------+
| admin_state_up            | UP           |
| mtu                       | 1450         |
| name                      | selfservice2 |
| port_security_enabled     | True         |
| router:external           | Internal     |
| shared                    | False        |
```

(continues on next page)

(continued from previous page)

```
| status                    | ACTIVE         |
+---------------------------+----------------+
```

3. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 198.51.100.0/24 \
  --network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4
+-------------------+------------------------------+
| Field             | Value                        |
+-------------------+------------------------------+
| allocation_pools  | 198.51.100.2-198.51.100.254  |
| cidr              | 198.51.100.0/24              |
| dns_nameservers   | 8.8.4.4                      |
| enable_dhcp       | True                         |
| gateway_ip        | 198.51.100.1                 |
| ip_version        | 4                            |
| name              | selfservice2-v4              |
+-------------------+------------------------------+
```

4. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:198:51:100::/64 --ip-
↪version 6 \
  --ipv6-ra-mode slaac --ipv6-address-mode slaac --network␣
↪selfservice2 \
  --dns-nameserver 2001:4860:4860::8844 selfservice2-v6
+-------------------+--------------------------------------------------
↪-------+
| Field             | Value                                            ␣
↪       |
+-------------------+--------------------------------------------------
↪-------+
| allocation_pools  | fd00:198:51:100::2-
↪fd00:198:51:100:ffff:ffff:ffff:ffff |
| cidr              | fd00:198:51:100::/64                             ␣
↪       |
| dns_nameservers   | 2001:4860:4860::8844                             ␣
↪       |
| enable_dhcp       | True                                             ␣
↪       |
| gateway_ip        | fd00:198:51:100::1                               ␣
↪       |
| ip_version        | 6                                                ␣
↪       |
| ipv6_address_mode | slaac                                            ␣
↪       |
| ipv6_ra_mode      | slaac                                            ␣
↪       |
| name              | selfservice2-v6                                  ␣
↪       |
+-------------------+--------------------------------------------------
↪-------+
```

5. Create a router.

```
$ openstack router create router2
+-----------------------+---------+
| Field                 | Value   |
+-----------------------+---------+
| admin_state_up        | UP      |
| name                  | router2 |
| status                | ACTIVE  |
+-----------------------+---------+
```

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4
$ openstack router add subnet router2 selfservice2-v6
```

Note: These commands provide no output.

7. Add the provider network as a gateway on the router.

```
$ openstack router set --external-gateway provider1 router2
```

Verify network operation

1. Source the administrative project credentials.
2. Verify creation of the internal high-availability network that handles VRRP *heartbeat* traffic.

```
$ openstack network list
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
| ID                                   | Name                       ␣
↪                      | Subnets                              |
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
| 1b8519c1-59c4-415c-9da2-a67d53c68455 | HA network tenant␣
↪f986edf55ae945e2bef3cb4bfd589928 | 6843314a-1e76-4cc9-94f5-
↪c64b7a39364a |
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
```

3. On each network node, verify creation of a `qrouter` namespace with the same ID.

Network node 1:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Network node 2:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Note: The namespace for router 1 from *Linux bridge: Self-service networks* should only appear on network node 1 because of creation prior to enabling VRRP.

4. On each network node, show the IP address of interfaces in the `qrouter` namespace. With the exception of the VRRP interface, only one namespace belonging to the master router instance contains IP addresses on the interfaces.

   Network node 1:

```
# ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr␣
↪show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN␣
↪group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ha-eb820380-40@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:78:ba:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 169.254.192.1/18 brd 169.254.255.255 scope global ha-
↪eb820380-40
       valid_lft forever preferred_lft forever
    inet 169.254.0.1/24 scope global ha-eb820380-40
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe78:ba99/64 scope link
       valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 198.51.100.1/24 scope global qr-da3504ad-ba
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fedc:8ea8/64 scope link
       valid_lft forever preferred_lft forever
4: qr-442e36eb-fc@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:ee:c8:41 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fd00:198:51:100::1/64 scope global nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:feee:c841/64 scope link
       valid_lft forever preferred_lft forever
5: qg-33fedbc5-43@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 203.0.113.21/24 scope global qg-33fedbc5-43
       valid_lft forever preferred_lft forever
    inet6 fd00:203:0:113::21/64 scope global nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe03:1af6/64 scope link
       valid_lft forever preferred_lft forever
```

   Network node 2:

```
# ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr␣
↪show
```

(continues on next page)

(continued from previous page)

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN␣
↪group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ha-7a7ce184-36@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:16:59:84 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 169.254.192.2/18 brd 169.254.255.255 scope global ha-
↪7a7ce184-36
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe16:5984/64 scope link
       valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
4: qr-442e36eb-fc@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
5: qg-33fedbc5-43@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Note: The master router may reside on network node 2.

5. Launch an instance with an interface on the additional self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-
↪id=NETWORK_ID selfservice-instance2
```

Replace NETWORK_ID with the ID of the additional self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+-----------------------+-------
↪--+---------------------------------------------------------------
↪+---------+---------+
| ID                                   | Name                  |␣
↪Status | Networks                                                  ␣
↪       | Image   | Flavor  |
+--------------------------------------+-----------------------+-------
↪--+---------------------------------------------------------------
↪+---------+---------+
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 |␣
↪ACTIVE | selfservice2=fd00:198:51:100:f816:3eff:fe71:e93e, 198.51.
↪100.4 | cirros | m1.tiny |
+--------------------------------------+-----------------------+-------
↪--+---------------------------------------------------------------
↪+---------+---------+
```

7. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| fixed_ip    | None                                 |
| id          | 0174056a-fa56-4403-b1ea-b5151a31191f |
| instance_id | None                                 |
| ip          | 203.0.113.17                         |
| pool        | provider1                            |
+-------------+--------------------------------------+
```

8. Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

Note: This command provides no output.

Verify failover operation

1. Begin a continuous `ping` of both the floating IPv4 address and IPv6 address of the instance. While performing the next three steps, you should see a minimal, if any, interruption of connectivity to the instance.
2. On the network node with the master router, administratively disable the overlay network interface.
3. On the other network node, verify promotion of the backup router to master router by noting addition of IP addresses to the interfaces in the `qrouter` namespace.
4. On the original network node in step 2, administratively enable the overlay network interface. Note that the master router remains on the network node in step 3.

Keepalived VRRP health check

The health of your `keepalived` instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the `l3_agent.ini` file:

```
ha_vrrp_health_check_interval = 30
```

Where `ha_vrrp_health_check_interval` indicates how often in seconds the health check should run. The default value is `0`, which indicates that the check should not run at all.

Network traffic flow

This high-availability mechanism simply augments *Linux bridge: Self-service networks* with failover of layer-3 services to another router if the master router fails. Thus, you can reference *Self-service network traffic flow* for normal operation.

Open vSwitch mechanism driver

The Open vSwitch (OVS) mechanism driver uses a combination of OVS and Linux bridges as interconnection devices. However, optionally enabling the OVS native implementation of security groups removes the dependency on Linux bridges.

We recommend using Open vSwitch version 2.4 or higher. Optional features may require a higher minimum version.

Open vSwitch: Provider networks

This architecture example provides layer-2 connectivity between instances and the physical network infrastructure using VLAN (802.1q) tagging. It supports one untagged (flat) network and up to 4095 tagged (VLAN) networks. The actual quantity of VLAN networks depends on the physical network infrastructure. For more information on provider networks, see *Provider networks*.

> **Warning:** Linux distributions often package older releases of Open vSwitch that can introduce issues during operation with the Networking service. We recommend using at least the latest long-term stable (LTS) release of Open vSwitch for the best experience and support from Open vSwitch. See http://www.openvswitch.org for available releases and the installation instructions for more details.

Prerequisites

One controller node with the following components:

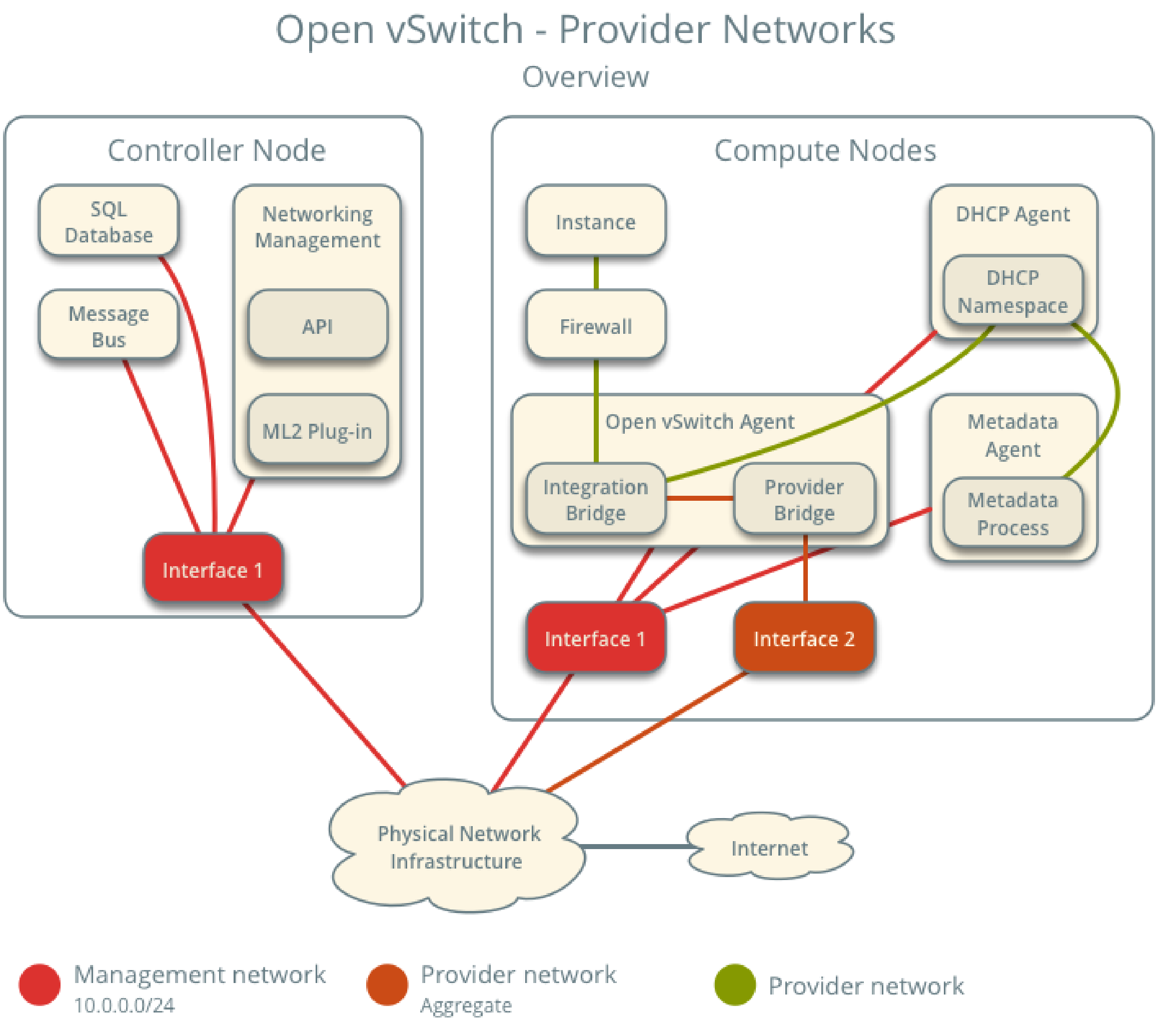
- Two network interfaces: management and provider.
- OpenStack Networking server service and ML2 plug-in.

Two compute nodes with the following components:

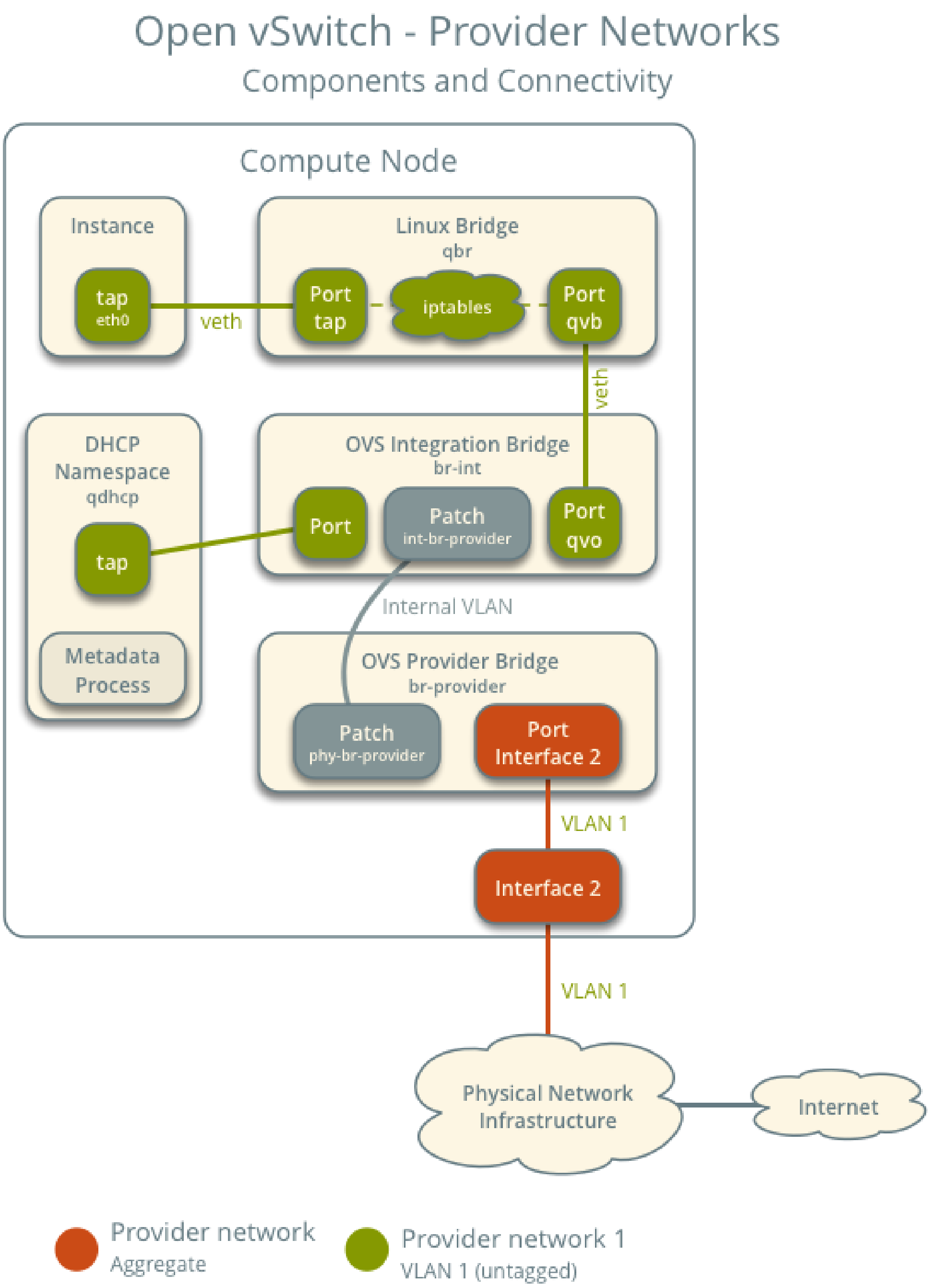
- Two network interfaces: management and provider.
- OpenStack Networking Open vSwitch (OVS) layer-2 agent, DHCP agent, metadata agent, and any dependencies including OVS.

Note: Larger deployments typically deploy the DHCP and metadata agents on a subset of compute nodes to increase performance and redundancy. However, too many agents can overwhelm the message bus. Also, to further simplify any deployment, you can omit the metadata agent and use a configuration drive to provide metadata to instances.

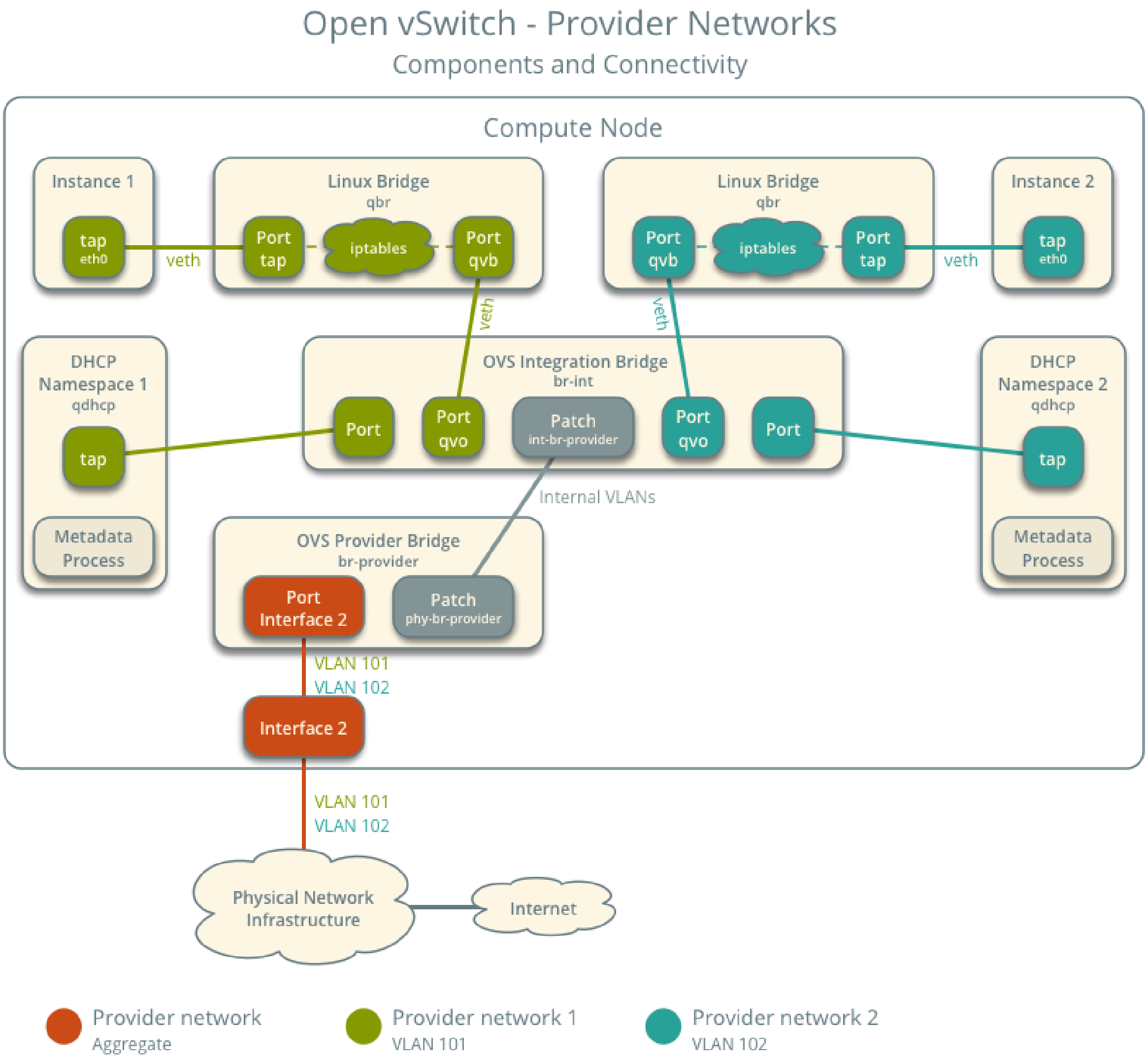
Architecture



The following figure shows components and connectivity for one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace with a port on the OVS integration bridge.



The following figure describes virtual connectivity among components for two tagged (VLAN) networks. Essentially, all networks use a single OVS integration bridge with different internal VLAN tags. The internal VLAN tags almost always differ from the network VLAN assignment in the Networking service. Similar to the untagged network case, the DHCP agent may reside on a different compute node.



Note: These figures omit the controller node because it does not handle instance network traffic.

Example configuration

Use the following example configuration as a template to deploy provider networks in your environment.

Controller node

1. Install the Networking service components that provide the `neutron-server` service and ML2 plug-in.
2. In the `neutron.conf` file:
   - Configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

- Disable service plug-ins because provider networks do not require any. However, this breaks portions of the dashboard that manage the Networking service. See the latest Install Tutorials and Guides for more information.

```
[DEFAULT]
service_plugins =
```

- Enable two DHCP agents per network so both compute nodes can provide DHCP service provider networks.

```
[DEFAULT]
dhcp_agents_per_network = 2
```

- If necessary, *configure MTU*.

3. In the `ml2_conf.ini` file:

- Configure drivers and network types:

```
[ml2]
type_drivers = flat,vlan
tenant_network_types =
mechanism_drivers = openvswitch
extension_drivers = port_security
```

- Configure network mappings:

```
[ml2_type_flat]
flat_networks = provider

[ml2_type_vlan]
network_vlan_ranges = provider
```

Note: The `tenant_network_types` option contains no value because the architecture does not support self-service networks.

Note: The `provider` value in the `network_vlan_ranges` option lacks VLAN ID ranges to support use of arbitrary VLAN IDs.

4. Populate the database.

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/
↪neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head"␣
↪neutron
```

5. Start the following services:

   - Server

Compute nodes

1. Install the Networking service OVS layer-2 agent, DHCP agent, and metadata agent.
2. Install OVS.
3. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

   See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

4. In the `openvswitch_agent.ini` file, configure the OVS agent:

```
[ovs]
bridge_mappings = provider:br-provider

[securitygroup]
firewall_driver = iptables_hybrid
```

5. In the `dhcp_agent.ini` file, configure the DHCP agent:

```
[DEFAULT]
interface_driver = openvswitch
```

(continues on next page)

(continued from previous page)

```
enable_isolated_metadata = True
force_metadata = True
```

Note: The `force_metadata` option forces the DHCP agent to provide a host route to the metadata service on `169.254.169.254` regardless of whether the subnet contains an interface on a router, thus maintaining similar and predictable metadata behavior among subnets.

6. In the `metadata_agent.ini` file, configure the metadata agent:

   ```
   [DEFAULT]
   nova_metadata_host = controller
   metadata_proxy_shared_secret = METADATA_SECRET
   ```

   The value of `METADATA_SECRET` must match the value of the same option in the `[neutron]` section of the `nova.conf` file.

7. Start the following services:

   - OVS

8. Create the OVS provider bridge `br-provider`:

   ```
   $ ovs-vsctl add-br br-provider
   ```

9. Add the provider network interface as a port on the OVS provider bridge `br-provider`:

   ```
   $ ovs-vsctl add-port br-provider PROVIDER_INTERFACE
   ```

   Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

10. Start the following services:

    - OVS agent
    - DHCP agent
    - Metadata agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents:

   ```
   $ openstack network agent list
   +--------------------------------------+--------------------+----------
   ↪-+-------------------+-------+-------+---------------------------+
   | ID                                   | Agent Type         | Host     ␣
   ↪ | Availability Zone | Alive | State | Binary                    |
   +--------------------------------------+--------------------+----------
   ↪-+-------------------+-------+-------+---------------------------+
   | 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent     |␣
   ↪compute2 | None              | True  | UP    | neutron-metadata-
   ↪agent    |
   ```

(continues on next page)

(continued from previous page)

```
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent |␣
↪compute2 | None                 | True  | UP    | neutron-openvswitch-
↪agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent         |␣
↪compute2 | nova                 | True  | UP    | neutron-dhcp-agent  ␣
↪      |
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent     |␣
↪compute1 | None                 | True  | UP    | neutron-metadata-
↪agent     |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent         |␣
↪compute1 | nova                 | True  | UP    | neutron-dhcp-agent  ␣
↪      |
| bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1 | Open vSwitch agent |␣
↪compute1 | None                 | True  | UP    | neutron-openvswitch-
↪agent |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
```

Create initial networks

The configuration supports one flat or multiple VLAN provider networks. For simplicity, the following procedure creates one flat provider network.

1. Source the administrative project credentials.
2. Create a flat network.

```
$ openstack network create --share --provider-physical-network␣
↪provider \
  --provider-network-type flat provider1
+---------------------------+-----------+-
| Field                     | Value     |
+---------------------------+-----------+
| admin_state_up            | UP        |
| mtu                       | 1500      |
| name                      | provider1 |
| port_security_enabled     | True      |
| provider:network_type     | flat      |
| provider:physical_network | provider  |
| provider:segmentation_id  | None      |
| router:external           | Internal  |
| shared                    | True      |
| status                    | ACTIVE    |
+---------------------------+-----------+
```

Note: The `share` option allows any project to use this network. To limit access to provider networks, see *Role-Based Access Control (RBAC)*.

Note: To create a VLAN network instead of a flat network, change `--provider-network-type flat` to `--provider-network-type vlan` and add `--provider-segment` with a value referencing the VLAN ID.

3. Create a IPv4 subnet on the provider network.

```
$ openstack subnet create --subnet-range 203.0.113.0/24 --gateway 203.
↪0.113.1 \
  --network provider1 --allocation-pool start=203.0.113.11,end=203.0.
↪113.250 \
  --dns-nameserver 8.8.4.4 provider1-v4
+-------------------+-----------------------------+
| Field             | Value                       |
+-------------------+-----------------------------+
| allocation_pools  | 203.0.113.11-203.0.113.250 |
| cidr              | 203.0.113.0/24              |
| dns_nameservers   | 8.8.4.4                     |
| enable_dhcp       | True                        |
| gateway_ip        | 203.0.113.1                 |
| ip_version        | 4                           |
| name              | provider1-v4                |
+-------------------+-----------------------------+
```

Important: Enabling DHCP causes the Networking service to provide DHCP which can interfere with existing DHCP services on the physical network infrastructure. Use the `--no-dhcp` option to have the subnet managed by existing DHCP services.

4. Create a IPv6 subnet on the provider network.

```
$ openstack subnet create --subnet-range fd00:203:0:113::/64 --
↪gateway fd00:203:0:113::1 \
  --ip-version 6 --ipv6-address-mode slaac --network provider1 \
  --dns-nameserver 2001:4860:4860::8844 provider1-v6
+-------------------+------------------------------------------------
↪------+
| Field             | Value                                          ␣
↪      |
+-------------------+------------------------------------------------
↪------+
| allocation_pools  | fd00:203:0:113::2-
↪fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr              | fd00:203:0:113::/64                            ␣
↪      |
| dns_nameservers   | 2001:4860:4860::8844                           ␣
↪      |
| enable_dhcp       | True                                           ␣
↪      |
| gateway_ip        | fd00:203:0:113::1                              ␣
↪      |
| ip_version        | 6                                              ␣
↪      |
| ipv6_address_mode | slaac                                          ␣
↪      |
| ipv6_ra_mode      | None                                           ␣
↪      |
| name              | provider1-v6                                   ␣
↪      |
```

(continues on next page)

(continued from previous page)

```
+--------------------+-------------------------------------------------
↪-----+
```

Note: The Networking service uses the layer-3 agent to provide router advertisement. Provider networks rely on physical network infrastructure for layer-3 services rather than the layer-3 agent. Thus, the physical network infrastructure must provide router advertisement on provider networks for proper operation of IPv6.

Verify network operation

1. On each compute node, verify creation of the `qdhcp` namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
```

2. Source a regular (non-administrative) project credentials.
3. Create the appropriate security group rules to allow `ping` and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| protocol         | icmp      |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-
↪icmp default
+-----------+-----------+
| Field     | Value     |
+-----------+-----------+
| direction | ingress   |
| ethertype | IPv6      |
| protocol  | ipv6-icmp |
+-----------+-----------+

$ openstack security group rule create --proto tcp --dst-port 22␣
↪default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+
```

(continues on next page)

(continued from previous page)

```
$ openstack security group rule create --ethertype IPv6 --proto tcp --
↪dst-port 22 default
+-------------------+------------+
| Field             | Value      |
+-------------------+------------+
| direction         | ingress    |
| ethertype         | IPv6       |
| port_range_max    | 22         |
| port_range_min    | 22         |
| protocol          | tcp        |
+-------------------+------------+
```

4. Launch an instance with an interface on the provider network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros \
  --nic net-id=NETWORK_ID provider-instance1
```

Replace `NETWORK_ID` with the ID of the provider network.

5. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+----------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
| ID                                   | Name                 | Status␣
↪| Networks                                                   |␣
↪Image  | Flavor  |
+--------------------------------------+----------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
| 018e0ae2-b43c-4271-a78d-62653dd03285 | provider-instance1 | ACTIVE␣
↪| provider1=203.0.113.13, fd00:203:0:113:f816:3eff:fe58:be4e |␣
↪cirros | m1.tiny |
+--------------------------------------+----------------------+--------
↪+------------------------------------------------------------+------
↪--+----------+
```

6. On the controller node or any host with access to the provider network, `ping` the IPv4 and IPv6 addresses of the instance.

```
$ ping -c 4 203.0.113.13
PING 203.0.113.13 (203.0.113.13) 56(84) bytes of data.
64 bytes from 203.0.113.13: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.13: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.13: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.13: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.13 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms

$ ping6 -c 4 fd00:203:0:113:f816:3eff:fe58:be4e
PING␣
↪fd00:203:0:113:f816:3eff:fe58:be4e(fd00:203:0:113:f816:3eff:fe58:be4e)␣
↪56 data bytes
```

(continues on next page)

(continued from previous page)

```
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=1 ttl=64␣
↪time=1.25 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=2 ttl=64␣
↪time=0.683 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=3 ttl=64␣
↪time=0.762 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=4 ttl=64␣
↪time=0.486 ms

--- fd00:203:0:113:f816:3eff:fe58:be4e ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.486/0.796/1.253/0.282 ms
```

7. Obtain access to the instance.
8. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network 1 (VLAN)
  - VLAN ID 101 (tagged)
  - IP address ranges 203.0.113.0/24 and fd00:203:0:113::/64
  - Gateway (via physical network infrastructure)
    - IP addresses 203.0.113.1 and fd00:203:0:113:0::1
- Provider network 2 (VLAN)
  - VLAN ID 102 (tagged)
  - IP address range 192.0.2.0/24 and fd00:192:0:2::/64
  - Gateway
    - IP addresses 192.0.2.1 and fd00:192:0:2::1
- Instance 1
  - IP addresses 203.0.113.101 and fd00:203:0:113:0::101
- Instance 2
  - IP addresses 192.0.2.101 and fd00:192:0:2:0::101

North-south

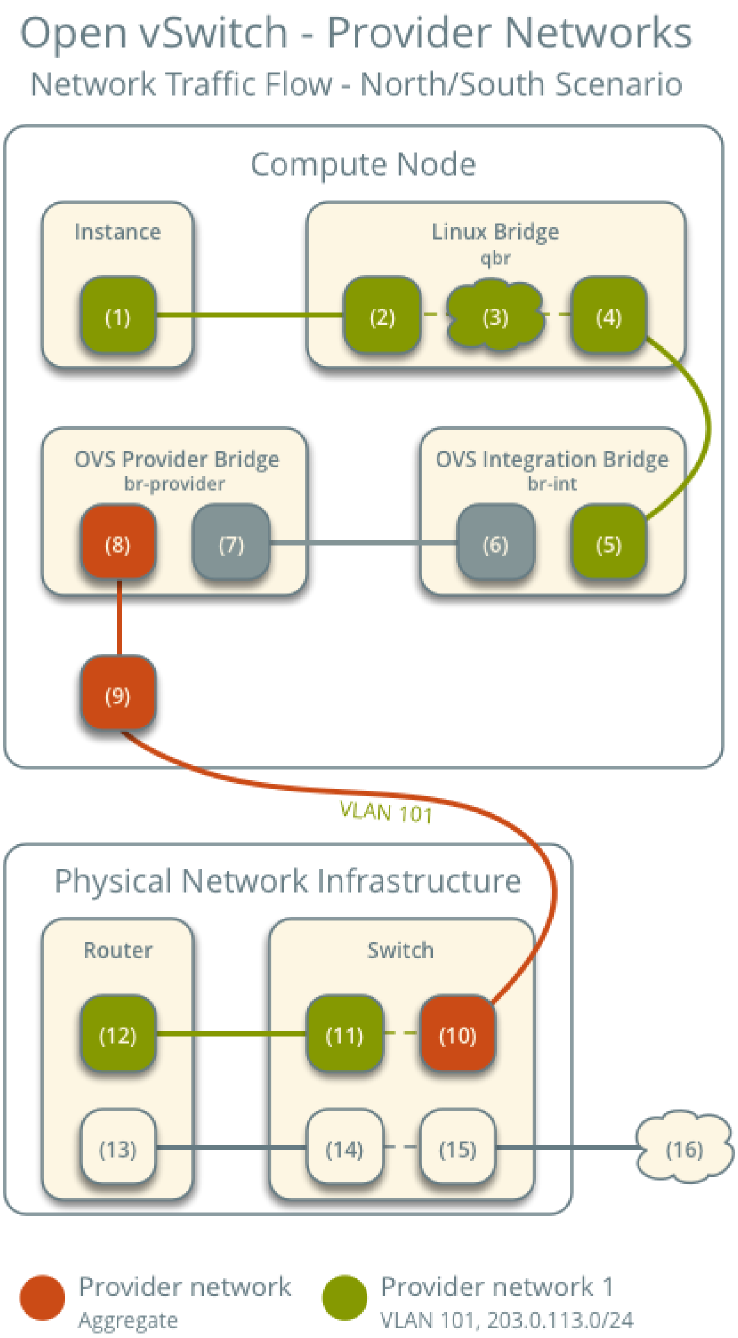
- The instance resides on compute node 1 and uses provider network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1.

1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge `int-br-provider` patch port (6) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (11).
2. The router routes the packet from the provider network (12) to the external network (13) and forwards the packet to the switch (14).
3. The switch forwards the packet to the external network (15).
4. The external network (16) receives the packet.



Note: Return traffic follows similar steps in reverse.

East-west scenario 1: Instances on the same network

Instances on the same network communicate directly between compute nodes containing those instances.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 2 and uses provider network 1.
- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

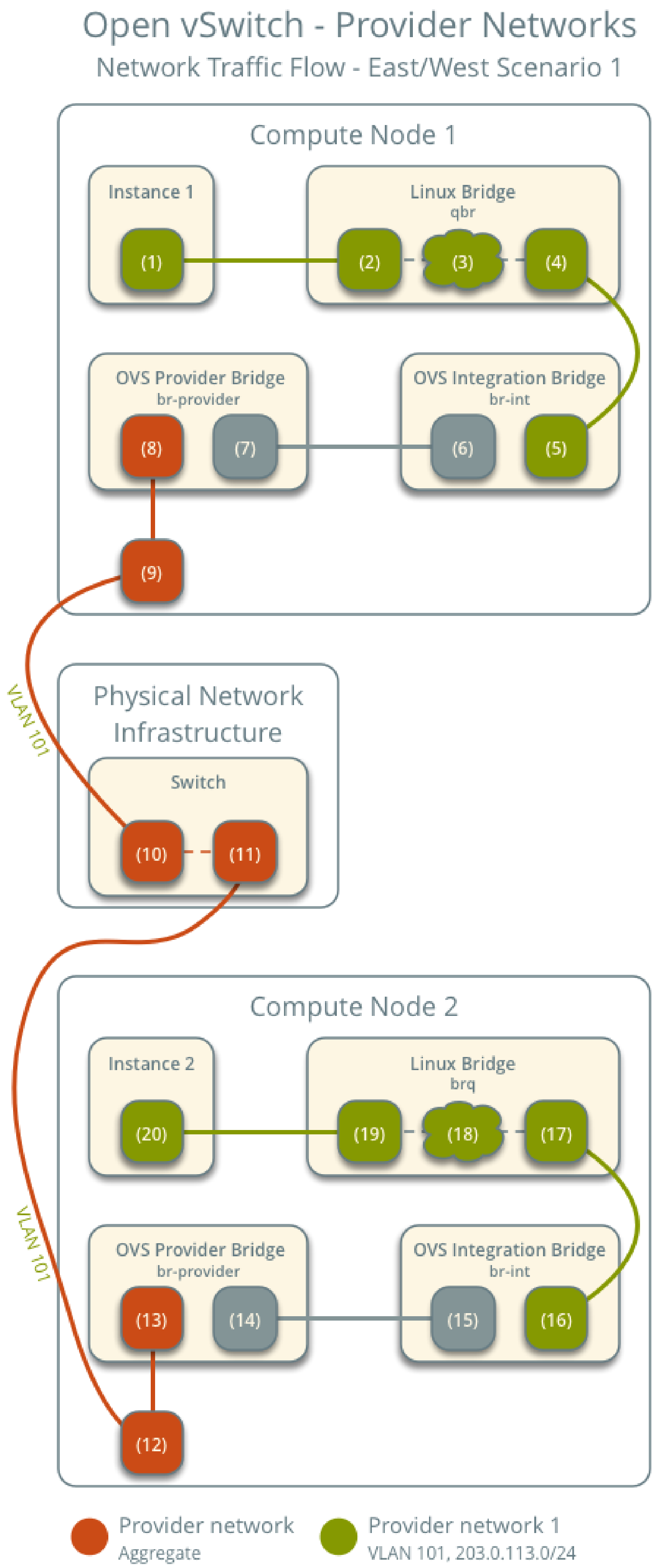
1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge `int-br-provider` patch port (6) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

The following steps involve the physical network infrastructure:

1. The switch forwards the packet from compute node 1 to compute node 2 (11).

The following steps involve compute node 2:

1. The physical network interface (12) forwards the packet to the OVS provider bridge provider network port (13).
2. The OVS provider bridge `phy-br-provider` patch port (14) forwards the packet to the OVS integration bridge `int-br-provider` patch port (15).
3. The OVS integration bridge swaps the actual VLAN tag 101 with the internal VLAN tag.
4. The OVS integration bridge security group port (16) forwards the packet to the security group bridge OVS port (17).
5. Security group rules (18) on the security group bridge handle firewalling and connection tracking for the packet.
6. The security group bridge instance port (19) forwards the packet to the instance 2 interface (20) via `veth` pair.



Open vSwitch - Provider Networks
Network Traffic Flow - East/West Scenario 1
Compute Node 1
Instance 1
(1)
Linux Bridge
qbr
(2)
(3)
(4)
OVS Provider Bridge
br-provider
(8)
(7)
OVS Integration Bridge
br-int
(6)
(5)
(9)
VLAN 101
Physical Network Infrastructure
Switch
(10)
(11)
Compute Node 2
Instance 2
(20)
Linux Bridge
brq
(19)
(18)
(17)
VLAN 101
OVS Provider Bridge
br-provider
(13)
(14)
OVS Integration Bridge
br-int
(15)
(16)
(12)
Provider network
Aggregate
Provider network 1
VLAN 101, 203.0.113.0/24

Note: Return traffic follows similar steps in reverse.

East-west scenario 2: Instances on different networks

Instances communicate via router on the physical network infrastructure.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 1 and uses provider network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VLAN tagging enables multiple logical layer-2 networks to use the same physical layer-2 network.

The following steps involve the compute node:

1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge `int-br-provider` patch port (6) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

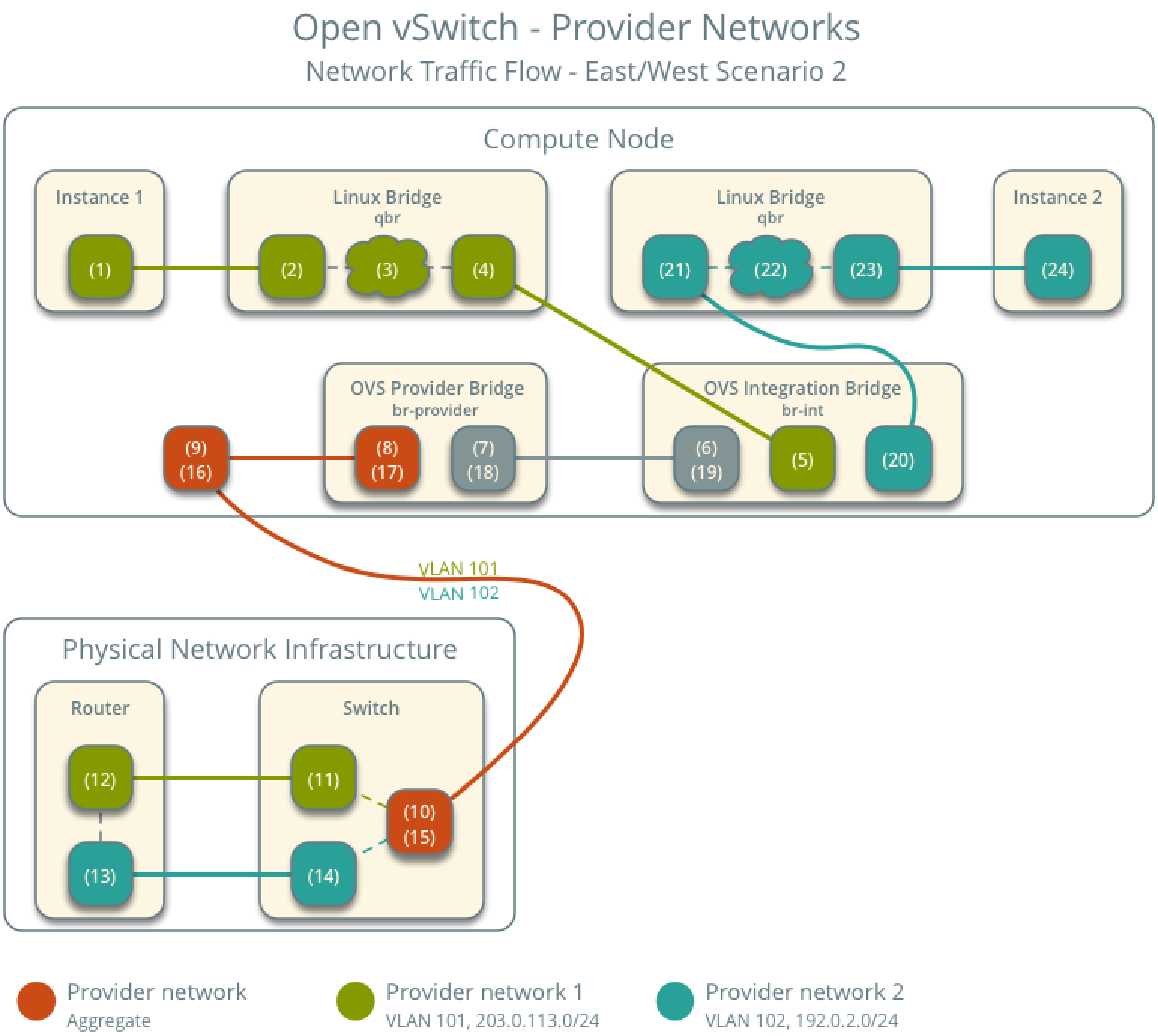
The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (11).
2. The router routes the packet from provider network 1 (12) to provider network 2 (13).
3. The router forwards the packet to the switch (14).
4. The switch adds VLAN tag 102 to the packet and forwards it to compute node 1 (15).

The following steps involve the compute node:

1. The physical network interface (16) forwards the packet to the OVS provider bridge provider network port (17).
2. The OVS provider bridge `phy-br-provider` patch port (18) forwards the packet to the OVS integration bridge `int-br-provider` patch port (19).

3. The OVS integration bridge swaps the actual VLAN tag 102 with the internal VLAN tag.
4. The OVS integration bridge security group port (20) removes the internal VLAN tag and forwards the packet to the security group bridge OVS port (21).
5. Security group rules (22) on the security group bridge handle firewalling and connection tracking for the packet.
6. The security group bridge instance port (23) forwards the packet to the instance 2 interface (24) via `veth` pair.



Note: Return traffic follows similar steps in reverse.

Open vSwitch: Self-service networks

This architecture example augments *Open vSwitch: Provider networks* to support a nearly limitless quantity of entirely virtual networks. Although the Networking service supports VLAN self-service networks, this example focuses on VXLAN self-service networks. For more information on self-service networks, see *Self-service networks*.

Prerequisites

Add one network node with the following components:

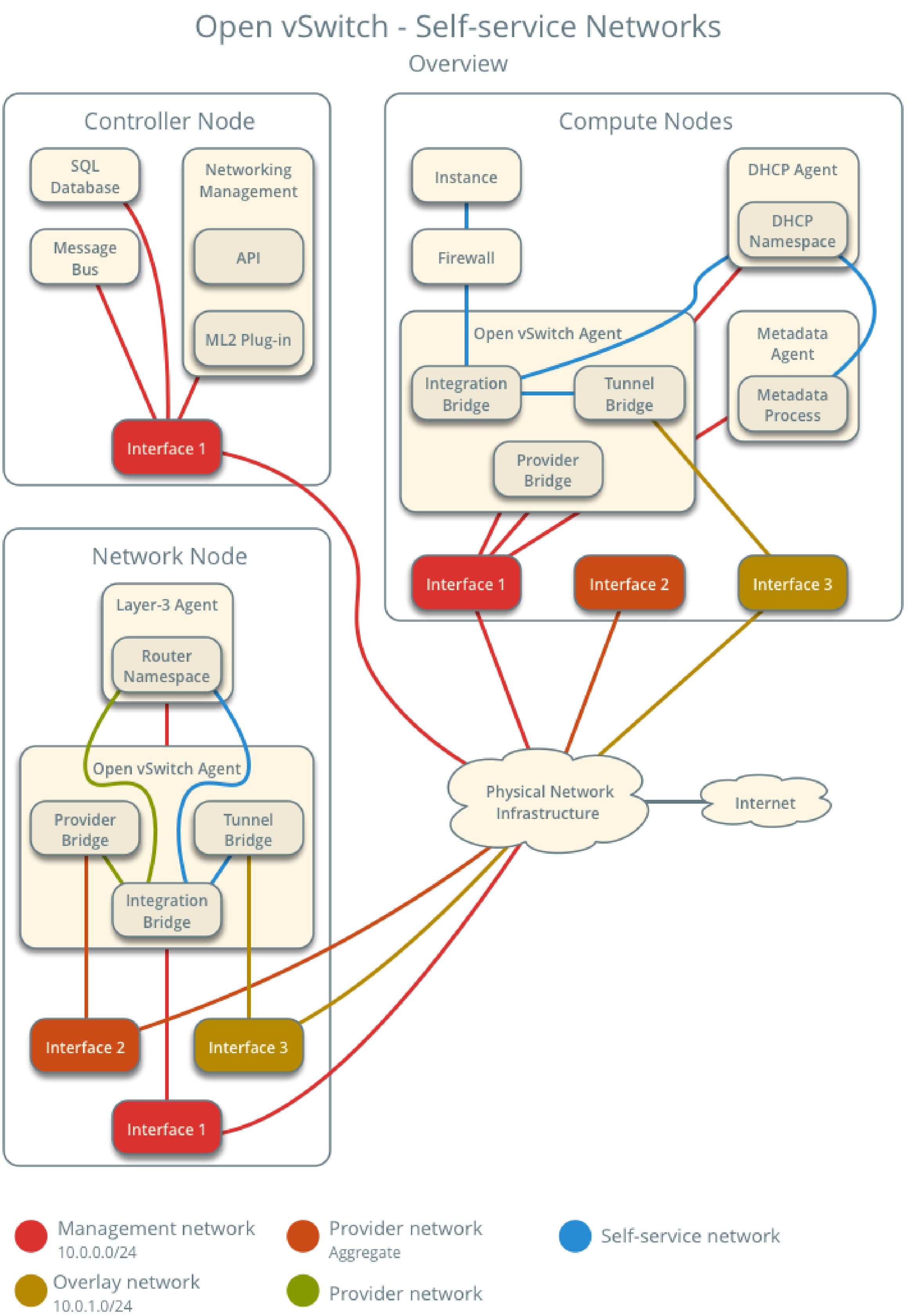
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking Open vSwitch (OVS) layer-2 agent, layer-3 agent, and any including OVS.

Modify the compute nodes with the following components:

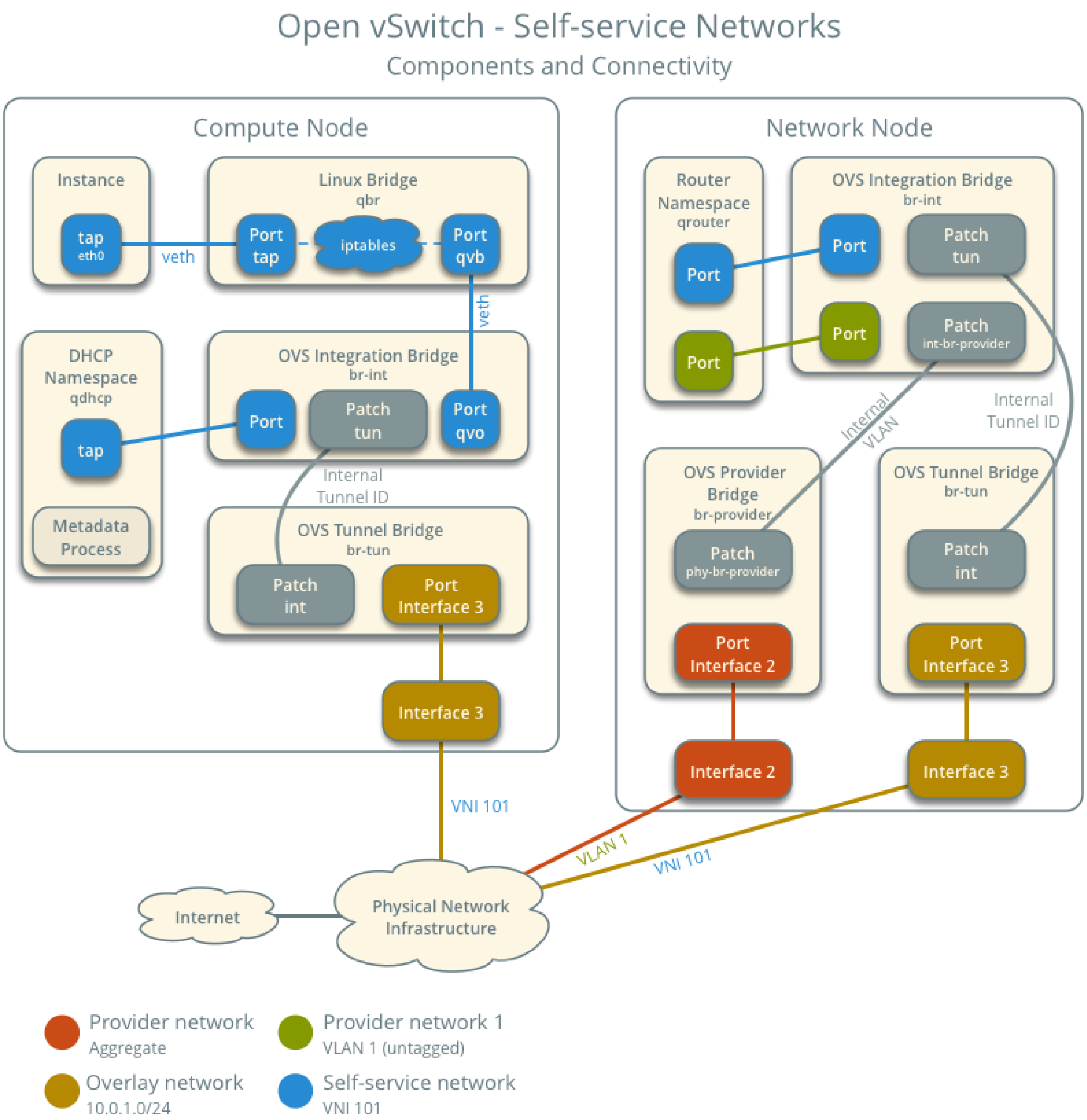
- Add one network interface: overlay.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network node.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) provider network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and with a port on the OVS integration bridge.



Example configuration

Use the following example configuration as a template to add support for self-service networks to an existing operational environment that supports provider networks.

Controller node

1. In the `neutron.conf` file:

   - Enable routing and allow overlapping IP address ranges.

   ```
   [DEFAULT]
   service_plugins = router
   allow_overlapping_ips = True
   ```

2. In the `ml2_conf.ini` file:

   - Add `vxlan` to type drivers and project network types.

   ```
   [ml2]
   type_drivers = flat,vlan,vxlan
   tenant_network_types = vxlan
   ```

   - Enable the layer-2 population mechanism driver.

   ```
   [ml2]
   mechanism_drivers = openvswitch,l2population
   ```

   - Configure the VXLAN network ID (VNI) range.

   ```
   [ml2_type_vxlan]
   vni_ranges = VNI_START:VNI_END
   ```

   Replace `VNI_START` and `VNI_END` with appropriate numerical values.

3. Restart the following services:

   - Neutron Server
   - Open vSwitch agent

Network node

1. Install the Networking service OVS layer-2 agent and layer-3 agent.
2. Install OVS.
3. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...
```

(continues on next page)

(continued from previous page)

```
[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_authtoken]`, `[nova]`, and `[agent]` sections.

4. Start the following services:

   - OVS

5. Create the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-br br-provider
```

6. Add the provider network interface as a port on the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-port br-provider PROVIDER_INTERFACE
```

   Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

7. In the `openvswitch_agent.ini` file, configure the layer-2 agent.

```
[ovs]
bridge_mappings = provider:br-provider
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = True

[securitygroup]
firewall_driver = iptables_hybrid
```

   Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

8. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
```

9. Start the following services:

   - Open vSwitch agent
   - Layer-3 agent

Compute nodes

1. In the `openvswitch_agent.ini` file, enable VXLAN support including layer-2 population.

```
[ovs]
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = True
```

   Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

2. Restart the following services:

   - Open vSwitch agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host    ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
| 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent     |␣
↪compute2 | None              | True  | UP    | neutron-metadata-
↪agent    |
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent |␣
↪compute2 | None              | True  | UP    | neutron-openvswitch-
↪agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent         |␣
↪compute2 | nova              | True  | UP    | neutron-dhcp-agent    ␣
↪      |
| 8805b962-de95-4e40-bdc2-7a0add7521e8 | L3 agent           |␣
↪network1 | nova              | True  | UP    | neutron-l3-agent      ␣
↪      |
| a33cac5a-0266-48f6-9cac-4cef4f8b0358 | Open vSwitch agent |␣
↪network1 | None              | True  | UP    | neutron-openvswitch-
↪agent |
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent     |␣
↪compute1 | None              | True  | UP    | neutron-metadata-
↪agent    |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent         |␣
↪compute1 | nova              | True  | UP    | neutron-dhcp-agent    ␣
↪      |
| bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1 | Open vSwitch agent |␣
↪compute1 | None              | True  | UP    | neutron-openvswitch-
↪agent |
+--------------------------------------+--------------------+---------
↪-+-------------------+-------+-------+---------------------------+
```

(continues on next page)

(continued from previous page)

Create initial networks

The configuration supports multiple VXLAN self-service networks. For simplicity, the following procedure creates one self-service network and a router with a gateway on the flat provider network. The router uses NAT for IPv4 network traffic and directly routes IPv6 network traffic.

Note: IPv6 connectivity with self-service networks often requires addition of static routes to nodes and physical network infrastructure.

1. Source the administrative project credentials.
2. Update the provider network to support external connectivity for self-service networks.

```
$ openstack network set --external provider1
```

Note: This command provides no output.

3. Source a regular (non-administrative) project credentials.
4. Create a self-service network.

```
$ openstack network create selfservice1
+-------------------------+--------------+
| Field                   | Value        |
+-------------------------+--------------+
| admin_state_up          | UP           |
| mtu                     | 1450         |
| name                    | selfservice1 |
| port_security_enabled   | True         |
| router:external         | Internal     |
| shared                  | False        |
| status                  | ACTIVE       |
+-------------------------+--------------+
```

5. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
  --network selfservice1 --dns-nameserver 8.8.4.4 selfservice1-v4
+-------------------+---------------------------+
| Field             | Value                     |
+-------------------+---------------------------+
| allocation_pools  | 192.0.2.2-192.0.2.254     |
| cidr              | 192.0.2.0/24              |
| dns_nameservers   | 8.8.4.4                   |
| enable_dhcp       | True                      |
| gateway_ip        | 192.0.2.1                 |
| ip_version        | 4                         |
| name              | selfservice1-v4           |
+-------------------+---------------------------+
```

6. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-
↪version 6 \
  --ipv6-ra-mode slaac --ipv6-address-mode slaac --network␣
↪selfservice1 \
  --dns-nameserver 2001:4860:4860::8844 selfservice1-v6
+-------------------+-----------------------------------------------
↪------+
| Field             | Value                                         ␣
↪      |
+-------------------+-----------------------------------------------
↪------+
| allocation_pools  | fd00:192:0:2::2-
↪fd00:192:0:2:ffff:ffff:ffff:ffff     |
| cidr              | fd00:192:0:2::/64                             ␣
↪      |
| dns_nameservers   | 2001:4860:4860::8844                          ␣
↪      |
| enable_dhcp       | True                                          ␣
↪      |
| gateway_ip        | fd00:192:0:2::1                               ␣
↪      |
| ip_version        | 6                                             ␣
↪      |
| ipv6_address_mode | slaac                                         ␣
↪      |
| ipv6_ra_mode      | slaac                                         ␣
↪      |
| name              | selfservice1-v6                               ␣
↪      |
+-------------------+-----------------------------------------------
↪------+
```

7. Create a router.

```
$ openstack router create router1
+-----------------------+---------+
| Field                 | Value   |
+-----------------------+---------+
| admin_state_up        | UP      |
| name                  | router1 |
| status                | ACTIVE  |
+-----------------------+---------+
```

8. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router1 selfservice1-v4
$ openstack router add subnet router1 selfservice1-v6
```

Note: These commands provide no output.

9. Add the provider network as the gateway on the router.

```
$ openstack router set --external-gateway provider1 router1
```

Verify network operation

1. On each compute node, verify creation of a second `qdhcp` namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
qdhcp-8fbc13ca-cfe0-4b8a-993b-e33f37ba66d1
```

2. On the network node, verify creation of the `qrouter` namespace.

```
# ip netns
qrouter-17db2a15-e024-46d0-9250-4cd4d336a2cc
```

3. Source a regular (non-administrative) project credentials.

4. Create the appropriate security group rules to allow `ping` and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| protocol         | icmp      |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-
↪icmp default
+-----------+-----------+
| Field     | Value     |
+-----------+-----------+
| direction | ingress   |
| ethertype | IPv6      |
| protocol  | ipv6-icmp |
+-----------+-----------+

$ openstack security group rule create --proto tcp --dst-port 22␣
↪default
+------------------+-----------+
| Field            | Value     |
+------------------+-----------+
| direction        | ingress   |
| ethertype        | IPv4      |
| port_range_max   | 22        |
| port_range_min   | 22        |
| protocol         | tcp       |
| remote_ip_prefix | 0.0.0.0/0 |
+------------------+-----------+

$ openstack security group rule create --ethertype IPv6 --proto tcp --
↪dst-port 22 default
```

(continues on next page)

(continued from previous page)

```
+-------------------+------------+
| Field             | Value      |
+-------------------+------------+
| direction         | ingress    |
| ethertype         | IPv6       |
| port_range_max    | 22         |
| port_range_min    | 22         |
| protocol          | tcp        |
+-------------------+------------+
```

5. Launch an instance with an interface on the self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-
↪id=NETWORK_ID selfservice-instance1
```

Replace `NETWORK_ID` with the ID of the self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+-----------------------+------
↪--+----------------------------------------------------------+------
↪--+----------+
| ID                                   | Name                  |␣
↪Status | Networks                                                 |␣
↪Image  | Flavor  |
+--------------------------------------+-----------------------+------
↪--+----------------------------------------------------------+------
↪--+----------+
| c055cdb0-ebb4-4d65-957c-35cbdbd59306 | selfservice-instance1 |␣
↪ACTIVE | selfservice1=192.0.2.4, fd00:192:0:2:f816:3eff:fe30:9cb0 |␣
↪cirros | m1.tiny |
+--------------------------------------+-----------------------+------
↪--+----------------------------------------------------------+------
↪--+----------+
```

Warning: The IPv4 address resides in a private IP address range (RFC1918). Thus, the Networking service performs source network address translation (SNAT) for the instance to access external networks such as the Internet. Access from external networks such as the Internet to the instance requires a floating IPv4 address. The Networking service performs destination network address translation (DNAT) from the floating IPv4 address to the instance IPv4 address on the self-service network. On the other hand, the Networking service architecture for IPv6 lacks support for NAT due to the significantly larger address space and complexity of NAT. Thus, floating IP addresses do not exist for IPv6 and the Networking service only performs routing for IPv6 subnets on self-service networks. In other words, you cannot rely on NAT to hide instances with IPv4 and IPv6 addresses or only IPv6 addresses and must properly implement security groups to restrict access.

7. On the controller node or any host with access to the provider network, `ping` the IPv6 address of the instance.

```
$ ping6 -c 4 fd00:192:0:2:f816:3eff:fe30:9cb0
PING␣
↪fd00:192:0:2:f816:3eff:fe30:9cb0(fd00:192:0:2:f816:3eff:fe30:9cb0)␣
↪56 data bytes
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=1 ttl=63␣
↪time=2.08 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=2 ttl=63␣
↪time=1.88 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=3 ttl=63␣
↪time=1.55 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=4 ttl=63␣
↪time=1.62 ms

--- fd00:192:0:2:f816:3eff:fe30:9cb0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.557/1.788/2.085/0.217 ms
```

8. Optionally, enable IPv4 access from external networks such as the Internet to the instance.

   1. Create a floating IPv4 address on the provider network.

      ```
      $ openstack floating ip create provider1
      +-------------+--------------------------------------+
      | Field       | Value                                |
      +-------------+--------------------------------------+
      | fixed_ip    | None                                 |
      | id          | 22a1b088-5c9b-43b4-97f3-970ce5df77f2 |
      | instance_id | None                                 |
      | ip          | 203.0.113.16                         |
      | pool        | provider1                            |
      +-------------+--------------------------------------+
      ```

   2. Associate the floating IPv4 address with the instance.

      ```
      $ openstack server add floating ip selfservice-instance1 203.0.
      ↪113.16
      ```

      Note: This command provides no output.

   3. On the controller node or any host with access to the provider network, `ping` the floating IPv4 address of the instance.

      ```
      $ ping -c 4 203.0.113.16
      PING 203.0.113.16 (203.0.113.16) 56(84) bytes of data.
      64 bytes from 203.0.113.16: icmp_seq=1 ttl=63 time=3.41 ms
      64 bytes from 203.0.113.16: icmp_seq=2 ttl=63 time=1.67 ms
      64 bytes from 203.0.113.16: icmp_seq=3 ttl=63 time=1.47 ms
      64 bytes from 203.0.113.16: icmp_seq=4 ttl=63 time=1.59 ms

      --- 203.0.113.16 ping statistics ---
      4 packets transmitted, 4 received, 0% packet loss, time 3005ms
      rtt min/avg/max/mdev = 1.473/2.040/3.414/0.798 ms
      ```

9. Obtain access to the instance.

10. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
  - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
  - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
  - VXLAN ID (VNI) 102
- Self-service router
  - Gateway on the provider network
  - Interface on self-service network 1
  - Interface on self-service network 2
- Instance 1
- Instance 2

North-south scenario 1: Instance with a fixed IP address

For instances with a fixed IPv4 address, the network node performs SNAT on north-south traffic passing from self-service to external networks such as the Internet. For instances with a fixed IPv6 address, the network node performs conventional routing of traffic between self-service and external networks.

- The instance resides on compute node 1 and uses self-service network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1:

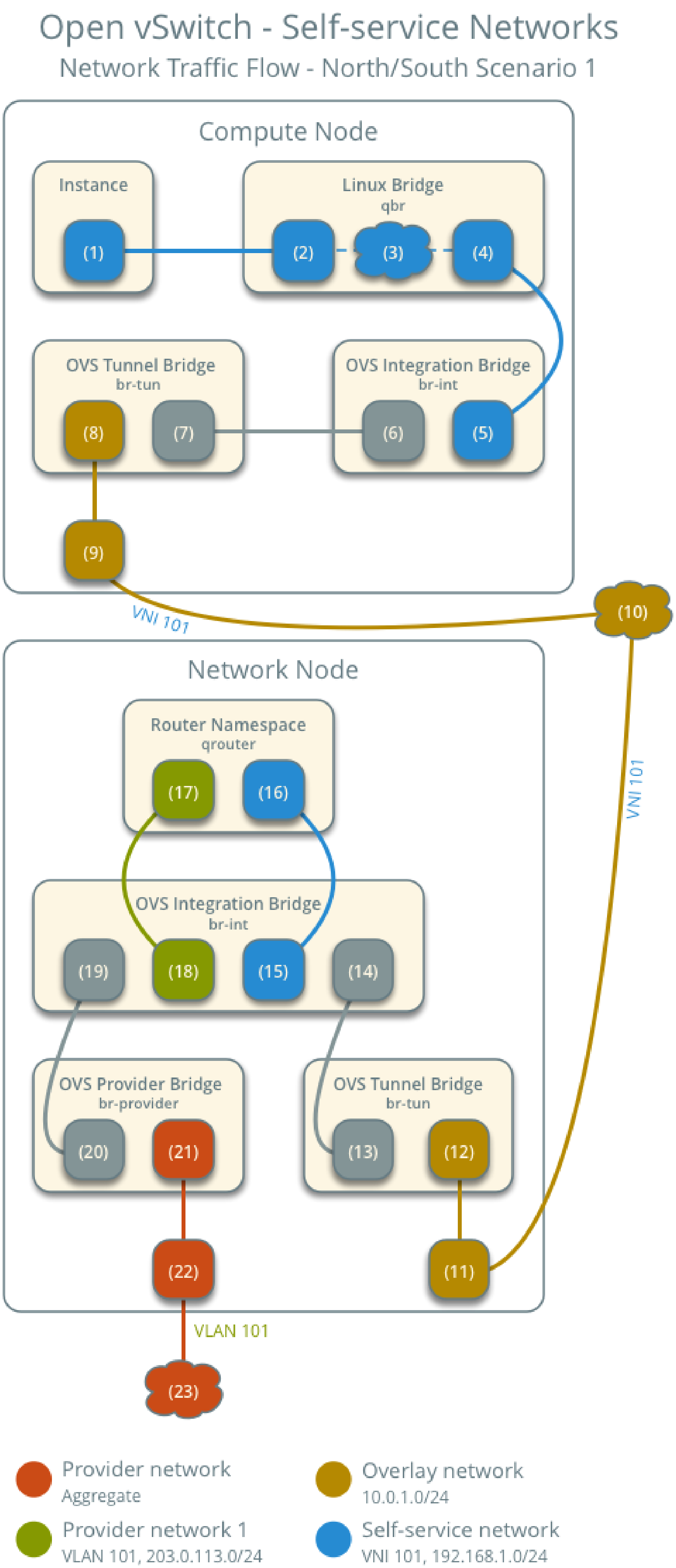
1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.

6. The OVS integration bridge patch port (6) forwards the packet to the OVS tunnel bridge patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.
8. The underlying physical interface (9) for overlay networks forwards the packet to the network node via the overlay network (10).

The following steps involve the network node:

1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch port (13) forwards the packet to the OVS integration bridge patch port (14).
5. The OVS integration bridge port for the self-service network (15) removes the internal VLAN tag and forwards the packet to the self-service network interface (16) in the router namespace.
   - For IPv4, the router performs SNAT on the packet which changes the source IP address to the router IP address on the provider network and sends it to the gateway IP address on the provider network via the gateway interface on the provider network (17).
   - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the provider network, via the provider gateway interface (17).
6. The router forwards the packet to the OVS integration bridge port for the provider network (18).
7. The OVS integration bridge adds the internal VLAN tag to the packet.
8. The OVS integration bridge `int-br-provider` patch port (19) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (20).
9. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
10. The OVS provider bridge provider network port (21) forwards the packet to the physical network interface (22).
11. The physical network interface forwards the packet to the Internet via physical network infrastructure (23).

Note: Return traffic follows similar steps in reverse. However, without a floating IPv4 address, hosts on the provider or external networks cannot originate connections to instances on the self-service network.


Open vSwitch - Self-service Networks
Network Traffic Flow - North/South Scenario 1
Compute Node
Instance
(1)
Linux Bridge
qbr
(2)
(3)
(4)
OVS Tunnel Bridge
br-tun
(8)
(7)
OVS Integration Bridge
br-int
(6)
(5)
(9)
VNI 101
(10)
Network Node
Router Namespace
qrouter
(17)
(16)
OVS Integration Bridge
br-int
(19)
(18)
(15)
(14)
VNI 101
OVS Provider Bridge
br-provider
(20)
(21)
OVS Tunnel Bridge
br-tun
(13)
(12)
(22)
(11)
VLAN 101
(23)
Provider network
Aggregate
Overlay network
10.0.1.0/24
Provider network 1
VLAN 101, 203.0.113.0/24
Self-service network
VNI 101, 192.168.1.0/24

North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address, the network node performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario.

- The instance resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

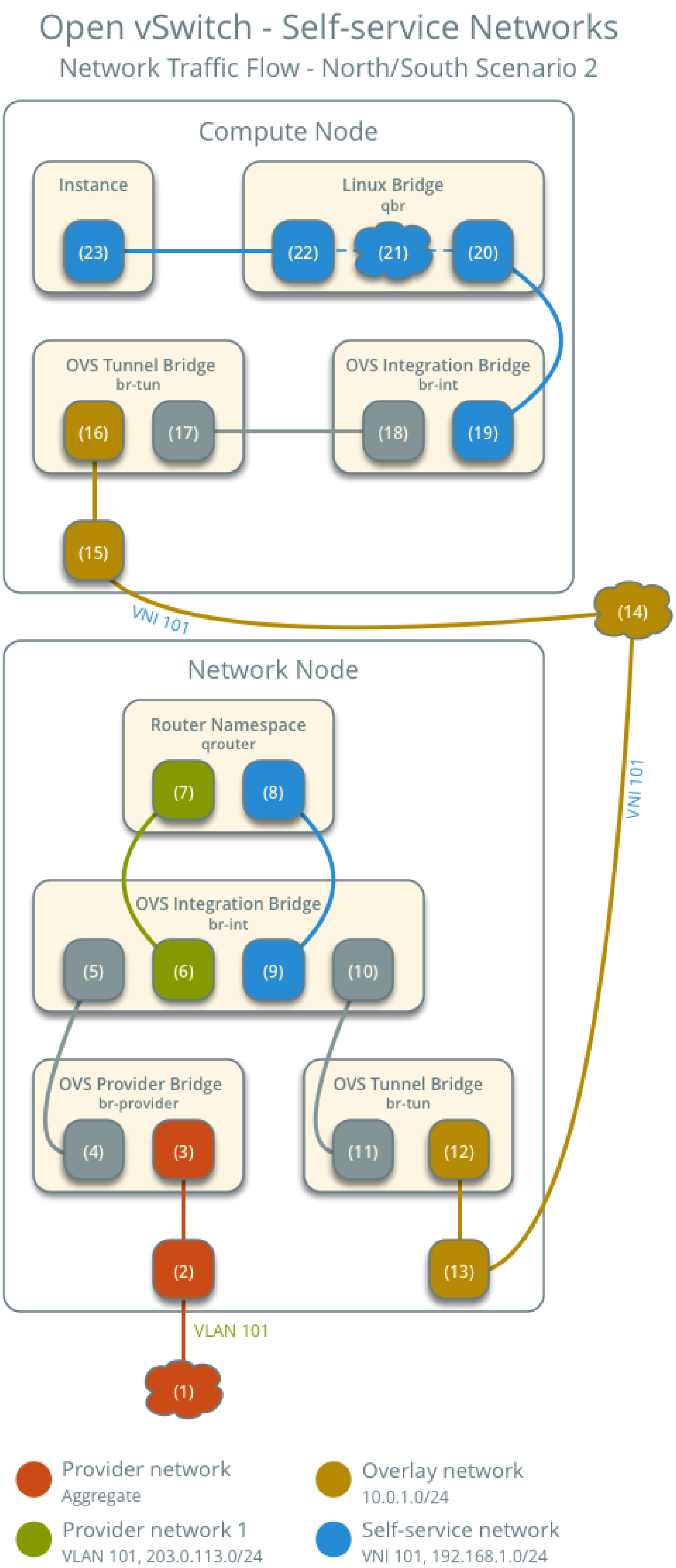
The following steps involve the network node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface (2).
2. The provider physical network interface forwards the packet to the OVS provider bridge provider network port (3).
3. The OVS provider bridge swaps actual VLAN tag 101 with the internal VLAN tag.
4. The OVS provider bridge `phy-br-provider` port (4) forwards the packet to the OVS integration bridge `int-br-provider` port (5).
5. The OVS integration bridge port for the provider network (6) removes the internal VLAN tag and forwards the packet to the provider network interface (6) in the router namespace.
   - For IPv4, the router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network and sends it to the gateway IP address on the self-service network via the self-service interface (7).
   - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the self-service network, via the self-service interface (8).
6. The router forwards the packet to the OVS integration bridge port for the self-service network (9).
7. The OVS integration bridge adds an internal VLAN tag to the packet.
8. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
9. The OVS integration bridge `patch-tun` patch port (10) forwards the packet to the OVS tunnel bridge `patch-int` patch port (11).
10. The OVS tunnel bridge (12) wraps the packet using VNI 101.
11. The underlying physical interface (13) for overlay networks forwards the packet to the network node via the overlay network (14).

The following steps involve the compute node:

1. The underlying physical interface (15) for overlay networks forwards the packet to the OVS tunnel bridge (16).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge `patch-int` patch port (17) forwards the packet to the OVS integration bridge `patch-tun` patch port (18).
5. The OVS integration bridge removes the internal VLAN tag from the packet.

6. The OVS integration bridge security group port (19) forwards the packet to the security group bridge OVS port (20) via `veth` pair.
7. Security group rules (21) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (22) forwards the packet to the instance interface (23) via `veth` pair.


Open vSwitch - Self-service Networks
Network Traffic Flow - North/South Scenario 2
Compute Node
Instance
(23)
Linux Bridge
qbr
(22)
(21)
(20)
OVS Tunnel Bridge
br-tun
(16)
(17)
OVS Integration Bridge
br-int
(18)
(19)
(15)
VNI 101
(14)
Network Node
Router Namespace
qrouter
(7)
(8)
VNI 101
OVS Integration Bridge
br-int
(5)
(6)
(9)
(10)
OVS Provider Bridge
br-provider
(4)
(3)
OVS Tunnel Bridge
br-tun
(11)
(12)
(2)
(13)
VLAN 101
(1)
Provider network
Aggregate
Overlay network
10.0.1.0/24
Provider network 1
VLAN 101, 203.0.113.0/24
Self-service network
VNI 101, 192.168.1.0/24

Note: Egress instance traffic flows similar to north-south scenario 1, except SNAT changes the source IP address of the packet to the floating IPv4 address rather than the router IP address on the provider network.

East-west scenario 1: Instances on the same network

Instances with a fixed IPv4/IPv6 address or floating IPv4 address on the same network communicate directly between compute nodes containing those instances.

By default, the VXLAN protocol lacks knowledge of target location and uses multicast to discover it. After discovery, it stores the location in the local forwarding database. In large deployments, the discovery process can generate a significant amount of network that all nodes must process. To eliminate the latter and generally increase efficiency, the Networking service includes the layer-2 population mechanism driver that automatically populates the forwarding database for VXLAN interfaces. The example configuration enables this driver. For more information, see *ML2 plug-in*.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 2 and uses self-service network 1.
- Instance 1 sends a packet to instance 2.

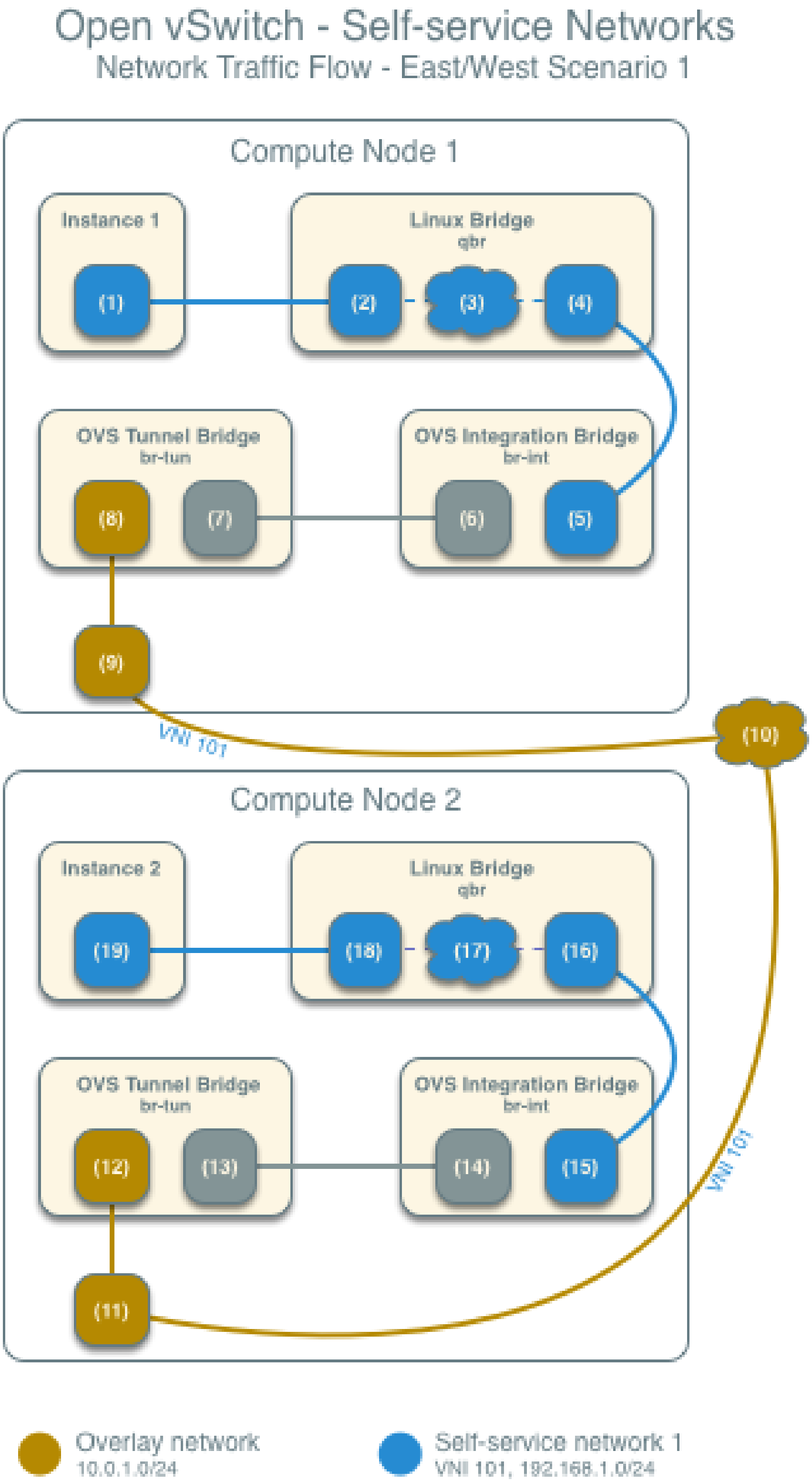
The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
6. The OVS integration bridge patch port (6) forwards the packet to the OVS tunnel bridge patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.
8. The underlying physical interface (9) for overlay networks forwards the packet to compute node 2 via the overlay network (10).

The following steps involve compute node 2:

1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge `patch-int` patch port (13) forwards the packet to the OVS integration bridge `patch-tun` patch port (14).
5. The OVS integration bridge removes the internal VLAN tag from the packet.

6. The OVS integration bridge security group port (15) forwards the packet to the security group bridge OVS port (16) via `veth` pair.
7. Security group rules (17) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (18) forwards the packet to the instance 2 interface (19) via `veth` pair.



Note: Return traffic follows similar steps in reverse.

East-west scenario 2: Instances on different networks

Instances using a fixed IPv4/IPv6 address or floating IPv4 address communicate via router on the network node. The self-service networks must reside on the same router.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 1 and uses self-service network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VXLAN enables multiple overlays to use the same layer-3 network.

The following steps involve the compute node:

1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
6. The OVS integration bridge `patch-tun` patch port (6) forwards the packet to the OVS tunnel bridge `patch-int` patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.
8. The underlying physical interface (9) for overlay networks forwards the packet to the network node via the overlay network (10).

The following steps involve the network node:

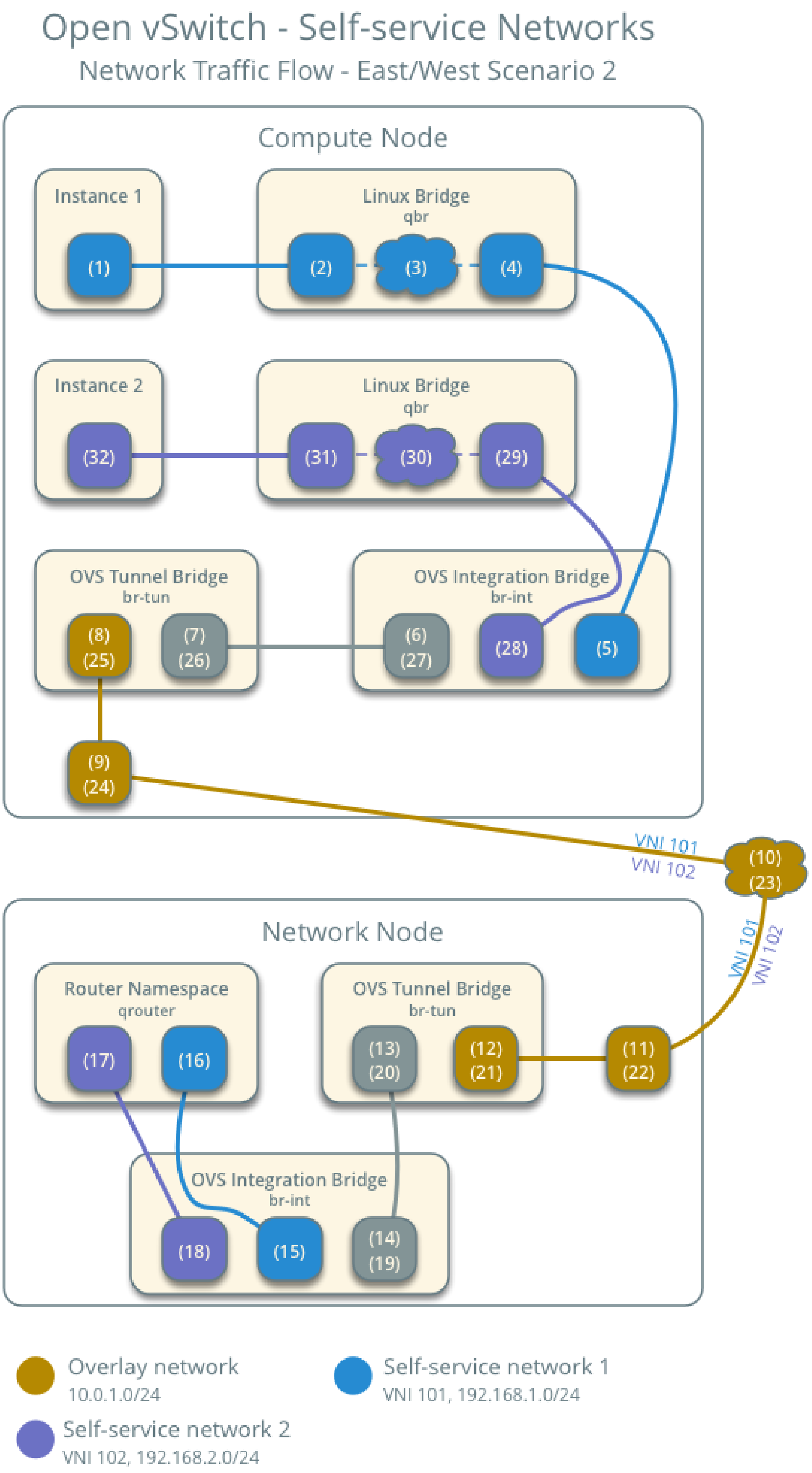
1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge `patch-int` patch port (13) forwards the packet to the OVS integration bridge `patch-tun` patch port (14).
5. The OVS integration bridge port for self-service network 1 (15) removes the internal VLAN tag and forwards the packet to the self-service network 1 interface (16) in the router namespace.
6. The router sends the packet to the next-hop IP address, typically the gateway IP address on self-service network 2, via the self-service network 2 interface (17).
7. The router forwards the packet to the OVS integration bridge port for self-service network 2 (18).
8. The OVS integration bridge adds the internal VLAN tag to the packet.
9. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.

10. The OVS integration bridge `patch-tun` patch port (19) forwards the packet to the OVS tunnel bridge `patch-int` patch port (20).
11. The OVS tunnel bridge (21) wraps the packet using VNI 102.
12. The underlying physical interface (22) for overlay networks forwards the packet to the compute node via the overlay network (23).

The following steps involve the compute node:

1. The underlying physical interface (24) for overlay networks forwards the packet to the OVS tunnel bridge (25).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge `patch-int` patch port (26) forwards the packet to the OVS integration bridge `patch-tun` patch port (27).
5. The OVS integration bridge removes the internal VLAN tag from the packet.
6. The OVS integration bridge security group port (28) forwards the packet to the security group bridge OVS port (29) via `veth` pair.
7. Security group rules (30) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (31) forwards the packet to the instance interface (32) via `veth` pair.

Note: Return traffic follows similar steps in reverse.


Open vSwitch - Self-service Networks
Network Traffic Flow - East/West Scenario 2
Compute Node
Instance 1
(1)
Linux Bridge
qbr
(2)
(3)
(4)
Instance 2
(32)
Linux Bridge
qbr
(31)
(30)
(29)
OVS Tunnel Bridge
br-tun
(8)
(25)
(7)
(26)
OVS Integration Bridge
br-int
(6)
(27)
(28)
(5)
(9)
(24)
VNI 101
VNI 102
(10)
(23)
VNI 101
VNI 102
Network Node
Router Namespace
qrouter
(17)
(16)
OVS Tunnel Bridge
br-tun
(13)
(20)
(12)
(21)
(11)
(22)
OVS Integration Bridge
br-int
(18)
(15)
(14)
(19)
Overlay network
10.0.1.0/24
Self-service network 1
VNI 101, 192.168.1.0/24
Self-service network 2
VNI 102, 192.168.2.0/24

Open vSwitch: High availability using VRRP

This architecture example augments the self-service deployment example with a high-availability mechanism using the Virtual Router Redundancy Protocol (VRRP) via `keepalived` and provides failover of routing for self-service networks. It requires a minimum of two network nodes because VRRP creates one master (active) instance and at least one backup instance of each router.

During normal operation, `keepalived` on the master router periodically transmits *heartbeat* packets over a hidden network that connects all VRRP routers for a particular project. Each project with VRRP routers uses a separate hidden network. By default this network uses the first value in the `tenant_network_types` option in the `ml2_conf.ini` file. For additional control, you can specify the self-service network type and physical network name for the hidden network using the `l3_ha_network_type` and `l3_ha_network_name` options in the `neutron.conf` file.

If `keepalived` on the backup router stops receiving *heartbeat* packets, it assumes failure of the master router and promotes the backup router to master router by configuring IP addresses on the interfaces in the `qrouter` namespace. In environments with more than one backup router, `keepalived` on the backup router with the next highest priority promotes that backup router to master router.

Note: This high-availability mechanism configures VRRP using the same priority for all routers. Therefore, VRRP promotes the backup router with the highest IP address to the master router.

Warning: There is a known bug with `keepalived` v1.2.15 and earlier which can cause packet loss when `max_l3_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to `keepalived` v1.2.16 or greater when using this feature.

Interruption of VRRP *heartbeat* traffic between network nodes, typically due to a network interface or physical network infrastructure failure, triggers a failover. Restarting the layer-3 agent, or failure of it, does not trigger a failover providing `keepalived` continues to operate.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Instance network traffic on self-service networks using a particular router only traverses the master instance of that router. Thus, resource limitations of a particular network node can impact all master instances of routers on that network node without triggering failover to another network node. However, you can configure the scheduler to distribute the master instance of each router uniformly across a pool of network nodes to reduce the chance of resource contention on any particular network node.
- Only supports self-service networks using a router. Provider networks operate at layer-2 and rely on physical network infrastructure for redundancy.
- For instances with a floating IPv4 address, maintains state of network connections during failover as a side effect of 1:1 static NAT. The mechanism does not actually implement connection tracking.

For production deployments, we recommend at least three network nodes with sufficient resources to handle network traffic for the entire environment if one network node fails. Also, the remaining two nodes can continue to provide redundancy.

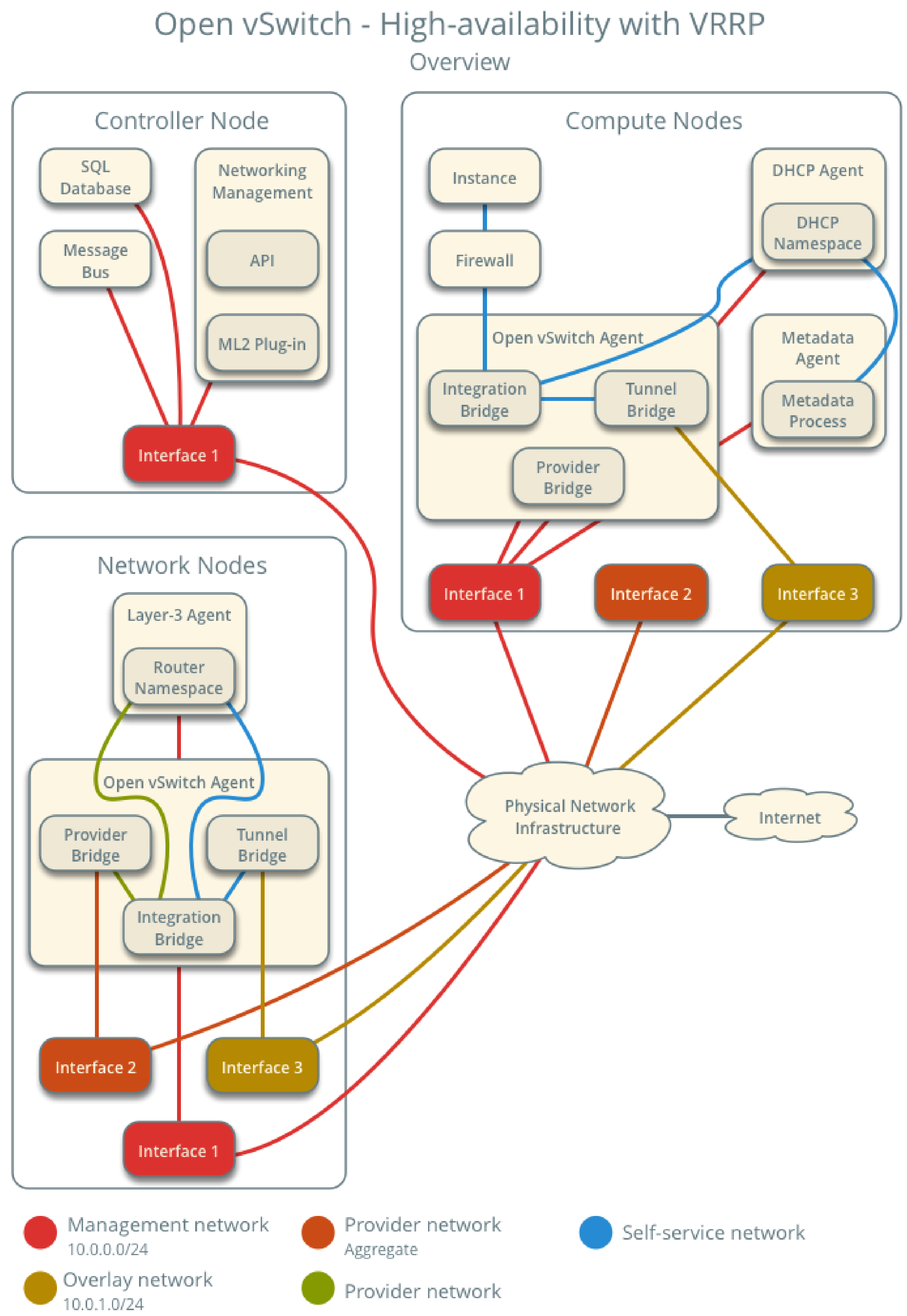
Prerequisites

Add one network node with the following components:

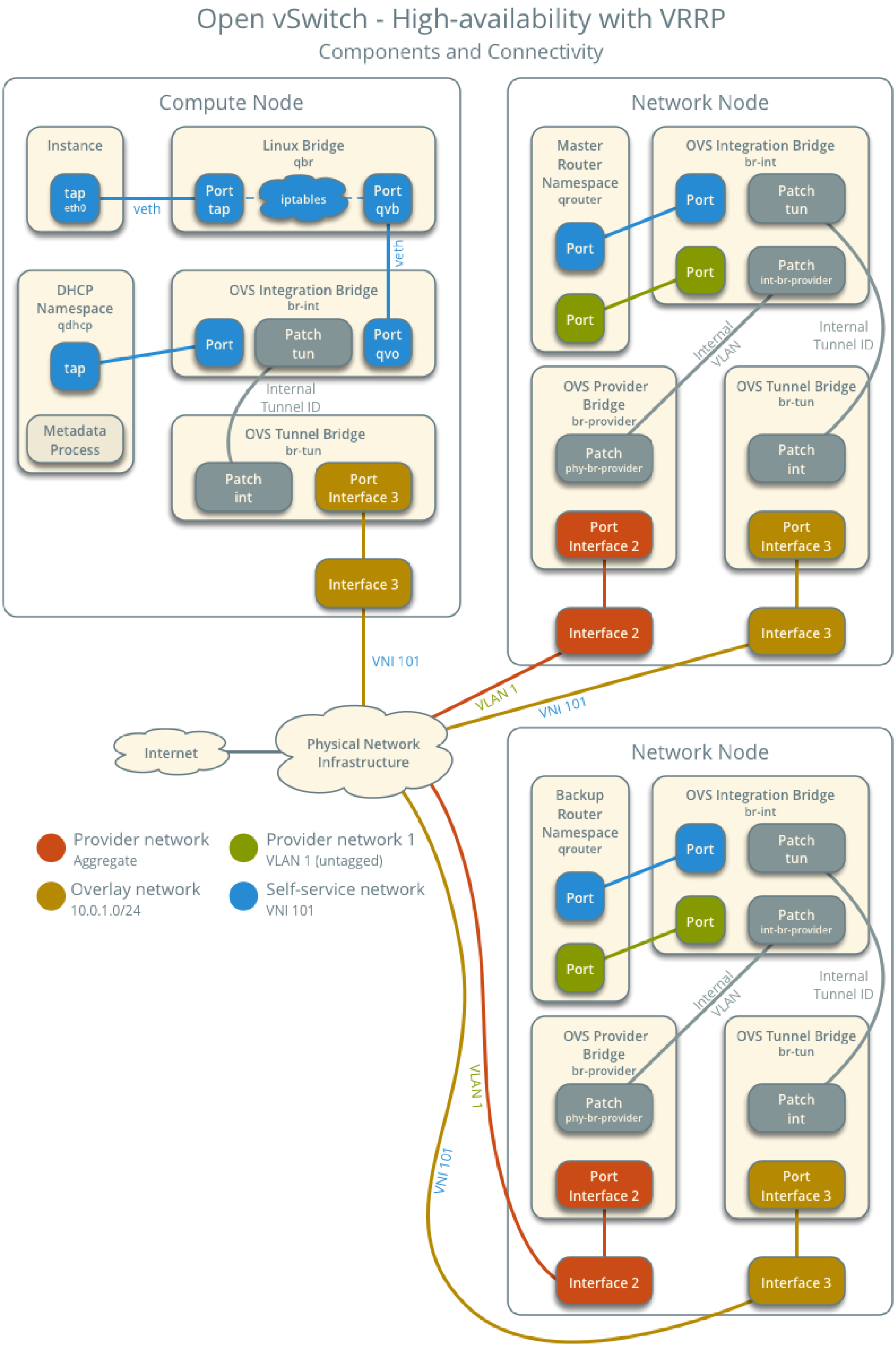
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking layer-2 agent, layer-3 agent, and any dependencies.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network nodes.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) network. The primary router resides on network node 1. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the overlay physical network interface.


Open vSwitch - High-availability with VRRP
Components and Connectivity
Compute Node
Instance
tap
eth0
veth
Linux Bridge
qbr
Port
tap
iptables
Port
qvb
veth
DHCP
Namespace
qdhcp
tap
Metadata
Process
OVS Integration Bridge
br-int
Port
Patch
tun
Port
qvo
Internal
Tunnel ID
OVS Tunnel Bridge
br-tun
Patch
int
Port
Interface 3
Interface 3
VNI 101
Network Node
Master
Router
Namespace
qrouter
Port
Port
OVS Integration Bridge
br-int
Port
Patch
tun
Port
Patch
int-br-provider
Internal
VLAN
Internal
Tunnel ID
OVS Provider
Bridge
br-provider
Patch
phy-br-provider
Port
Interface 2
OVS Tunnel Bridge
br-tun
Patch
int
Port
Interface 3
Interface 2
Interface 3
VLAN 1
VNI 101
Internet
Physical Network
Infrastructure
Provider network
Aggregate
Provider network 1
VLAN 1 (untagged)
Overlay network
10.0.1.0/24
Self-service network
VNI 101
Network Node
Backup
Router
Namespace
qrouter
Port
Port
OVS Integration Bridge
br-int
Port
Patch
tun
Port
Patch
int-br-provider
Internal
VLAN
Internal
Tunnel ID
OVS Provider
Bridge
br-provider
Patch
phy-br-provider
Port
Interface 2
OVS Tunnel Bridge
br-tun
Patch
int
Port
Interface 3
VLAN 1
VNI 101
Interface 2
Interface 3

Example configuration

Use the following example configuration as a template to add support for high-availability using VRRP to an existing operational environment that supports self-service networks.

Controller node

1. In the neutron.conf file:

   - Enable VRRP.

```
[DEFAULT]
l3_ha = True
```

2. Restart the following services:

   - Server

Network node 1

No changes.

Network node 2

1. Install the Networking service OVS layer-2 agent and layer-3 agent.

2. Install OVS.

3. In the neutron.conf file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authtoken]
# ...

[nova]
# ...

[agent]
# ...
```

   See the Installation Tutorials and Guides and Configuration Reference for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_authtoken], [nova], and [agent] sections.

4. Start the following services:

   - OVS

5. Create the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-br br-provider
```

6. Add the provider network interface as a port on the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-port br-provider PROVIDER_INTERFACE
```

   Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

7. In the `openvswitch_agent.ini` file, configure the layer-2 agent.

```
[ovs]
bridge_mappings = provider:br-provider
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = true

[securitygroup]
firewall_driver = iptables_hybrid
```

   Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

8. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
```

9. Start the following services:

   - Open vSwitch agent
   - Layer-3 agent

Compute nodes

No changes.

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host     ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
```

(continues on next page)

(continued from previous page)

```
| 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent     |␣
↪compute2 | None                  | True  | UP    | neutron-metadata-
↪agent    |
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent |␣
↪compute2 | None                  | True  | UP    | neutron-openvswitch-
↪agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent         |␣
↪compute2 | nova                  | True  | UP    | neutron-dhcp-agent  ␣
↪       |
| 8805b962-de95-4e40-bdc2-7a0add7521e8 | L3 agent           |␣
↪network1 | nova                  | True  | UP    | neutron-l3-agent    ␣
↪       |
| a33cac5a-0266-48f6-9cac-4cef4f8b0358 | Open vSwitch agent |␣
↪network1 | None                  | True  | UP    | neutron-openvswitch-
↪agent |
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent     |␣
↪compute1 | None                  | True  | UP    | neutron-metadata-
↪agent    |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent         |␣
↪compute1 | nova                  | True  | UP    | neutron-dhcp-agent  ␣
↪       |
| bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1 | Open vSwitch agent |␣
↪compute1 | None                  | True  | UP    | neutron-openvswitch-
↪agent |
| 7f00d759-f2c9-494a-9fbf-fd9118104d03 | Open vSwitch agent |␣
↪network2 | None                  | True  | UP    | neutron-openvswitch-
↪agent |
| b28d8818-9e32-4888-930b-29addbdd2ef9 | L3 agent           |␣
↪network2 | nova                  | True  | UP    | neutron-l3-agent    ␣
↪       |
+--------------------------------------+--------------------+---------
↪-+-----------------------+-------+-------+---------------------------+
```

Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use VRRP. The following procedure creates an additional self-service network and router. The Networking service also supports adding high-availability to existing routers. However, the procedure requires administratively disabling and enabling each router which temporarily interrupts network connectivity for self-service networks with interfaces on that router.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

```
$ openstack network create selfservice2
+-------------------------+--------------+
| Field                   | Value        |
+-------------------------+--------------+
| admin_state_up          | UP           |
| mtu                     | 1450         |
| name                    | selfservice2 |
| port_security_enabled   | True         |
```

(continues on next page)

(continued from previous page)

```
| router:external           | Internal      |
| shared                    | False         |
| status                    | ACTIVE        |
+---------------------------+---------------+
```

3. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 198.51.100.0/24 \
  --network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4
+-------------------+------------------------------+
| Field             | Value                        |
+-------------------+------------------------------+
| allocation_pools  | 198.51.100.2-198.51.100.254  |
| cidr              | 198.51.100.0/24              |
| dns_nameservers   | 8.8.4.4                      |
| enable_dhcp       | True                         |
| gateway_ip        | 198.51.100.1                 |
| ip_version        | 4                            |
| name              | selfservice2-v4              |
+-------------------+------------------------------+
```

4. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:198:51:100::/64 --ip-
↪version 6 \
  --ipv6-ra-mode slaac --ipv6-address-mode slaac --network␣
↪selfservice2 \
  --dns-nameserver 2001:4860:4860::8844 selfservice2-v6
+-------------------+----------------------------------------------------
↪--------+
| Field             | Value                                              ␣
↪        |
+-------------------+----------------------------------------------------
↪--------+
| allocation_pools  | fd00:198:51:100::2-
↪fd00:198:51:100:ffff:ffff:ffff:ffff |
| cidr              | fd00:198:51:100::/64                               ␣
↪        |
| dns_nameservers   | 2001:4860:4860::8844                               ␣
↪        |
| enable_dhcp       | True                                               ␣
↪        |
| gateway_ip        | fd00:198:51:100::1                                 ␣
↪        |
| ip_version        | 6                                                  ␣
↪        |
| ipv6_address_mode | slaac                                              ␣
↪        |
| ipv6_ra_mode      | slaac                                              ␣
↪        |
| name              | selfservice2-v6                                    ␣
↪        |
+-------------------+----------------------------------------------------
↪--------+
```

5. Create a router.

```
$ openstack router create router2
+-----------------------+---------+
| Field                 | Value   |
+-----------------------+---------+
| admin_state_up        | UP      |
| name                  | router2 |
| status                | ACTIVE  |
+-----------------------+---------+
```

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4
$ openstack router add subnet router2 selfservice2-v6
```

Note: These commands provide no output.

7. Add the provider network as a gateway on the router.

```
$ openstack router set --external-gateway provider1 router2
```

Verify network operation

1. Source the administrative project credentials.
2. Verify creation of the internal high-availability network that handles VRRP *heartbeat* traffic.

```
$ openstack network list
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
| ID                                   | Name                       ␣
↪                      | Subnets                              |
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
| 1b8519c1-59c4-415c-9da2-a67d53c68455 | HA network tenant␣
↪f986edf55ae945e2bef3cb4bfd589928 | 6843314a-1e76-4cc9-94f5-
↪c64b7a39364a |
+--------------------------------------+----------------------------
↪----------------------+--------------------------------------+
```

3. On each network node, verify creation of a `qrouter` namespace with the same ID.

   Network node 1:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

   Network node 2:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Note: The namespace for router 1 from *Linux bridge: Self-service networks* should only appear on network node 1 because of creation prior to enabling VRRP.

4. On each network node, show the IP address of interfaces in the `qrouter` namespace. With the exception of the VRRP interface, only one namespace belonging to the master router instance contains IP addresses on the interfaces.

   Network node 1:

```
# ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr␣
↪show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN␣
↪group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ha-eb820380-40@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:78:ba:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 169.254.192.1/18 brd 169.254.255.255 scope global ha-
↪eb820380-40
       valid_lft forever preferred_lft forever
    inet 169.254.0.1/24 scope global ha-eb820380-40
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe78:ba99/64 scope link
       valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 198.51.100.1/24 scope global qr-da3504ad-ba
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fedc:8ea8/64 scope link
       valid_lft forever preferred_lft forever
4: qr-442e36eb-fc@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:ee:c8:41 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fd00:198:51:100::1/64 scope global nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:feee:c841/64 scope link
       valid_lft forever preferred_lft forever
5: qg-33fedbc5-43@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 203.0.113.21/24 scope global qg-33fedbc5-43
       valid_lft forever preferred_lft forever
    inet6 fd00:203:0:113::21/64 scope global nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe03:1af6/64 scope link
       valid_lft forever preferred_lft forever
```

   Network node 2:

```
# ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr␣
↪show
```

(continues on next page)

(continued from previous page)

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN␣
↪group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ha-7a7ce184-36@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:16:59:84 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 169.254.192.2/18 brd 169.254.255.255 scope global ha-
↪7a7ce184-36
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe16:5984/64 scope link
       valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
4: qr-442e36eb-fc@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450␣
↪qdisc noqueue state UP group default qlen 1000
5: qg-33fedbc5-43@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Note: The master router may reside on network node 2.

5. Launch an instance with an interface on the additional self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-
↪id=NETWORK_ID selfservice-instance2
```

Replace `NETWORK_ID` with the ID of the additional self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+----------------------+-------
↪--+-------------------------------------------------------------
↪+--------+---------+
| ID                                   | Name                 |␣
↪Status | Networks                                                    ␣
↪      | Image  | Flavor  |
+--------------------------------------+----------------------+-------
↪--+-------------------------------------------------------------
↪+--------+---------+
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 |␣
↪ACTIVE | selfservice2=fd00:198:51:100:f816:3eff:fe71:e93e, 198.51.
↪100.4 | cirros | m1.tiny |
+--------------------------------------+----------------------+-------
↪--+-------------------------------------------------------------
↪+--------+---------+
```

7. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| fixed_ip    | None                                 |
| id          | 0174056a-fa56-4403-b1ea-b5151a31191f |
| instance_id | None                                 |
| ip          | 203.0.113.17                         |
| pool        | provider1                            |
+-------------+--------------------------------------+
```

8. Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

Note: This command provides no output.

Verify failover operation

1. Begin a continuous `ping` of both the floating IPv4 address and IPv6 address of the instance. While performing the next three steps, you should see a minimal, if any, interruption of connectivity to the instance.
2. On the network node with the master router, administratively disable the overlay network interface.
3. On the other network node, verify promotion of the backup router to master router by noting addition of IP addresses to the interfaces in the `qrouter` namespace.
4. On the original network node in step 2, administratively enable the overlay network interface. Note that the master router remains on the network node in step 3.

Keepalived VRRP health check

The health of your `keepalived` instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the `l3_agent.ini` file:

```
ha_vrrp_health_check_interval = 30
```

Where `ha_vrrp_health_check_interval` indicates how often in seconds the health check should run. The default value is `0`, which indicates that the check should not run at all.

Network traffic flow

This high-availability mechanism simply augments *Open vSwitch: Self-service networks* with failover of layer-3 services to another router if the primary router fails. Thus, you can reference *Self-service network traffic flow* for normal operation.

Open vSwitch: High availability using DVR

This architecture example augments the self-service deployment example with the Distributed Virtual Router (DVR) high-availability mechanism that provides connectivity between self-service and provider networks on compute nodes rather than network nodes for specific scenarios. For instances with a floating IPv4 address, routing between self-service and provider networks resides completely on the compute nodes to eliminate single point of failure and performance issues with network nodes. Routing also resides completely on the compute nodes for instances with a fixed or floating IPv4 address using self-service networks on the same distributed virtual router. However, instances with a fixed IP address still rely on the network node for routing and SNAT services between self-service and provider networks.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Only provides connectivity to an instance via the compute node on which the instance resides if the instance resides on a self-service network with a floating IPv4 address. Instances on self-service networks with only an IPv6 address or both IPv4 and IPv6 addresses rely on the network node for IPv6 connectivity.
- The instance of a router on each compute node consumes an IPv4 address on the provider network on which it contains a gateway.

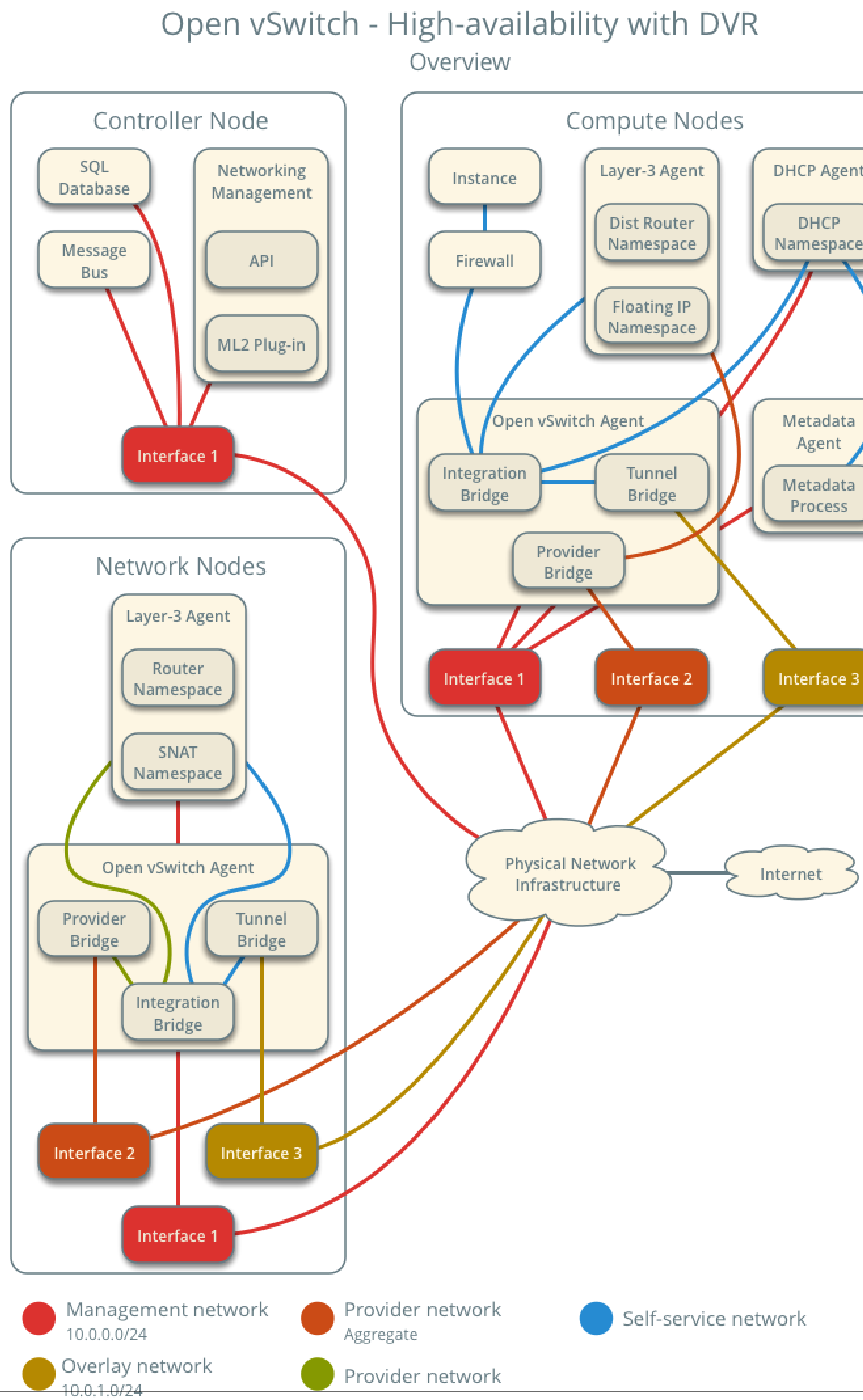
Prerequisites

Modify the compute nodes with the following components:

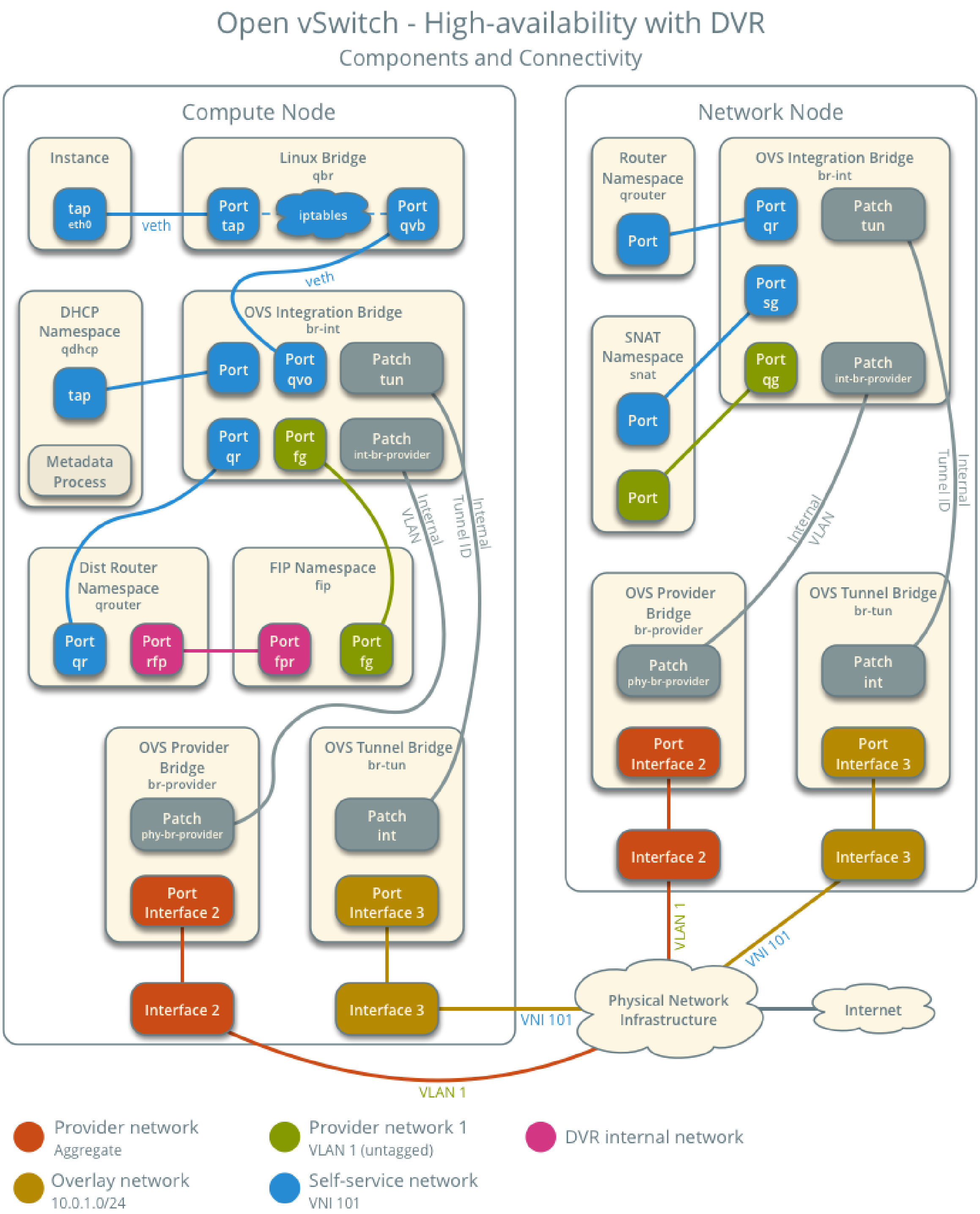
- Install the OpenStack Networking layer-3 agent.

Note: Consider adding at least one additional network node to provide high-availability for instances with a fixed IP address. See See *Distributed Virtual Routing with VRRP* for more information.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace with a port on the OVS integration bridge.



Example configuration

Use the following example configuration as a template to add support for high-availability using DVR to an existing operational environment that supports self-service networks.

Controller node

1. In the `neutron.conf` file:

   - Enable distributed routing by default for all routers.

```
[DEFAULT]
router_distributed = True
```

Note: For a large scale cloud, if your deployment is running DVR with DHCP, we recommend you set `host_dvr_for_dhcp=False` to achieve higher L3 agent router processing performance. When this is set to False, DNS functionality will not be available via the DHCP namespace (dnsmasq) however, a different nameserver will have to be configured, for example, by specifying a value in `dns_nameservers` for subnets.

1. Restart the following services:

   - Server

Network node

1. In the `openvswitch_agent.ini` file, enable distributed routing.

```
[agent]
enable_distributed_routing = True
```

2. In the `l3_agent.ini` file, configure the layer-3 agent to provide SNAT services.

```
[DEFAULT]
agent_mode = dvr_snat
```

3. Restart the following services:

   - Open vSwitch agent
   - Layer-3 agent

Compute nodes

1. Install the Networking service layer-3 agent.
2. In the `openswitch_agent.ini` file, enable distributed routing.

```
[agent]
enable_distributed_routing = True
```

3. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
agent_mode = dvr
```

4. Restart the following services:
   - Open vSwitch agent
   - Layer-3 agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host     ␣
↪ | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+----------
↪-+-------------------+-------+-------+---------------------------+
| 05d980f2-a4fc-4815-91e7-a7f7e118c0db | L3 agent           |␣
↪compute1 | nova              | True  | UP    | neutron-l3-agent         ␣
↪        |
| 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent     |␣
↪compute2 | None              | True  | UP    | neutron-metadata-
↪agent     |
| 2a2e9a90-51b8-4163-a7d6-3e199ba2374b | L3 agent           |␣
↪compute2 | nova              | True  | UP    | neutron-l3-agent         ␣
↪        |
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent |␣
↪compute2 | None              | True  | UP    | neutron-openvswitch-
↪agent |
| 513caa68-0391-4e53-a530-082e2c23e819 | Linux bridge agent |␣
↪compute1 | None              | True  | UP    | neutron-linuxbridge-
↪agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent         |␣
↪compute2 | nova              | True  | UP    | neutron-dhcp-agent       ␣
↪        |
| 8805b962-de95-4e40-bdc2-7a0add7521e8 | L3 agent           |␣
↪network1 | nova              | True  | UP    | neutron-l3-agent         ␣
↪        |
| a33cac5a-0266-48f6-9cac-4cef4f8b0358 | Open vSwitch agent |␣
↪network1 | None              | True  | UP    | neutron-openvswitch-
↪agent |
```

(continues on next page)

(continued from previous page)

```
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent     |␣
↪compute1 | None                 | True  | UP    | neutron-metadata-
↪agent    |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent         |␣
↪compute1 | nova                 | True  | UP    | neutron-dhcp-agent   ␣
↪      |
| bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1 | Open vSwitch agent |␣
↪compute1 | None                 | True  | UP    | neutron-openvswitch-
↪agent |
+--------------------------------------+--------------------+---------
↪-+--------------------+-------+-------+---------------------------+
```

Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use distributed routing. The following procedure creates an additional self-service network and router. The Networking service also supports adding distributed routing to existing routers.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

```
$ openstack network create selfservice2
+-------------------------+--------------+
| Field                   | Value        |
+-------------------------+--------------+
| admin_state_up          | UP           |
| mtu                     | 1450         |
| name                    | selfservice2 |
| port_security_enabled   | True         |
| revision_number         | 1            |
| router:external         | Internal     |
| shared                  | False        |
| status                  | ACTIVE       |
| tags                    | []           |
+-------------------------+--------------+
```

3. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
  --network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4
+-------------------+---------------------------+
| Field             | Value                     |
+-------------------+---------------------------+
| allocation_pools  | 192.0.2.2-192.0.2.254     |
| cidr              | 192.0.2.0/24              |
| dns_nameservers   | 8.8.4.4                   |
| enable_dhcp       | True                      |
| gateway_ip        | 192.0.2.1                 |
| ip_version        | 4                         |
| name              | selfservice2-v4           |
| revision_number   | 1                         |
```

(continues on next page)

(continued from previous page)

```
| tags              | []                         |
+-------------------+----------------------------+
```

4. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-
↪version 6 \
  --ipv6-ra-mode slaac --ipv6-address-mode slaac --network␣
↪selfservice2 \
  --dns-nameserver 2001:4860:4860::8844 selfservice2-v6
+-------------------+------------------------------------------------
↪-----+
| Field             | Value                                          ␣
↪     |
+-------------------+------------------------------------------------
↪-----+
| allocation_pools  | fd00:192:0:2::2-
↪fd00:192:0:2:ffff:ffff:ffff:ffff     |
| cidr              | fd00:192:0:2::/64                              ␣
↪     |
| dns_nameservers   | 2001:4860:4860::8844                           ␣
↪     |
| enable_dhcp       | True                                           ␣
↪     |
| gateway_ip        | fd00:192:0:2::1                                ␣
↪     |
| ip_version        | 6                                              ␣
↪     |
| ipv6_address_mode | slaac                                          ␣
↪     |
| ipv6_ra_mode      | slaac                                          ␣
↪     |
| name              | selfservice2-v6                                ␣
↪     |
| revision_number   | 1                                              ␣
↪     |
| tags              | []                                             ␣
↪     |
+-------------------+------------------------------------------------
↪-----+
```

5. Create a router.

```
$ openstack router create router2
+-----------------------+----------+
| Field                 | Value    |
+-----------------------+----------+
| admin_state_up        | UP       |
| name                  | router2  |
| revision_number       | 1        |
| status                | ACTIVE   |
| tags                  | []       |
+-----------------------+----------+
```

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4
$ openstack router add subnet router2 selfservice2-v6
```

Note: These commands provide no output.

7. Add the provider network as a gateway on the router.

```
$ openstack router set router2 --external-gateway provider1
```

Verify network operation

1. Source the administrative project credentials.
2. Verify distributed routing on the router.

```
$ openstack router show router2
+-------------------------+---------+
| Field                   | Value   |
+-------------------------+---------+
| admin_state_up          | UP      |
| distributed             | True    |
| ha                      | False   |
| name                    | router2 |
| revision_number         | 1       |
| status                  | ACTIVE  |
+-------------------------+---------+
```

3. On each compute node, verify creation of a `qrouter` namespace with the same ID.

Compute node 1:

```
# ip netns
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

Compute node 2:

```
# ip netns
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

4. On the network node, verify creation of the `snat` and `qrouter` namespaces with the same ID.

```
# ip netns
snat-78d2f628-137c-4f26-a257-25fc20f203c1
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

Note: The namespace for router 1 from *Open vSwitch: Self-service networks* should also appear on network node 1 because of creation prior to enabling distributed routing.

5. Launch an instance with an interface on the additional self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-
↪id=NETWORK_ID selfservice-instance2
```

Replace NETWORK_ID with the ID of the additional self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+--------------------------------------+-----------------------+------
↪--+---------------------------------------------------------+------
↪--+---------+
| ID                                   | Name                  |␣
↪Status | Networks                                                |␣
↪Image  | Flavor  |
+--------------------------------------+-----------------------+------
↪--+---------------------------------------------------------+------
↪--+---------+
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 |␣
↪ACTIVE | selfservice2=fd00:192:0:2:f816:3eff:fe71:e93e, 192.0.2.4 |␣
↪cirros | m1.tiny |
+--------------------------------------+-----------------------+------
↪--+---------------------------------------------------------+------
↪--+---------+
```

7. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| fixed_ip          | None                                 |
| id                | 0174056a-fa56-4403-b1ea-b5151a31191f |
| instance_id       | None                                 |
| ip                | 203.0.113.17                         |
| pool              | provider1                            |
| revision_number   | 1                                    |
| tags              | []                                   |
+-------------------+--------------------------------------+
```

8. Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

Note: This command provides no output.

9. On the compute node containing the instance, verify creation of the fip namespace with the same ID as the provider network.

```
# ip netns
fip-4bfa3075-b4b2-4f7d-b88e-df1113942d43
```

Network traffic flow

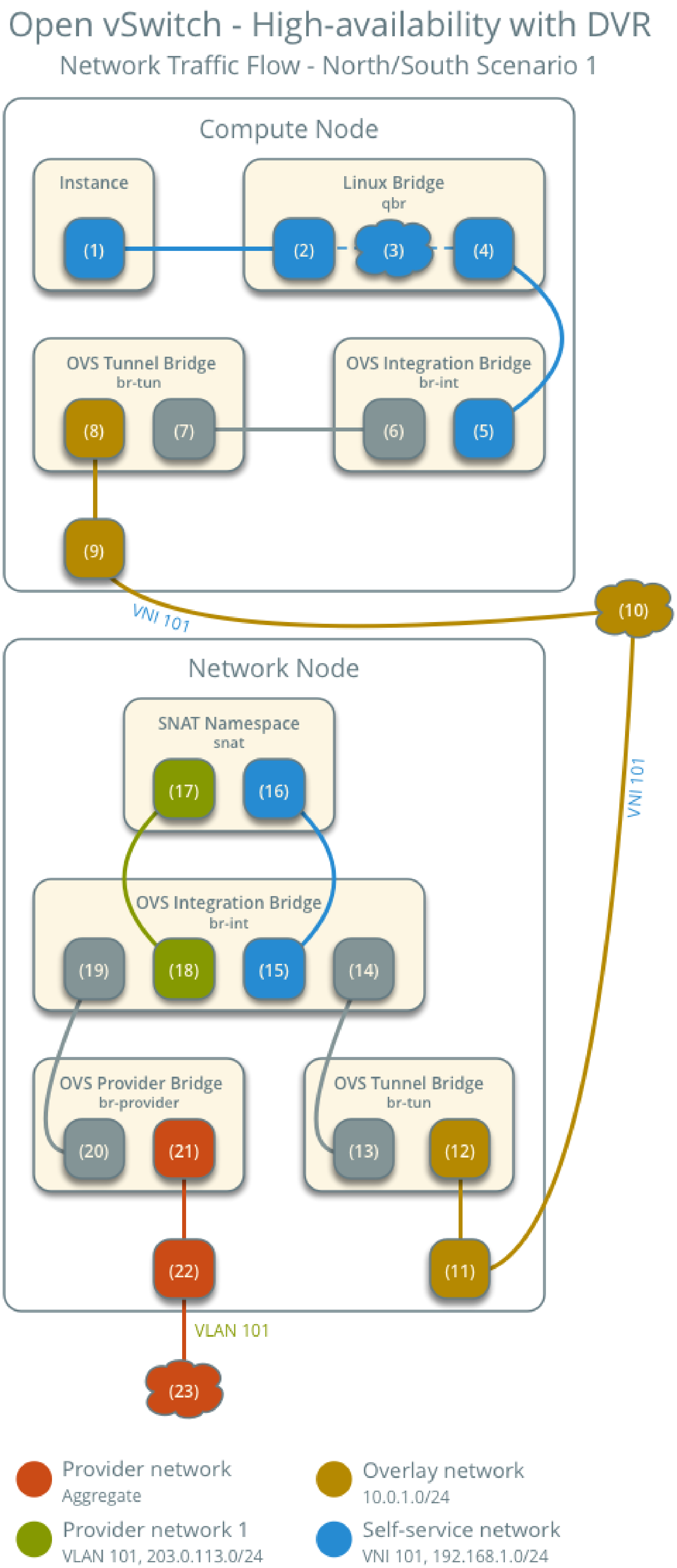
The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
  - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
  - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
  - VXLAN ID (VNI) 102
- Self-service router
  - Gateway on the provider network
  - Interface on self-service network 1
  - Interface on self-service network 2
- Instance 1
- Instance 2

This section only contains flow scenarios that benefit from distributed virtual routing or that differ from conventional operation. For other flow scenarios, see *Network traffic flow*.

North-south scenario 1: Instance with a fixed IP address

Similar to *North-south scenario 1: Instance with a fixed IP address*, except the router namespace on the network node becomes the SNAT namespace. The network node still contains the router namespace, but it serves no purpose in this case.


Open vSwitch - High-availability with DVR
Network Traffic Flow - North/South Scenario 1
Compute Node
Instance
(1)
Linux Bridge
qbr
(2)
(3)
(4)
OVS Tunnel Bridge
br-tun
(8)
(7)
OVS Integration Bridge
br-int
(6)
(5)
(9)
VNI 101
(10)
Network Node
SNAT Namespace
snat
(17)
(16)
VNI 101
OVS Integration Bridge
br-int
(19)
(18)
(15)
(14)
OVS Provider Bridge
br-provider
(20)
(21)
OVS Tunnel Bridge
br-tun
(13)
(12)
(22)
(11)
VLAN 101
(23)
Provider network
Aggregate
Overlay network
10.0.1.0/24
Provider network 1
VLAN 101, 203.0.113.0/24
Self-service network
VNI 101, 192.168.1.0/24

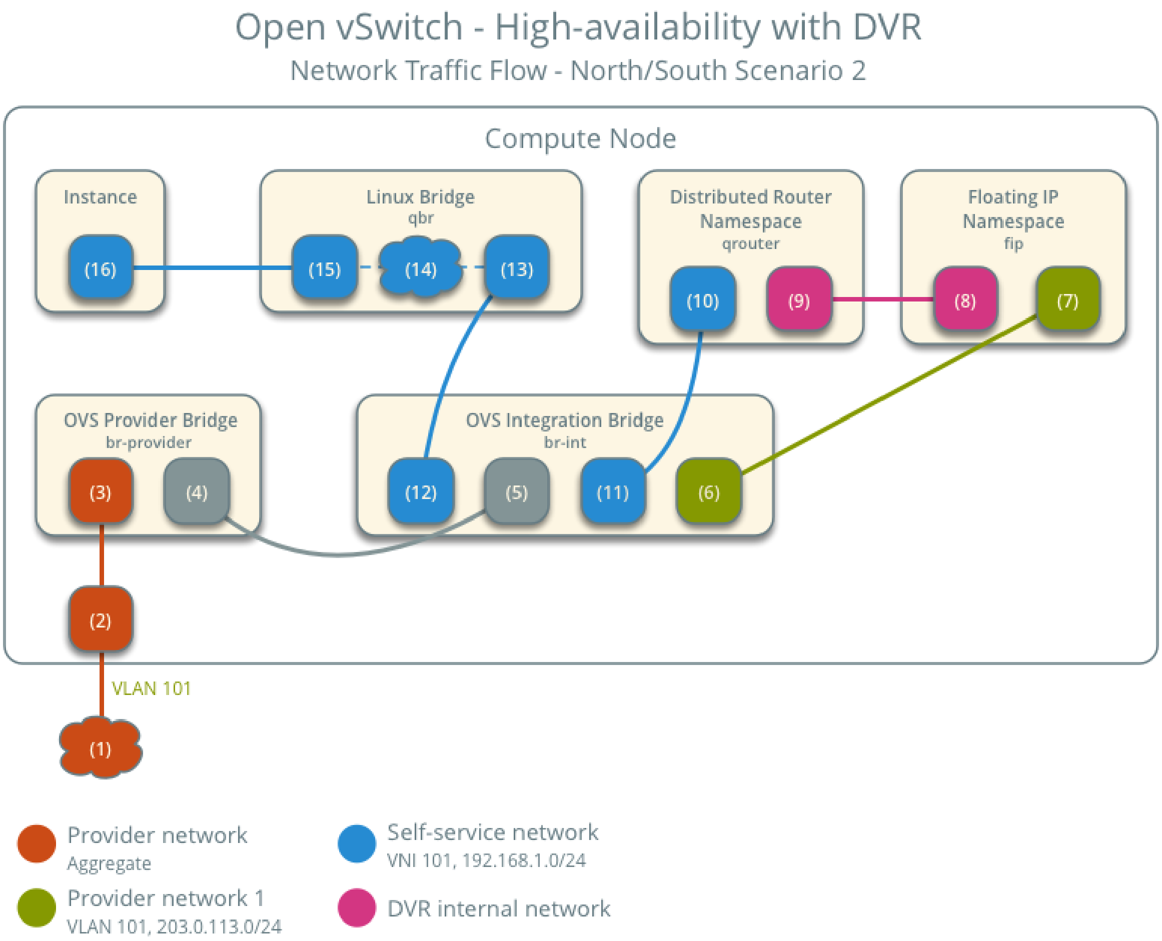
North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address using a self-service network on a distributed router, the compute node containing the instance performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario. north-south traffic passing between the instance and external networks such as the Internet.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

The following steps involve the compute node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface (2).
2. The provider physical network interface forwards the packet to the OVS provider bridge provider network port (3).
3. The OVS provider bridge swaps actual VLAN tag 101 with the internal VLAN tag.
4. The OVS provider bridge `phy-br-provider` port (4) forwards the packet to the OVS integration bridge `int-br-provider` port (5).
5. The OVS integration bridge port for the provider network (6) removes the internal VLAN tag and forwards the packet to the provider network interface (7) in the floating IP namespace. This interface responds to any ARP requests for the instance floating IPv4 address.
6. The floating IP namespace routes the packet (8) to the distributed router namespace (9) using a pair of IP addresses on the DVR internal network. This namespace contains the instance floating IPv4 address.
7. The router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network via the self-service network interface (10).
8. The router forwards the packet to the OVS integration bridge port for the self-service network (11).
9. The OVS integration bridge adds an internal VLAN tag to the packet.
10. The OVS integration bridge removes the internal VLAN tag from the packet.
11. The OVS integration bridge security group port (12) forwards the packet to the security group bridge OVS port (13) via `veth` pair.
12. Security group rules (14) on the security group bridge handle firewalling and connection tracking for the packet.
13. The security group bridge instance port (15) forwards the packet to the instance interface (16) via `veth` pair.



Note: Egress traffic follows similar steps in reverse, except SNAT changes the source IPv4 address of the packet to the floating IPv4 address.

East-west scenario 1: Instances on different networks on the same router

Instances with fixed IPv4/IPv6 address or floating IPv4 address on the same compute node communicate via router on the compute node. Instances on different compute nodes communicate via an instance of the router on each compute node.

Note: This scenario places the instances on different compute nodes to show the most complex situation.

The following steps involve compute node 1:

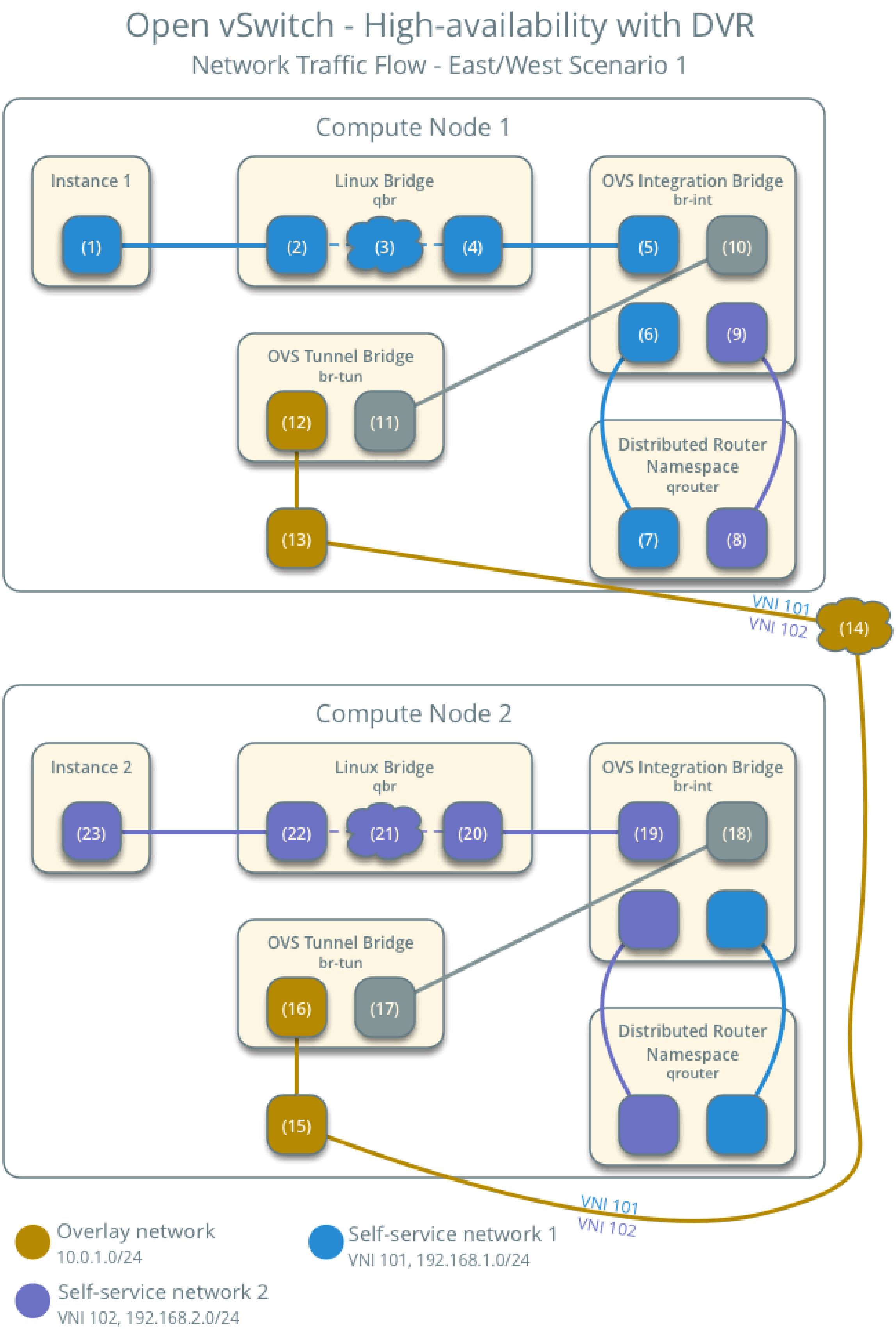
1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via `veth` pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.

3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via `veth` pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge port for self-service network 1 (6) removes the internal VLAN tag and forwards the packet to the self-service network 1 interface in the distributed router namespace (6).
6. The distributed router namespace routes the packet to self-service network 2.
7. The self-service network 2 interface in the distributed router namespace (8) forwards the packet to the OVS integration bridge port for self-service network 2 (9).
8. The OVS integration bridge adds an internal VLAN tag to the packet.
9. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
10. The OVS integration bridge `patch-tun` port (10) forwards the packet to the OVS tunnel bridge `patch-int` port (11).
11. The OVS tunnel bridge (12) wraps the packet using VNI 101.
12. The underlying physical interface (13) for overlay networks forwards the packet to compute node 2 via the overlay network (14).

The following steps involve compute node 2:

1. The underlying physical interface (15) for overlay networks forwards the packet to the OVS tunnel bridge (16).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge `patch-int` patch port (17) forwards the packet to the OVS integration bridge `patch-tun` patch port (18).
5. The OVS integration bridge removes the internal VLAN tag from the packet.
6. The OVS integration bridge security group port (19) forwards the packet to the security group bridge OVS port (20) via `veth` pair.
7. Security group rules (21) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (22) forwards the packet to the instance 2 interface (23) via `veth` pair.

Note: Routing between self-service networks occurs on the compute node containing the instance sending the packet. In this scenario, routing occurs on compute node 1 for packets from instance 1 to instance 2 and on compute node 2 for packets from instance 2 to instance 1.



Open vSwitch - High-availability with DVR
Network Traffic Flow - East/West Scenario 1
Compute Node 1
Instance 1
(1)
Linux Bridge
qbr
(2)
(3)
(4)
OVS Integration Bridge
br-int
(5)
(10)
(6)
(9)
OVS Tunnel Bridge
br-tun
(12)
(11)
Distributed Router
Namespace
qrouter
(7)
(8)
(13)
VNI 101
VNI 102
(14)
Compute Node 2
Instance 2
(23)
Linux Bridge
qbr
(22)
(21)
(20)
OVS Integration Bridge
br-int
(19)
(18)
OVS Tunnel Bridge
br-tun
(16)
(17)
Distributed Router
Namespace
qrouter
(15)
VNI 101
VNI 102
Overlay network
10.0.1.0/24
Self-service network 1
VNI 101, 192.168.1.0/24
Self-service network 2
VNI 102, 192.168.2.0/24

8.4 Operations

8.4.1 IP availability metrics

Network IP Availability is an information-only API extension that allows a user or process to determine the number of IP addresses that are consumed across networks and the allocation pools of their subnets. This extension was added to neutron in the Mitaka release.

This section illustrates how you can get the Network IP address availability through the command-line interface.

Get Network IP address availability for all IPv4 networks:

```
$ openstack ip availability list

+--------------------------------------+--------------+-----------+--------
↪--+
| Network ID                           | Network Name | Total IPs | Used␣
↪IPs |
+--------------------------------------+--------------+-----------+--------
↪--+
| 363a611a-b08b-4281-b64e-198d90cb94fd | private      |       253 |        ␣
↪3 |
| c92d0605-caf2-4349-b1b8-8d5f9ac91df8 | public       |       253 |        ␣
↪1 |
+--------------------------------------+--------------+-----------+--------
↪--+
```

Get Network IP address availability for all IPv6 networks:

```
$ openstack ip availability list --ip-version 6

+--------------------------------------+--------------+--------------------
↪--+----------+
| Network ID                           | Network Name | Total IPs          ␣
↪   | Used IPs |
+--------------------------------------+--------------+--------------------
↪--+----------+
| 363a611a-b08b-4281-b64e-198d90cb94fd | private      |␣
↪18446744073709551614 |        3 |
| c92d0605-caf2-4349-b1b8-8d5f9ac91df8 | public       |␣
↪18446744073709551614 |        1 |
+--------------------------------------+--------------+--------------------
↪--+----------+
```

Get Network IP address availability statistics for a specific network:

```
$ openstack ip availability show NETWORKUUID

+------------------------+--------------------------------------------------
↪-------------+
| Field                  | Value                                            ␣
↪             |
+------------------------+--------------------------------------------------
↪-------------+
| network_id             | 0bf90de6-fc0f-4dba-b80d-96670dfb331a             ␣
↪             |
```

(continues on next page)

(continued from previous page)

```
| network_name           | public                                                 ␣
↪             |
| project_id             | 5669caad86a04256994cdf755df4d3c1                       ␣
↪             |
| subnet_ip_availability | cidr='192.0.2.224/28', ip_version='4', subnet_
↪id='346806ee-   |
|                        | a53e-44fd-968a-ddb2bcd2ba96', subnet_name=
↪'public_subnet',    |
|                        | total_ips='13', used_ips='5'                           ␣
↪             |
| total_ips              | 13                                                     ␣
↪             |
| used_ips               | 5                                                      ␣
↪             |
+------------------------+--------------------------------------------------
↪-------------+
```

8.4.2 Resource tags

Various virtual networking resources support tags for use by external systems or any other clients of the Networking service API.

All resources that support standard attributes are applicable for tagging. This includes:

- networks
- subnets
- subnetpools
- ports
- routers
- floatingips
- logs
- security-groups
- security-group-rules
- segments
- policies
- trunks
- network_segment_ranges

Use cases

The following use cases refer to adding tags to networks, but the same can be applicable to any other supported Networking service resource:

1. Ability to map different networks in different OpenStack locations to one logically same network (for multi-site OpenStack).
2. Ability to map IDs from different management/orchestration systems to OpenStack networks in mixed environments. For example, in the Kuryr project, the Docker network ID is mapped to the Neutron network ID.
3. Ability to leverage tags by deployment tools.
4. Ability to tag information about provider networks (for example, high-bandwidth, low-latency, and so on).

Filtering with tags

The API allows searching/filtering of the `GET /v2.0/networks` API. The following query parameters are supported:

- `tags`
- `tags-any`
- `not-tags`
- `not-tags-any`

To request the list of networks that have a single tag, `tags` argument should be set to the desired tag name. Example:

```
GET /v2.0/networks?tags=red
```

To request the list of networks that have two or more tags, the `tags` argument should be set to the list of tags, separated by commas. In this case, the tags given must all be present for a network to be included in the query result. Example that returns networks that have the red and blue tags:

```
GET /v2.0/networks?tags=red,blue
```

To request the list of networks that have one or more of a list of given tags, the `tags-any` argument should be set to the list of tags, separated by commas. In this case, as long as one of the given tags is present, the network will be included in the query result. Example that returns the networks that have the red or the blue tag:

```
GET /v2.0/networks?tags-any=red,blue
```

To request the list of networks that do not have one or more tags, the `not-tags` argument should be set to the list of tags, separated by commas. In this case, only the networks that do not have any of the given tags will be included in the query results. Example that returns the networks that do not have either red or blue tag:

```
GET /v2.0/networks?not-tags=red,blue
```

To request the list of networks that do not have at least one of a list of tags, the `not-tags-any` argument should be set to the list of tags, separated by commas. In this case, only the networks that do not have at least one of the given tags will be included in the query result. Example that returns the networks that do not have the red tag, or do not have the blue tag:

```
GET /v2.0/networks?not-tags-any=red,blue
```

The `tags`, `tags-any`, `not-tags`, and `not-tags-any` arguments can be combined to build more complex queries. Example:

```
GET /v2.0/networks?tags=red,blue&tags-any=green,orange
```

The above example returns any networks that have the red and blue tags, plus at least one of green and orange.

Complex queries may have contradictory parameters. Example:

```
GET /v2.0/networks?tags=blue&not-tags=blue
```

In this case, we should let the Networking service find these networks. Obviously, there are no such networks and the service will return an empty list.

User workflow

Add a tag to a resource:

```
$ openstack network set --tag red ab442634-1cc9-49e5-bd49-0dac9c811f69
$ openstack network show net
+---------------------------+----------------------------------------------
↪------------------------------+
| Field                     | Value                                        ␣
↪                              |
+---------------------------+----------------------------------------------
↪------------------------------+
| admin_state_up            | UP                                           ␣
↪                              |
| availability_zone_hints   |                                              ␣
↪                              |
| availability_zones        | nova                                         ␣
↪                              |
| created_at                | 2018-07-11T09:44:50Z                         ␣
↪                              |
| description               |                                              ␣
↪                              |
| dns_domain                | None                                         ␣
↪                              |
| id                        | ab442634-1cc9-49e5-bd49-0dac9c811f69         ␣
↪                              |
| ipv4_address_scope        | None                                         ␣
↪                              |
| ipv6_address_scope        | None                                         ␣
↪                              |
| is_default                | None                                         ␣
↪                              |
| is_vlan_transparent       | None                                         ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| mtu                       | 1450                                        ␣
↪                               |
| name                      | net                                         ␣
↪                               |
| port_security_enabled     | True                                        ␣
↪                               |
| project_id                | e6710680bfd14555891f265644e1dd5c            ␣
↪                               |
| provider:network_type     | vxlan                                       ␣
↪                               |
| provider:physical_network | None                                        ␣
↪                               |
| provider:segmentation_id  | 1047                                        ␣
↪                               |
| qos_policy_id             | None                                        ␣
↪                               |
| revision_number           | 5                                           ␣
↪                               |
| router:external           | Internal                                    ␣
↪                               |
| segments                  | None                                        ␣
↪                               |
| shared                    | False                                       ␣
↪                               |
| status                    | ACTIVE                                      ␣
↪                               |
| subnets                   |                                             ␣
↪                               |
| tags                      | red                                         ␣
↪                               |
| updated_at                | 2018-07-16T06:22:01Z                        ␣
↪                               |
+---------------------------+---------------------------------------------
↪-------------------------------+
```

Remove a tag from a resource:

```
$ openstack network unset --tag red ab442634-1cc9-49e5-bd49-0dac9c811f69
$ openstack network show net
+---------------------------+---------------------------------------------
↪-------------------------------+
| Field                     | Value                                       ␣
↪                               |
+---------------------------+---------------------------------------------
↪-------------------------------+
| admin_state_up            | UP                                          ␣
↪                               |
| availability_zone_hints   |                                             ␣
↪                               |
| availability_zones        | nova                                        ␣
↪                               |
| created_at                | 2018-07-11T09:44:50Z                        ␣
↪                               |
| description               |                                             ␣
↪                               |
| dns_domain                | None                                        ␣
↪                               |
```

(continues on next page)

(continued from previous page)

```
| id                        | ab442634-1cc9-49e5-bd49-0dac9c811f69                                    ␣
↪                           |
| ipv4_address_scope        | None                                                                    ␣
↪                           |
| ipv6_address_scope        | None                                                                    ␣
↪                           |
| is_default                | None                                                                    ␣
↪                           |
| is_vlan_transparent       | None                                                                    ␣
↪                           |
| mtu                       | 1450                                                                    ␣
↪                           |
| name                      | net                                                                     ␣
↪                           |
| port_security_enabled     | True                                                                    ␣
↪                           |
| project_id                | e6710680bfd14555891f265644e1dd5c                                        ␣
↪                           |
| provider:network_type     | vxlan                                                                   ␣
↪                           |
| provider:physical_network | None                                                                    ␣
↪                           |
| provider:segmentation_id  | 1047                                                                    ␣
↪                           |
| qos_policy_id             | None                                                                    ␣
↪                           |
| revision_number           | 5                                                                       ␣
↪                           |
| router:external           | Internal                                                                ␣
↪                           |
| segments                  | None                                                                    ␣
↪                           |
| shared                    | False                                                                   ␣
↪                           |
| status                    | ACTIVE                                                                  ␣
↪                           |
| subnets                   |                                                                         ␣
↪                           |
| tags                      |                                                                         ␣
↪                           |
| updated_at                | 2018-07-16T06:32:11Z                                                    ␣
↪                           |
+---------------------------+-----------------------------------------
↪-----------------------------+
```

Replace all tags on the resource:

```
$ openstack network set --tag red --tag blue ab442634-1cc9-49e5-bd49-
↪0dac9c811f69
$ openstack network show net
+---------------------------+-----------------------------------------
↪-----------------------------+
| Field                     | Value                                                                   ␣
↪                           |
+---------------------------+-----------------------------------------
↪-----------------------------+
```

(continues on next page)

(continued from previous page)

```
| admin_state_up            | UP                                    ␣
↪                               |
| availability_zone_hints   |                                       ␣
↪                               |
| availability_zones        | nova                                  ␣
↪                               |
| created_at                | 2018-07-11T09:44:50Z                  ␣
↪                               |
| description               |                                       ␣
↪                               |
| dns_domain                | None                                  ␣
↪                               |
| id                        | ab442634-1cc9-49e5-bd49-0dac9c811f69  ␣
↪                               |
| ipv4_address_scope        | None                                  ␣
↪                               |
| ipv6_address_scope        | None                                  ␣
↪                               |
| is_default                | None                                  ␣
↪                               |
| is_vlan_transparent       | None                                  ␣
↪                               |
| mtu                       | 1450                                  ␣
↪                               |
| name                      | net                                   ␣
↪                               |
| port_security_enabled     | True                                  ␣
↪                               |
| project_id                | e6710680bfd14555891f265644e1dd5c      ␣
↪                               |
| provider:network_type     | vxlan                                 ␣
↪                               |
| provider:physical_network | None                                  ␣
↪                               |
| provider:segmentation_id  | 1047                                  ␣
↪                               |
| qos_policy_id             | None                                  ␣
↪                               |
| revision_number           | 5                                     ␣
↪                               |
| router:external           | Internal                              ␣
↪                               |
| segments                  | None                                  ␣
↪                               |
| shared                    | False                                 ␣
↪                               |
| status                    | ACTIVE                                ␣
↪                               |
| subnets                   |                                       ␣
↪                               |
| tags                      | blue, red                             ␣
↪                               |
| updated_at                | 2018-07-16T06:50:19Z                  ␣
↪                               |
+---------------------------+-----------------------------------------
↪-----------------------------+
```

Clear tags from a resource:

```
$ openstack network unset --all-tag ab442634-1cc9-49e5-bd49-0dac9c811f69
$ openstack network show net
+---------------------------+---------------------------------------------
↪-------------------------------+
| Field                     | Value                                       ␣
↪                               |
+---------------------------+---------------------------------------------
↪-------------------------------+
| admin_state_up            | UP                                          ␣
↪                               |
| availability_zone_hints   |                                             ␣
↪                               |
| availability_zones        | nova                                        ␣
↪                               |
| created_at                | 2018-07-11T09:44:50Z                        ␣
↪                               |
| description               |                                             ␣
↪                               |
| dns_domain                | None                                        ␣
↪                               |
| id                        | ab442634-1cc9-49e5-bd49-0dac9c811f69        ␣
↪                               |
| ipv4_address_scope        | None                                        ␣
↪                               |
| ipv6_address_scope        | None                                        ␣
↪                               |
| is_default                | None                                        ␣
↪                               |
| is_vlan_transparent       | None                                        ␣
↪                               |
| mtu                       | 1450                                        ␣
↪                               |
| name                      | net                                         ␣
↪                               |
| port_security_enabled     | True                                        ␣
↪                               |
| project_id                | e6710680bfd14555891f265644e1dd5c            ␣
↪                               |
| provider:network_type     | vxlan                                       ␣
↪                               |
| provider:physical_network | None                                        ␣
↪                               |
| provider:segmentation_id  | 1047                                        ␣
↪                               |
| qos_policy_id             | None                                        ␣
↪                               |
| revision_number           | 5                                           ␣
↪                               |
| router:external           | Internal                                    ␣
↪                               |
| segments                  | None                                        ␣
↪                               |
| shared                    | False                                       ␣
↪                               |
| status                    | ACTIVE                                      ␣
↪                               |
```

(continues on next page)

(continued from previous page)

```
| subnets                 |                                          ␣
↪                             |
| tags                    |                                          ␣
↪                             |
| updated_at              | 2018-07-16T07:03:02Z                     ␣
↪                             |
+-------------------------+------------------------------------------
↪-----------------------------+
```

Get list of resources with tag filters from networks. The networks are: test-net1 with red tag, test-net2 with red and blue tags, test-net3 with red, blue, and green tags, and test-net4 with green tag.

Get list of resources with `tags` filter:

```
$ openstack network list --tags red,blue
+--------------------------------------+-----------+---------+
| ID                                   | Name      | Subnets |
+--------------------------------------+-----------+---------+
| 8ca3b9ed-f578-45fa-8c44-c53f13aec05a | test-net3 |         |
| e736e63d-42e4-4f4c-836c-6ad286ffd68a | test-net2 |         |
+--------------------------------------+-----------+---------+
```

Get list of resources with `any-tags` filter:

```
$ openstack network list --any-tags red,blue
+--------------------------------------+-----------+---------+
| ID                                   | Name      | Subnets |
+--------------------------------------+-----------+---------+
| 30491224-3855-431f-a688-fb29df004d82 | test-net1 |         |
| 8ca3b9ed-f578-45fa-8c44-c53f13aec05a | test-net3 |         |
| e736e63d-42e4-4f4c-836c-6ad286ffd68a | test-net2 |         |
+--------------------------------------+-----------+---------+
```

Get list of resources with `not-tags` filter:

```
$ openstack network list --not-tags red,blue
+--------------------------------------+-----------+---------+
| ID                                   | Name      | Subnets |
+--------------------------------------+-----------+---------+
| 30491224-3855-431f-a688-fb29df004d82 | test-net1 |         |
| cdb3ed08-ca63-4090-ba12-30b366372993 | test-net4 |         |
+--------------------------------------+-----------+---------+
```

Get list of resources with `not-any-tags` filter:

```
$ openstack network list --not-any-tags red,blue
+--------------------------------------+-----------+---------+
| ID                                   | Name      | Subnets |
+--------------------------------------+-----------+---------+
| cdb3ed08-ca63-4090-ba12-30b366372993 | test-net4 |         |
+--------------------------------------+-----------+---------+
```

Limitations

Filtering resources with a tag whose name contains a comma is not supported. Thus, do not put such a tag name to resources.

Future support

In future releases, the Networking service may support setting tags for additional resources.

8.4.3 Resource purge

The Networking service provides a purge mechanism to delete the following network resources for a project:

- Networks
- Subnets
- Ports
- Router interfaces
- Routers
- Floating IP addresses
- Security groups

Typically, one uses this mechanism to delete networking resources for a defunct project regardless of its existence in the Identity service.

Usage

1. Source the necessary project credentials. The administrative project can delete resources for all other projects. A regular project can delete its own network resources and those belonging to other projects for which it has sufficient access.
2. Delete the network resources for a particular project.

   ```
   $ neutron purge PROJECT_ID
   ```

   Replace `PROJECT_ID` with the project ID.

The command provides output that includes a completion percentage and the quantity of successful or unsuccessful network resource deletions. An unsuccessful deletion usually indicates sharing of a resource with one or more additional projects.

```
Purging resources: 100% complete.
Deleted 1 security_group, 2 ports, 1 router, 1 floatingip, 2 networks.
The following resources could not be deleted: 1 network.
```

The command also indicates if a project lacks network resources.

```
Tenant has no supported resources.
```

8.4.4 Manage Networking service quotas

A quota limits the number of available resources. A default quota might be enforced for all projects. When you try to create more resources than the quota allows, an error occurs:

```
$ openstack network create test_net
 Quota exceeded for resources: ['network']
```

Per-project quota configuration is also supported by the quota extension API. See *Configure per-project quotas* for details.

Basic quota configuration

In the Networking default quota mechanism, all projects have the same quota values, such as the number of resources that a project can create.

The quota value is defined in the OpenStack Networking `/etc/neutron/neutron.conf` configuration file. This example shows the default quota values:

```
[quotas]
# number of networks allowed per tenant, and minus means unlimited
quota_network = 10

# number of subnets allowed per tenant, and minus means unlimited
quota_subnet = 10

# number of ports allowed per tenant, and minus means unlimited
quota_port = 50

# default driver to use for quota checks
quota_driver = neutron.quota.ConfDriver
```

OpenStack Networking also supports quotas for L3 resources: router and floating IP. Add these lines to the `quotas` section in the `/etc/neutron/neutron.conf` file:

```
[quotas]
# number of routers allowed per tenant, and minus means unlimited
quota_router = 10

# number of floating IPs allowed per tenant, and minus means unlimited
quota_floatingip = 50
```

OpenStack Networking also supports quotas for security group resources: number of security groups and the number of rules for each security group. Add these lines to the `quotas` section in the `/etc/neutron/neutron.conf` file:

```
[quotas]
# number of security groups per tenant, and minus means unlimited
quota_security_group = 10

# number of security rules allowed per tenant, and minus means unlimited
quota_security_group_rule = 100
```

Configure per-project quotas

OpenStack Networking also supports per-project quota limit by quota extension API.

Todo: This document needs to be migrated to using `openstack` commands rather than the deprecated `neutron` commands.

Use these commands to manage per-project quotas:

neutron quota-delete Delete defined quotas for a specified project

neutron quota-list Lists defined quotas for all projects

neutron quota-show Shows quotas for a specified project

neutron quota-default-show Show default quotas for a specified tenant

neutron quota-update Updates quotas for a specified project

Only users with the `admin` role can change a quota value. By default, the default set of quotas are enforced for all projects, so no **`quota-create`** command exists.

1. Configure Networking to show per-project quotas

   Set the `quota_driver` option in the `/etc/neutron/neutron.conf` file.

   ```
   quota_driver = neutron.db.quota_db.DbQuotaDriver
   ```

   When you set this option, the output for Networking commands shows `quotas`.

2. List Networking extensions.

   To list the Networking extensions, run this command:

   ```
   $ openstack extension list --network
   ```

   The command shows the `quotas` extension, which provides per-project quota management support.

   Note: Many of the extensions shown below are supported in the Mitaka release and later.

   ```
   +------------------------+------------------------+--------------------
   ↪-------+
   | Name                   | Alias                  | Description        ␣
   ↪       |
   +------------------------+------------------------+--------------------
   ↪-------+
   | ...                    | ...                    | ...                ␣
   ↪       |
   | Quota management       | quotas                 | Expose functions␣
   ↪for    |
   | support                |                        | quotas management␣
   ↪per    |
   |                        |                        | tenant             ␣
   ↪       |
   ```

(continues on next page)

(continued from previous page)

```
| ...                      | ...                      | ...                  ␣
↪        |
+------------------------+------------------------+-------------------
↪-------+
```

3. Show information for the quotas extension.

   To show information for the `quotas` extension, run this command:

```
$ neutron ext-show quotas
+-------------+--------------------------------------------------------
↪-----+
| Field       | Value                                                 ␣
↪     |
+-------------+--------------------------------------------------------
↪-----+
| alias       | quotas                                                ␣
↪     |
| description | Expose functions for quotas management per tenant     ␣
↪     |
| links       |                                                       ␣
↪     |
| name        | Quota management support                              ␣
↪     |
| namespace   | https://docs.openstack.org/network/ext/quotas-sets/
↪api/v2.0 |
| updated     | 2012-07-29T10:00:00-00:00                             ␣
↪     |
+-------------+--------------------------------------------------------
↪-----+
```

Note: Only some plug-ins support per-project quotas. Specifically, Open vSwitch, Linux Bridge, and VMware NSX support them, but new versions of other plug-ins might bring additional functionality. See the documentation for each plug-in.

4. List projects who have per-project quota support.

   The **neutron quota-list** command lists projects for which the per-project quota is enabled. The command does not list projects with default quota support. You must be an administrative user to run this command:

```
$ neutron quota-list
+------------+---------+------+--------+--------+----------------------
↪-------------+
| floatingip | network | port | router | subnet | tenant_id            ␣
↪             |
+------------+---------+------+--------+--------+----------------------
↪-------------+
|         20 |       5 |   20 |     10 |      5 |␣
↪6f88036c45344d9999a1f971e4882723 |
|         25 |      10 |   30 |     10 |     10 |␣
↪bff5c9455ee24231b5bc713c1b96d422 |
+------------+---------+------+--------+--------+----------------------
↪-------------+
```

5. Show per-project quota values.

   The **neutron quota-show** command reports the current set of quota limits for the specified project. Non-administrative users can run this command without the `--tenant_id` parameter. If per-project quota limits are not enabled for the project, the command shows the default set of quotas.

   Note: Additional quotas added in the Mitaka release include `security_group`, `security_group_rule`, `subnet`, and `subnetpool`.

```
$ neutron quota-show --tenant_id 6f88036c45344d9999a1f971e4882723
+---------------------+-------+
| Field               | Value |
+---------------------+-------+
| floatingip          | 50    |
| network             | 10    |
| port                | 50    |
| rbac_policy         | 10    |
| router              | 10    |
| security_group      | 10    |
| security_group_rule | 100   |
| subnet              | 10    |
| subnetpool          | -1    |
+---------------------+-------+
```

   The following command shows the command output for a non-administrative user.

```
$ neutron quota-show
+---------------------+-------+
| Field               | Value |
+---------------------+-------+
| floatingip          | 50    |
| network             | 10    |
| port                | 50    |
| rbac_policy         | 10    |
| router              | 10    |
| security_group      | 10    |
| security_group_rule | 100   |
| subnet              | 10    |
| subnetpool          | -1    |
+---------------------+-------+
```

6. Update quota values for a specified project.

   Use the **neutron quota-update** command to update a quota for a specified project.

```
$ neutron quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --
↪network 5
+---------------------+-------+
| Field               | Value |
+---------------------+-------+
| floatingip          | 50    |
| network             | 5     |
| port                | 50    |
| rbac_policy         | 10    |
```

(continues on next page)

(continued from previous page)

```
| router               | 10     |
| security_group       | 10     |
| security_group_rule  | 100    |
| subnet               | 10     |
| subnetpool           | -1     |
+----------------------+--------+
```

You can update quotas for multiple resources through one command.

```
$ neutron quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --
↪subnet 5 --port 20
+----------------------+--------+
| Field                | Value  |
+----------------------+--------+
| floatingip           | 50     |
| network              | 5      |
| port                 | 20     |
| rbac_policy          | 10     |
| router               | 10     |
| security_group       | 10     |
| security_group_rule  | 100    |
| subnet               | 5      |
| subnetpool           | -1     |
+----------------------+--------+
```

To update the limits for an L3 resource such as, router or floating IP, you must define new values for the quotas after the -- directive.

This example updates the limit of the number of floating IPs for the specified project.

```
$ neutron quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --
↪floatingip 20
+----------------------+--------+
| Field                | Value  |
+----------------------+--------+
| floatingip           | 20     |
| network              | 5      |
| port                 | 20     |
| rbac_policy          | 10     |
| router               | 10     |
| security_group       | 10     |
| security_group_rule  | 100    |
| subnet               | 5      |
| subnetpool           | -1     |
+----------------------+--------+
```

You can update the limits of multiple resources by including L2 resources and L3 resource through one command:

```
$ neutron quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 \
  --network 3 --subnet 3 --port 3 --floatingip 3 --router 3
+----------------------+--------+
| Field                | Value  |
+----------------------+--------+
| floatingip           | 3      |
| network              | 3      |
```

(continues on next page)

(continued from previous page)

```
| port                 | 3     |
| rbac_policy          | 10    |
| router               | 3     |
| security_group       | 10    |
| security_group_rule  | 100   |
| subnet               | 3     |
| subnetpool           | -1    |
+----------------------+-------+
```

7. Delete per-project quota values.

   To clear per-project quota limits, use the **neutron quota-delete** command.

```
$ neutron quota-delete --tenant_id 6f88036c45344d9999a1f971e4882723
 Deleted quota: 6f88036c45344d9999a1f971e4882723
```

   After you run this command, you can see that quota values for the project are reset to the default values.

```
$ openstack quota show 6f88036c45344d9999a1f971e4882723
+----------------------+-------+
| Field                | Value |
+----------------------+-------+
| floatingip           | 50    |
| network              | 10    |
| port                 | 50    |
| rbac_policy          | 10    |
| router               | 10    |
| security_group       | 10    |
| security_group_rule  | 100   |
| subnet               | 10    |
| subnetpool           | -1    |
+----------------------+-------+
```

Note: Listing defualt quotas with the OpenStack command line client will provide all quotas for networking and other services. Previously, the **neutron quota-show --tenant_id** would list only networking quotas.

8.5 Migration

8.5.1 Database

The upgrade of the Networking service database is implemented with Alembic migration chains. The migrations in the `alembic/versions` contain the changes needed to migrate from older Networking service releases to newer ones.

Since Liberty, Networking maintains two parallel Alembic migration branches.

The first branch is called expand and is used to store expansion-only migration rules. These rules are strictly additive and can be applied while the Neutron server is running.

The second branch is called contract and is used to store those migration rules that are not safe to apply while Neutron server is running.

The intent of separate branches is to allow invoking those safe migrations from the expand branch while the Neutron server is running and therefore reducing downtime needed to upgrade the service.

A database management command-line tool uses the Alembic library to manage the migration.

Database management command-line tool

The database management command-line tool is called **neutron-db-manage**. Pass the `--help` option to the tool for usage information.

The tool takes some options followed by some commands:

```
$ neutron-db-manage <options> <commands>
```

The tool needs to access the database connection string, which is provided in the `neutron.conf` configuration file in an installation. The tool automatically reads from `/etc/neutron/neutron.conf` if it is present. If the configuration is in a different location, use the following command:

```
$ neutron-db-manage --config-file /path/to/neutron.conf <commands>
```

Multiple `--config-file` options can be passed if needed.

Instead of reading the DB connection from the configuration file(s), you can use the `--database-connection` option:

```
$ neutron-db-manage --database-connection
  mysql+pymysql://root:secret@127.0.0.1/neutron?charset=utf8 <commands>
```

The *branches*, *current*, and *history* commands all accept a `--verbose` option, which, when passed, will instruct **neutron-db-manage** to display more verbose output for the specified command:

```
$ neutron-db-manage current --verbose
```

Note: The tool usage examples below do not show the options. It is assumed that you use the options that you need for your environment.

In new deployments, you start with an empty database and then upgrade to the latest database version using the following command:

```
$ neutron-db-manage upgrade heads
```

After installing a new version of the Neutron server, upgrade the database using the following command:

```
$ neutron-db-manage upgrade heads
```

In existing deployments, check the current database version using the following command:

```
$ neutron-db-manage current
```

To apply the expansion migration rules, use the following command:

```
$ neutron-db-manage upgrade --expand
```

To apply the non-expansive migration rules, use the following command:

```
$ neutron-db-manage upgrade --contract
```

To check if any contract migrations are pending and therefore if offline migration is required, use the following command:

```
$ neutron-db-manage has_offline_migrations
```

Note: Offline migration requires all Neutron server instances in the cluster to be shutdown before you apply any contract scripts.

To generate a script of the command instead of operating immediately on the database, use the following command:

```
$ neutron-db-manage upgrade heads --sql

.. note::

   The `--sql` option causes the command to generate a script.  The script
   can be run later (online or offline), perhaps after verifying and/or
   modifying it.
```

To migrate between specific migration versions, use the following command:

```
$ neutron-db-manage upgrade <start version>:<end version>
```

To upgrade the database incrementally, use the following command:

```
$ neutron-db-manage upgrade --delta <# of revs>
```

Note: Database downgrade is not supported.

To look for differences between the schema generated by the upgrade command and the schema defined by the models, use the **`revision --autogenerate`** command:

```
neutron-db-manage revision -m REVISION_DESCRIPTION --autogenerate
```

Note: This generates a prepopulated template with the changes needed to match the database state with the models.

8.5.2 Legacy nova-network to OpenStack Networking (neutron)

Two networking models exist in OpenStack. The first is called legacy networking (nova-network) and it is a sub-process embedded in the Compute project (nova). This model has some limitations, such as creating complex network topologies, extending its back-end implementation to vendor-specific technologies, and providing project-specific networking elements. These limitations are the main reasons the OpenStack Networking (neutron) model was created.

This section describes the process of migrating clouds based on the legacy networking model to the OpenStack Networking model. This process requires additional changes to both compute and networking to support the migration. This document describes the overall process and the features required in both Networking and Compute.

The current process as designed is a minimally viable migration with the goal of deprecating and then removing legacy networking. Both the Compute and Networking teams agree that a one-button migration process from legacy networking to OpenStack Networking (neutron) is not an essential requirement for the deprecation and removal of the legacy networking at a future date. This section includes a process and tools which are designed to solve a simple use case migration.

Users are encouraged to take these tools, test them, provide feedback, and then expand on the feature set to suit their own deployments; deployers that refrain from participating in this process intending to wait for a path that better suits their use case are likely to be disappointed.

Impact and limitations

The migration process from the legacy nova-network networking service to OpenStack Networking (neutron) has some limitations and impacts on the operational state of the cloud. It is critical to understand them in order to decide whether or not this process is acceptable for your cloud and all users.

Management impact

The Networking REST API is publicly read-only until after the migration is complete. During the migration, Networking REST API is read-write only to nova-api, and changes to Networking are only allowed via nova-api.

The Compute REST API is available throughout the entire process, although there is a brief period where it is made read-only during a database migration. The Networking REST API will need to expose (to nova-api) all details necessary for reconstructing the information previously held in the legacy networking database.

Compute needs a per-hypervisor has_transitioned boolean change in the data model to be used during the migration process. This flag is no longer required once the process is complete.

Operations impact

In order to support a wide range of deployment options, the migration process described here requires a rolling restart of hypervisors. The rate and timing of specific hypervisor restarts is under the control of the operator.

The migration may be paused, even for an extended period of time (for example, while testing or investigating issues) with some hypervisors on legacy networking and some on Networking, and Compute API remains fully functional. Individual hypervisors may be rolled back to legacy networking during this stage of the migration, although this requires an additional restart.

In order to support the widest range of deployer needs, the process described here is easy to automate but is not already automated. Deployers should expect to perform multiple manual steps or write some simple scripts in order to perform this migration.

Performance impact

During the migration, nova-network API calls will go through an additional internal conversion to Networking calls. This will have different and likely poorer performance characteristics compared with either the pre-migration or post-migration APIs.

Migration process overview

1. Start neutron-server in intended final config, except with REST API restricted to read-write only by nova-api.
2. Make the Compute REST API read-only.
3. Run a DB dump/restore tool that creates Networking data structures representing current legacy networking config.
4. Enable a nova-api proxy that recreates internal Compute objects from Networking information (via the Networking REST API).
5. Make Compute REST API read-write again. This means legacy networking DB is now unused, new changes are now stored in the Networking DB, and no rollback is possible from here without losing those new changes.

Note: At this moment the Networking DB is the source of truth, but nova-api is the only public read-write API.

Next, youll need to migrate each hypervisor. To do that, follow these steps:

1. Disable the hypervisor. This would be a good time to live migrate or evacuate the compute node, if supported.
2. Disable nova-compute.
3. Enable the Networking agent.
4. Set the has_transitioned flag in the Compute hypervisor database/config.
5. Reboot the hypervisor (or run smart live transition tool if available).

6. Re-enable the hypervisor.

At this point, all compute nodes have been migrated, but they are still using the nova-api API and Compute gateways. Finally, enable OpenStack Networking by following these steps:

1. Bring up the Networking (l3) nodes. The new routers will have identical MAC+IPs as old Compute gateways so some sort of immediate cutover is possible, except for stateful connections issues such as NAT.

2. Make the Networking API read-write and disable legacy networking.

Migration Completed!

8.5.3 Add VRRP to an existing router

This section describes the process of migrating from a classic router to an L3 HA router, which is available starting from the Mitaka release.

Similar to the classic scenario, all network traffic on a project network that requires routing actively traverses only one network node regardless of the quantity of network nodes providing HA for the router. Therefore, this high-availability implementation primarily addresses failure situations instead of bandwidth constraints that limit performance. However, it supports random distribution of routers on different network nodes to reduce the chances of bandwidth constraints and to improve scaling.

This section references parts of *Linux bridge: High availability using VRRP* and *Open vSwitch: High availability using VRRP*. For details regarding needed infrastructure and configuration to allow actual L3 HA deployment, read the relevant guide before continuing with the migration process.

Migration

The migration process is quite simple, it involves turning down the router by setting the routers `admin_state_up` attribute to `False`, upgrading the router to L3 HA and then setting the routers `admin_state_up` attribute back to `True`.

> **Warning:** Once starting the migration, south-north connections (instances to internet) will be severed. New connections will be able to start only when the migration is complete.

Here is the router we have used in our demonstration:

```
$ openstack router show router1
+-------------------------+-------------------------------------------+
| Field                   | Value                                     |
+-------------------------+-------------------------------------------+
| admin_state_up          | UP                                        |
| distributed             | False                                     |
| external_gateway_info   |                                           |
| ha                      | False                                     |
| id                      | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a      |
| name                    | router1                                   |
| project_id              | bb8b84ab75be4e19bd0dfe02f6c3f5c1          |
| routes                  |                                           |
| status                  | ACTIVE                                    |
```

(continues on next page)

(continued from previous page)

```
| tags                    | []                                       |
+-------------------------+------------------------------------------+
```

1. Source the administrative project credentials.
2. Set the admin_state_up to `False`. This will severe south-north connections until admin_state_up is set to `True` again.

```
$ openstack router set router1 --disable
```

3. Set the `ha` attribute of the router to `True`.

```
$ openstack router set router1 --ha
```

4. Set the admin_state_up to `True`. After this, south-north connections can start.

```
$ openstack router set router1 --enable
```

5. Make sure that the routers `ha` attribute has changed to `True`.

```
$ openstack router show router1
+-------------------------+------------------------------------------
↪+
| Field                   | Value                                    ␣
↪|
+-------------------------+------------------------------------------
↪+
| admin_state_up          | UP                                       ␣
↪|
| distributed             | False                                    ␣
↪|
| external_gateway_info   |                                          ␣
↪|
| ha                      | True                                     ␣
↪|
| id                      | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a     ␣
↪|
| name                    | router1                                  ␣
↪|
| project_id              | bb8b84ab75be4e19bd0dfe02f6c3f5c1         ␣
↪|
| routes                  |                                          ␣
↪|
| status                  | ACTIVE                                   ␣
↪|
| tags                    | []                                       ␣
↪|
+-------------------------+------------------------------------------
↪+
```

L3 HA to Legacy

To return to classic mode, turn down the router again, turning off L3 HA and starting the router again.

> **Warning:** Once starting the migration, south-north connections (instances to internet) will be severed. New connections will be able to start only when the migration is complete.

Here is the router we have used in our demonstration:

```
$ openstack router show router1
+-------------------------+-------------------------------------------+
| Field                   | Value                                     |
+-------------------------+-------------------------------------------+
| admin_state_up          | DOWN                                      |
| distributed             | False                                     |
| external_gateway_info   |                                           |
| ha                      | True                                      |
| id                      | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a      |
| name                    | router1                                   |
| project_id              | bb8b84ab75be4e19bd0dfe02f6c3f5c1          |
| routes                  |                                           |
| status                  | ACTIVE                                    |
| tags                    | []                                        |
+-------------------------+-------------------------------------------+
```

1. Source the administrative project credentials.
2. Set the admin_state_up to `False`. This will severe south-north connections until admin_state_up is set to `True` again.

   ```
   $ openstack router set router1 --disable
   ```

3. Set the `ha` attribute of the router to `True`.

   ```
   $ openstack router set router1 --no-ha
   ```

4. Set the admin_state_up to `True`. After this, south-north connections can start.

   ```
   $ openstack router set router1 --enable
   ```

5. Make sure that the routers `ha` attribute has changed to `False`.

   ```
   $ openstack router show router1
   +-------------------------+-------------------------------------------
   ↪+
   | Field                   | Value                                     ␣
   ↪|
   +-------------------------+-------------------------------------------
   ↪+
   | admin_state_up          | UP                                        ␣
   ↪|
   | distributed             | False                                     ␣
   ↪|
   | external_gateway_info   |                                           ␣
   ↪|
   ```

(continues on next page)

(continued from previous page)

```
| ha                     | False                                    ␣
↪|
| id                     | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a     ␣
↪|
| name                   | router1                                  ␣
↪|
| project_id             | bb8b84ab75be4e19bd0dfe02f6c3f5c1         ␣
↪|
| routes                 |                                          ␣
↪|
| status                 | ACTIVE                                   ␣
↪|
| tags                   | []                                       ␣
↪|
+------------------------+------------------------------------------
↪+
```

8.6 Miscellaneous

8.6.1 Firewall-as-a-Service (FWaaS) v2 scenario

Enable FWaaS v2

1. Enable the FWaaS plug-in in the `/etc/neutron/neutron.conf` file:

```
service_plugins = firewall_v2

[service_providers]
# ...
service_provider = FIREWALL_V2:fwaas_db:neutron_fwaas.services.
↪firewall.service_drivers.agents.agents.FirewallAgentDriver:default

[fwaas]
agent_version = v2
driver = neutron_fwaas.services.firewall.service_drivers.agents.
↪drivers.linux.iptables_fwaas_v2.IptablesFwaasDriver
enabled = True
```

Note: On Ubuntu and Centos, modify the `[fwaas]` section in the `/etc/neutron/fwaas_driver.ini` file instead of `/etc/neutron/neutron.conf`.

2. Configure the FWaaS plugin for the L3 agent.

   In the `AGENT` section of `l3_agent.ini`, make sure the FWaaS v2 extension is loaded:

```
[AGENT]
extensions = fwaas_v2
```

3. Configure the ML2 plugin agent extension.

   Add the following statements to `ml2_conf.ini`, this file is usually located at `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[agent]
extensions = fwaas_v2

[fwaas]
firewall_l2_driver = noop
```

4. Create the required tables in the database:

```
# neutron-db-manage --subproject neutron-fwaas upgrade head
```

5. Restart the `neutron-l3-agent`, `neutron-openvswitch-agent` and `neutron-server` services to apply the settings.

Note: Firewall v2 is not supported by horizon yet.

Configure Firewall-as-a-Service v2

Create the firewall rules and create a policy that contains them. Then, create a firewall that applies the policy.

1. Create a firewall rule:

```
$ openstack firewall group rule create --protocol {tcp,udp,icmp,any} \
  --source-ip-address SOURCE_IP_ADDRESS \
  --destination-ip-address DESTINATION_IP_ADDRESS \
  --source-port SOURCE_PORT_RANGE --destination-port DEST_PORT_RANGE \
  --action {allow,deny,reject}
```

The Networking client requires a protocol value. If the rule is protocol agnostic, you can use the `any` value.

Note: When the source or destination IP address are not of the same IP version (for example, IPv6), the command returns an error.

2. Create a firewall policy:

```
$ openstack firewall group policy create --firewall-rule \
  "FIREWALL_RULE_IDS_OR_NAMES" myfirewallpolicy
```

Separate firewall rule IDs or names with spaces. The order in which you specify the rules is important.

You can create a firewall policy without any rules and add rules later, as follows:

- To add multiple rules, use the update operation.
- To add a single rule, use the insert-rule operation.

For more details, see Networking command-line client in the OpenStack Command-Line Interface Reference.

Note: FWaaS always adds a default `deny all` rule at the lowest precedence of each policy. Consequently, a firewall policy with no rules blocks all traffic by default.

3. Create a firewall group:

```
$ openstack firewall group create --ingress-firewall-policy \
  "FIREWALL_POLICY_IDS_OR_NAMES" --egress-firewall-policy \
  "FIREWALL_POLICY_IDS_OR_NAMES" --port "PORT_IDS_OR_NAMES"
```

Separate firewall policy IDs or names with spaces. The direction in which you specify the policies is important.

Note: The firewall remains in PENDING_CREATE state until you create a Networking router and attach an interface to it.

8.6.2 Disable libvirt networking

Most OpenStack deployments use the libvirt toolkit for interacting with the hypervisor. Specifically, OpenStack Compute uses libvirt for tasks such as booting and terminating virtual machine instances. When OpenStack Compute boots a new instance, libvirt provides OpenStack with the VIF associated with the instance, and OpenStack Compute plugs the VIF into a virtual device provided by OpenStack Network. The libvirt toolkit itself does not provide any networking functionality in OpenStack deployments.

However, libvirt is capable of providing networking services to the virtual machines that it manages. In particular, libvirt can be configured to provide networking functionality akin to a simplified, single-node version of OpenStack. Users can use libvirt to create layer 2 networks that are similar to OpenStack Networkings networks, confined to a single node.

libvirt network implementation

By default, libvirts networking functionality is enabled, and libvirt creates a network when the system boots. To implement this network, libvirt leverages some of the same technologies that OpenStack Network does. In particular, libvirt uses:

- Linux bridging for implementing a layer 2 network
- dnsmasq for providing IP addresses to virtual machines using DHCP
- iptables to implement SNAT so instances can connect out to the public internet, and to ensure that virtual machines are permitted to communicate with dnsmasq using DHCP

By default, libvirt creates a network named *default*. The details of this network may vary by distribution; on Ubuntu this network involves:

- a Linux bridge named `virbr0` with an IP address of `192.0.2.1/24`
- a dnsmasq process that listens on the `virbr0` interface and hands out IP addresses in the range `192.0.2.2-192.0.2.254`
- a set of iptables rules

When libvirt boots a virtual machine, it places the machines VIF in the bridge `virbr0` unless explicitly told not to.

On Ubuntu, the iptables ruleset that libvirt creates includes the following rules:

```
*nat
-A POSTROUTING -s 192.0.2.0/24 -d 224.0.0.0/24 -j RETURN
-A POSTROUTING -s 192.0.2.0/24 -d 255.255.255.255/32 -j RETURN
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -p tcp -j MASQUERADE --to-
↪ports 1024-65535
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -p udp -j MASQUERADE --to-
↪ports 1024-65535
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -j MASQUERADE
*mangle
-A POSTROUTING -o virbr0 -p udp -m udp --dport 68 -j CHECKSUM --checksum-
↪fill
*filter
-A INPUT -i virbr0 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i virbr0 -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -i virbr0 -p udp -m udp --dport 67 -j ACCEPT
-A INPUT -i virbr0 -p tcp -m tcp --dport 67 -j ACCEPT
-A FORWARD -d 192.0.2.0/24 -o virbr0 -m conntrack --ctstate RELATED,
↪ESTABLISHED -j ACCEPT
-A FORWARD -s 192.0.2.0/24 -i virbr0 -j ACCEPT
-A FORWARD -i virbr0 -o virbr0 -j ACCEPT
-A FORWARD -o virbr0 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -i virbr0 -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -o virbr0 -p udp -m udp --dport 68 -j ACCEPT
```

The following shows the dnsmasq process that libvirt manages as it appears in the output of **ps**:

```
2881 ?        S      0:00 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/
↪dnsmasq/default.conf
```

How to disable libvirt networks

Although OpenStack does not make use of libvirts networking, this networking will not interfere with OpenStacks behavior, and can be safely left enabled. However, libvirts networking can be a nuisance when debugging OpenStack networking issues. Because libvirt creates an additional bridge, dnsmasq process, and iptables ruleset, these may distract an operator engaged in network troubleshooting. Unless you need to start up virtual machines using libvirt directly, you can safely disable libvirts network.

To view the defined libvirt networks and their state:

```
# virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
```

To deactivate the libvirt network named `default`:

```
# virsh net-destroy default
```

Deactivating the network will remove the `virbr0` bridge, terminate the dnsmasq process, and remove the iptables rules.

To prevent the network from automatically starting on boot:

```
# virsh net-autostart --network default --disable
```

To activate the network after it has been deactivated:

```
# virsh net-start default
```

8.6.3 neutron-linuxbridge-cleanup utility

Description

Automated removal of empty bridges has been disabled to fix a race condition between the Compute (nova) and Networking (neutron) services. Previously, it was possible for a bridge to be deleted during the time when the only instance using it was rebooted.

Usage

Use this script to remove empty bridges on compute nodes by running the following command:

```
$ neutron-linuxbridge-cleanup
```

Important: Do not use this tool when creating or migrating an instance as it throws an error when the bridge does not exist.

Note: Using this script can still trigger the original race condition. Only run this script if you have evacuated all instances off a compute node and you want to clean up the bridges. In addition to evacuating all instances, you should fence off the compute node where you are going to run this script so new instances do not get scheduled on it.

8.6.4 Virtual Private Network-as-a-Service (VPNaaS) scenario

Enabling VPNaaS

This section describes the setting for the reference implementation. Vendor plugins or drivers can have different setup procedure and perhaps they provide their version of manuals.

1. Enable the VPNaaS plug-in in the `/etc/neutron/neutron.conf` file by appending `vpnaas` to `service_plugins` in `[DEFAULT]`:

   ```
   [DEFAULT]
   # ...
   service_plugins = vpnaas
   ```

   Note: `vpnaas` is just example of reference implementation. It depends on a plugin that you are going to use. Consider to set suitable plugin for your own deployment.

2. Configure the VPNaaS service provider by creating the `/etc/neutron/neutron_vpnaas.conf` file as follows, `strongswan` used in Ubuntu distribution:

```
[service_providers]
service_provider = VPN:strongswan:neutron_vpnaas.services.vpn.service_
↪drivers.ipsec.IPsecVPNDriver:default
```

Note: There are several kinds of service drivers. Depending upon the Linux distribution, you may need to override this value. Select `libreswan` for RHEL/CentOS, the config will like this: `service_provider = VPN:openswan:neutron_vpnaas.services.vpn.service_drivers.ipsec.IPsecVPNDriver:default`. Consider to use the appropriate one for your deployment.

3. Configure the VPNaaS plugin for the L3 agent by adding to `/etc/neutron/l3_agent.ini` the following section, `StrongSwanDriver` used in Ubuntu distribution:

```
[AGENT]
extensions = vpnaas

[vpnagent]
vpn_device_driver = neutron_vpnaas.services.vpn.device_drivers.
↪strongswan_ipsec.StrongSwanDriver
```

Note: There are several kinds of device drivers. Depending upon the Linux distribution, you may need to override this value. Select `LibreSwanDriver` for RHEL/CentOS, the config will like this: `vpn_device_driver = neutron_vpnaas.services.vpn.device_drivers.libreswan_ipsec.LibreSwanDriver`. Consider to use the appropriate drivers for your deployment.

4. Create the required tables in the database:

```
# neutron-db-manage --subproject neutron-vpnaas upgrade head
```

Note: In order to run the above command, you need to have neutron-vpnaas package installed on controller node.

5. Restart the `neutron-server` in controller node to apply the settings.

6. Restart the `neutron-l3-agent` in network node to apply the settings.

Using VPNaaS with endpoint group (recommended)

IPsec site-to-site connections will support multiple local subnets, in addition to the current multiple peer CIDRs. The multiple local subnet feature is triggered by not specifying a local subnet, when creating a VPN service. Backwards compatibility is maintained with single local subnets, by providing the subnet in the VPN service creation.

To support multiple local subnets, a new capability called End Point Groups has been added. Each endpoint group will define one or more endpoints of a specific type, and can be used to specify both local and peer endpoints for IPsec connections. The endpoint groups separate the what gets connected from the how to connect for a VPN service, and can be used for different flavors of VPN, in the future.

Refer Multiple Local Subnets for more detail.

Create the IKE policy, IPsec policy, VPN service, local endpoint group and peer endpoint group. Then, create an IPsec site connection that applies the above policies and service.

1. Create an IKE policy:

```
$ openstack vpn ike policy create ikepolicy
  +-------------------------------+----------------------------------
↪-----+
  | Field                         | Value                            ␣
↪     |
  +-------------------------------+----------------------------------
↪-----+
  | Authentication Algorithm      | sha1                             ␣
↪     |
  | Description                   |                                  ␣
↪     |
  | Encryption Algorithm          | aes-128                          ␣
↪     |
  | ID                            | 735f4691-3670-43b2-b389-
↪f4d81a60ed56    |
  | IKE Version                   | v1                               ␣
↪     |
  | Lifetime                      | {u'units': u'seconds', u'value':␣
↪3600} |
  | Name                          | ikepolicy                        ␣
↪     |
  | Perfect Forward Secrecy (PFS) | group5                           ␣
↪     |
  | Phase1 Negotiation Mode       | main                             ␣
↪     |
  | Project                       | 095247cb2e22455b9850c6efff407584 ␣
↪     |
  | project_id                    | 095247cb2e22455b9850c6efff407584 ␣
↪     |
  +-------------------------------+----------------------------------
↪-----+
```

2. Create an IPsec policy:

```
$ openstack vpn ipsec policy create ipsecpolicy
  +-------------------------------+----------------------------------
↪-----+
  | Field                         | Value                            ␣
↪     |
```

(continues on next page)

(continued from previous page)

```
+-------------------------------+-----------------------------------
↪-----+
| Authentication Algorithm      | sha1                              ␣
↪     |
| Description                   |                                   ␣
↪     |
| Encapsulation Mode            | tunnel                            ␣
↪     |
| Encryption Algorithm          | aes-128                           ␣
↪     |
| ID                            | 4f3f46fc-f2dc-4811-a642-
↪9601ebae310f   |
| Lifetime                      | {u'units': u'seconds', u'value':␣
↪3600} |
| Name                          | ipsecpolicy                       ␣
↪     |
| Perfect Forward Secrecy (PFS) | group5                            ␣
↪     |
| Project                       | 095247cb2e22455b9850c6efff407584 ␣
↪     |
| Transform Protocol            | esp                               ␣
↪     |
| project_id                    | 095247cb2e22455b9850c6efff407584 ␣
↪     |
+-------------------------------+-----------------------------------
↪-----+
```

3. Create a VPN service:

```
$ openstack vpn service create vpn \
  --router 9ff3f20c-314f-4dac-9392-defdbbb36a66
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| Description    |                                      |
| Flavor         | None                                 |
| ID             | 9f499f9f-f672-4ceb-be3c-d5ff3858c680 |
| Name           | vpn                                  |
| Project        | 095247cb2e22455b9850c6efff407584     |
| Router         | 9ff3f20c-314f-4dac-9392-defdbbb36a66 |
| State          | True                                 |
| Status         | PENDING_CREATE                       |
| Subnet         | None                                 |
| external_v4_ip | 192.168.20.7                         |
| external_v6_ip | 2001:db8::7                          |
| project_id     | 095247cb2e22455b9850c6efff407584     |
+----------------+--------------------------------------+
```

Note: Please do not specify `--subnet` option in this case.

The Networking openstackclient requires a router (Name or ID) and name.

4. Create local endpoint group:

```
$ openstack vpn endpoint group create ep_subnet \
  --type subnet \
  --value 1f888dd0-2066-42a1-83d7-56518895e47d
  +-------------+------------------------------------------+
  | Field       | Value                                    |
  +-------------+------------------------------------------+
  | Description |                                          |
  | Endpoints   | [u'1f888dd0-2066-42a1-83d7-56518895e47d'] |
  | ID          | 667296d0-67ca-4d0f-b676-7650cf96e7b1     |
  | Name        | ep_subnet                                |
  | Project     | 095247cb2e22455b9850c6efff407584         |
  | Type        | subnet                                   |
  | project_id  | 095247cb2e22455b9850c6efff407584         |
  +-------------+------------------------------------------+
```

Note: The type of a local endpoint group must be `subnet`.

5. Create peer endpoint group:

```
$ openstack vpn endpoint group create ep_cidr \
  --type cidr \
  --value 192.168.1.0/24
  +-------------+--------------------------------------+
  | Field       | Value                                |
  +-------------+--------------------------------------+
  | Description |                                      |
  | Endpoints   | [u'192.168.1.0/24']                  |
  | ID          | 5c3d7f2a-4a2a-446b-9fcf-9a2557cfc641 |
  | Name        | ep_cidr                              |
  | Project     | 095247cb2e22455b9850c6efff407584     |
  | Type        | cidr                                 |
  | project_id  | 095247cb2e22455b9850c6efff407584     |
  +-------------+--------------------------------------+
```

Note: The type of a peer endpoint group must be `cidr`.

6. Create an ipsec site connection:

```
$ openstack vpn ipsec site connection create conn \
  --vpnservice vpn \
  --ikepolicy ikepolicy \
  --ipsecpolicy ipsecpolicy \
  --peer-address 192.168.20.9 \
  --peer-id 192.168.20.9 \
  --psk secret \
  --local-endpoint-group ep_subnet \
  --peer-endpoint-group ep_cidr
  +--------------------------+--------------------------------------
↪----------------+
  | Field                    | Value                                 ␣
↪                |
  +--------------------------+--------------------------------------
↪----------------+
```

(continues on next page)

(continued from previous page)

```
| Authentication Algorithm | psk                                    ␣
↪                |
| Description              |                                        ␣
↪                |
| ID                       | 07e400b7-9de3-4ea3-a9d0-90a185e5b00d   ␣
↪                |
| IKE Policy               | 735f4691-3670-43b2-b389-f4d81a60ed56   ␣
↪                |
| IPSec Policy             | 4f3f46fc-f2dc-4811-a642-9601ebae310f   ␣
↪                |
| Initiator                | bi-directional                         ␣
↪                |
| Local Endpoint Group ID  | 667296d0-67ca-4d0f-b676-7650cf96e7b1   ␣
↪                |
| Local ID                 |                                        ␣
↪                |
| MTU                      | 1500                                   ␣
↪                |
| Name                     | conn                                   ␣
↪                |
| Peer Address             | 192.168.20.9                           ␣
↪                |
| Peer CIDRs               |                                        ␣
↪                |
| Peer Endpoint Group ID   | 5c3d7f2a-4a2a-446b-9fcf-9a2557cfc641   ␣
↪                |
| Peer ID                  | 192.168.20.9                           ␣
↪                |
| Pre-shared Key           | secret                                 ␣
↪                |
| Project                  | 095247cb2e22455b9850c6efff407584       ␣
↪                |
| Route Mode               | static                                 ␣
↪                |
| State                    | True                                   ␣
↪                |
| Status                   | PENDING_CREATE                         ␣
↪                |
| VPN Service              | 9f499f9f-f672-4ceb-be3c-d5ff3858c680   ␣
↪                |
| dpd                      | {u'action': u'hold', u'interval': 30, u
↪'timeout': 120} |
| project_id               | 095247cb2e22455b9850c6efff407584       ␣
↪                |
+--------------------------+----------------------------------------
↪----------------+
```

Note: Please do not specify `--peer-cidr` option in this case. Peer CIDR(s) are provided by a peer endpoint group.

Configure VPNaaS without endpoint group (the legacy way)

Create the IKE policy, IPsec policy, VPN service. Then, create an ipsec site connection that applies the above policies and service.

1. Create an IKE policy:

```
$ openstack vpn ike policy create ikepolicy1
  +-------------------------------+----------------------------------
↪-----+
  | Field                         | Value                            ␣
↪     |
  +-------------------------------+----------------------------------
↪-----+
  | Authentication Algorithm      | sha1                             ␣
↪     |
  | Description                   |                                  ␣
↪     |
  | Encryption Algorithm          | aes-128                          ␣
↪     |
  | ID                            | 99e4345d-8674-4d73-acb4-
↪0e2524425e34    |
  | IKE Version                   | v1                               ␣
↪     |
  | Lifetime                      | {u'units': u'seconds', u'value':␣
↪3600} |
  | Name                          | ikepolicy1                       ␣
↪     |
  | Perfect Forward Secrecy (PFS) | group5                           ␣
↪     |
  | Phase1 Negotiation Mode       | main                             ␣
↪     |
  | Project                       | 095247cb2e22455b9850c6efff407584 ␣
↪     |
  | project_id                    | 095247cb2e22455b9850c6efff407584 ␣
↪     |
  +-------------------------------+----------------------------------
↪-----+
```

2. Create an IPsec policy:

```
$ openstack vpn ipsec policy create ipsecpolicy1
  +-------------------------------+----------------------------------
↪-----+
  | Field                         | Value                            ␣
↪     |
  +-------------------------------+----------------------------------
↪-----+
  | Authentication Algorithm      | sha1                             ␣
↪     |
  | Description                   |                                  ␣
↪     |
  | Encapsulation Mode            | tunnel                           ␣
↪     |
  | Encryption Algorithm          | aes-128                          ␣
↪     |
  | ID                            | e6f547af-4a1d-4c28-b40b-
↪b97cce746459    |
```

(continues on next page)

(continued from previous page)

```
| Lifetime                       | {u'units': u'seconds', u'value':␣
↪3600} |
| Name                           | ipsecpolicy1                     ␣
↪      |
| Perfect Forward Secrecy (PFS)  | group5                           ␣
↪      |
| Project                        | 095247cb2e22455b9850c6efff407584 ␣
↪      |
| Transform Protocol             | esp                              ␣
↪      |
| project_id                     | 095247cb2e22455b9850c6efff407584 ␣
↪      |
+--------------------------------+----------------------------------
↪-----+
```

3. Create a VPN service:

```
$ openstack vpn service create vpn \
  --router 66ca673a-cbbd-48b7-9fb6-bfa7ee3ef724 \
  --subnet cdfb411e-e818-466a-837c-7f96fc41a6d9
  +----------------+--------------------------------------+
  | Field          | Value                                |
  +----------------+--------------------------------------+
  | Description    |                                      |
  | Flavor         | None                                 |
  | ID             | 79ef6250-ddc3-428f-88c2-0ec8084f4e9a |
  | Name           | vpn                                  |
  | Project        | 095247cb2e22455b9850c6efff407584     |
  | Router         | 66ca673a-cbbd-48b7-9fb6-bfa7ee3ef724 |
  | State          | True                                 |
  | Status         | PENDING_CREATE                       |
  | Subnet         | cdfb411e-e818-466a-837c-7f96fc41a6d9 |
  | external_v4_ip | 192.168.20.2                         |
  | external_v6_ip | 2001:db8::d                          |
  | project_id     | 095247cb2e22455b9850c6efff407584     |
  +----------------+--------------------------------------+
```

Note: The `--subnet` option is required in this scenario.

4. Create an ipsec site connection:

```
$ openstack vpn ipsec site connection create conn \
  --vpnservice vpn \
  --ikepolicy ikepolicy1 \
  --ipsecpolicy ipsecpolicy1 \
  --peer-address 192.168.20.11 \
  --peer-id 192.168.20.11 \
  --peer-cidr 192.168.1.0/24 \
  --psk secret
  +--------------------------+--------------------------------------
↪----------------+
  | Field                    | Value                                ␣
↪                |
  +--------------------------+--------------------------------------
↪----------------+
```

(continues on next page)

(continued from previous page)

```
| Authentication Algorithm | psk                                                
↪                |
| Description              |                                                    
↪                |
| ID                       | 5b2935e6-b2f0-423a-8156-07ed48703d13               
↪                |
| IKE Policy               | 99e4345d-8674-4d73-acb4-0e2524425e34               
↪                |
| IPSec Policy             | e6f547af-4a1d-4c28-b40b-b97cce746459               
↪                |
| Initiator                | bi-directional                                     
↪                |
| Local Endpoint Group ID  | None                                               
↪                |
| Local ID                 |                                                    
↪                |
| MTU                      | 1500                                               
↪                |
| Name                     | conn                                               
↪                |
| Peer Address             | 192.168.20.11                                      
↪                |
| Peer CIDRs               | 192.168.1.0/24                                     
↪                |
| Peer Endpoint Group ID   | None                                               
↪                |
| Peer ID                  | 192.168.20.11                                      
↪                |
| Pre-shared Key           | secret                                             
↪                |
| Project                  | 095247cb2e22455b9850c6efff407584                   
↪                |
| Route Mode               | static                                             
↪                |
| State                    | True                                               
↪                |
| Status                   | PENDING_CREATE                                     
↪                |
| VPN Service              | 79ef6250-ddc3-428f-88c2-0ec8084f4e9a               
↪                |
| dpd                      | {u'action': u'hold', u'interval': 30, u
↪'timeout': 120} |
| project_id               | 095247cb2e22455b9850c6efff407584                   
↪                |
+--------------------------+----------------------------------------
↪----------------+
```

Note: Please do not specify `--local-endpoint-group` and `--peer-endpoint-group` options in this case.

8.7 OVN Driver Administration Guide

8.7.1 OVN information

The original OVN project announcement can be found here:

- https://networkheresy.com/2015/01/13/ovn-bringing-native-virtual-networking-to-ovs/

The OVN architecture is described here:

- http://www.ovn.org/support/dist-docs/ovn-architecture.7.html

Here are two tutorials that help with learning different aspects of OVN:

- https://blog.spinhirne.com/posts/an-introduction-to-ovn/a-primer-on-ovn/
- https://docs.ovn.org/en/stable/tutorials/ovn-sandbox.html

There is also an in depth tutorial on using OVN with OpenStack:

- https://docs.ovn.org/en/stable/tutorials/ovn-openstack.html

OVN DB schemas and other man pages:

- http://www.ovn.org/support/dist-docs/ovn-nb.5.html
- http://www.ovn.org/support/dist-docs/ovn-sb.5.html
- http://www.ovn.org/support/dist-docs/ovn-nbctl.8.html
- http://www.ovn.org/support/dist-docs/ovn-sbctl.8.html
- http://www.ovn.org/support/dist-docs/ovn-northd.8.html
- http://www.ovn.org/support/dist-docs/ovn-controller.8.html
- http://www.ovn.org/support/dist-docs/ovn-controller-vtep.8.html

or find a full list of OVS and OVN man pages here:

- http://docs.ovn.org/en/latest/ref/

The openvswitch web page includes a list of presentations, some of which are about OVN:

- http://openvswitch.org/support/

Here are some direct links to past OVN presentations:

- OVN talk at OpenStack Summit in Boston, Spring 2017
- OVN talk at OpenStack Summit in Barcelona, Fall 2016
- OVN talk at OpenStack Summit in Austin, Spring 2016
- OVN Project Update at the OpenStack Summit in Tokyo, Fall 2015 - Slides - Video
- OVN at OpenStack Summit in Vancouver, Sping 2015 - Slides - Video
- OVS Conference 2015

These blog resources may also help with testing and understanding OVN:

- http://networkop.co.uk/blog/2016/11/27/ovn-part1/
- http://networkop.co.uk/blog/2016/12/10/ovn-part2/

- https://blog.russellbryant.net/2016/12/19/comparing-openstack-neutron-ml2ovs-and-ovn-control-plane/
- https://blog.russellbryant.net/2016/11/11/ovn-logical-flows-and-ovn-trace/
- https://blog.russellbryant.net/2016/09/29/ovs-2-6-and-the-first-release-of-ovn/
- http://galsagie.github.io/2015/11/23/ovn-l3-deepdive/
- http://blog.russellbryant.net/2015/10/22/openstack-security-groups-using-ovn-acls/
- http://galsagie.github.io/sdn/openstack/ovs/2015/05/30/ovn-deep-dive/
- http://blog.russellbryant.net/2015/05/14/an-ez-bake-ovn-for-openstack/
- http://galsagie.github.io/sdn/openstack/ovs/2015/04/26/ovn-containers/
- http://blog.russellbryant.net/2015/04/21/ovn-and-openstack-status-2015-04-21/
- http://blog.russellbryant.net/2015/04/08/ovn-and-openstack-integration-development-update/
- http://dani.foroselectronica.es/category/openstack/ovn/

8.7.2 Features

Open Virtual Network (OVN) offers the following virtual network services:

- Layer-2 (switching)

  Native implementation. Replaces the conventional Open vSwitch (OVS) agent.

- Layer-3 (routing)

  Native implementation that supports distributed routing. Replaces the conventional Neutron L3 agent. This includes transparent L3HA :doc::*routing* support, based on BFD monitorization integrated in core OVN.

- DHCP

  Native distributed implementation. Replaces the conventional Neutron DHCP agent. Note that the native implementation does not yet support DNS features.

- DPDK

  OVN and the OVN mechanism driver may be used with OVS using either the Linux kernel datapath or the DPDK datapath.

- Trunk driver

  Uses OVNs functionality of parent port and port tagging to support trunk service plugin. One has to enable the trunk service plugin in neutron configuration files to use this feature.

- VLAN tenant networks

  The OVN driver does support VLAN tenant networks when used with OVN version 2.11 (or higher).

- DNS

  Native implementation. Since the version 2.8 OVN contains a built-in DNS implementation.

- Port Forwarding

  The OVN driver supports port forwarding as an extension of floating IPs. Enable the port_forwarding service plugin in neutron configuration files to use this feature.

The following Neutron API extensions are supported with OVN:

Extension Name	Extension Alias
Allowed Address Pairs	allowed-address-pairs
Auto Allocated Topology Services	auto-allocated-topology
Availability Zone	availability_zone
Default Subnetpools	default-subnetpools
Multi Provider Network	multi-provider
Network IP Availability	network-ip-availability
Neutron external network	external-net
Neutron Extra DHCP opts	extra_dhcp_opt
Neutron Extra Route	extraroute
Neutron L3 external gateway	ext-gw-mode
Neutron L3 Router	router
Network MTU	net-mtu
Port Binding	binding
Port Security	port-security
Provider Network	provider
Quality of Service	qos
Quota management support	quotas
RBAC Policies	rbac-policies
Resource revision numbers	standard-attr-revisions
security-group	security-group
standard-attr-description	standard-attr-description
Subnet Allocation	subnet_allocation
Tag support	standard-attr-tag
Time Stamp Fields	standard-attr-timestamp
Domain Name System (DNS)	dns_integration
Port Forwarding	port_forwarding

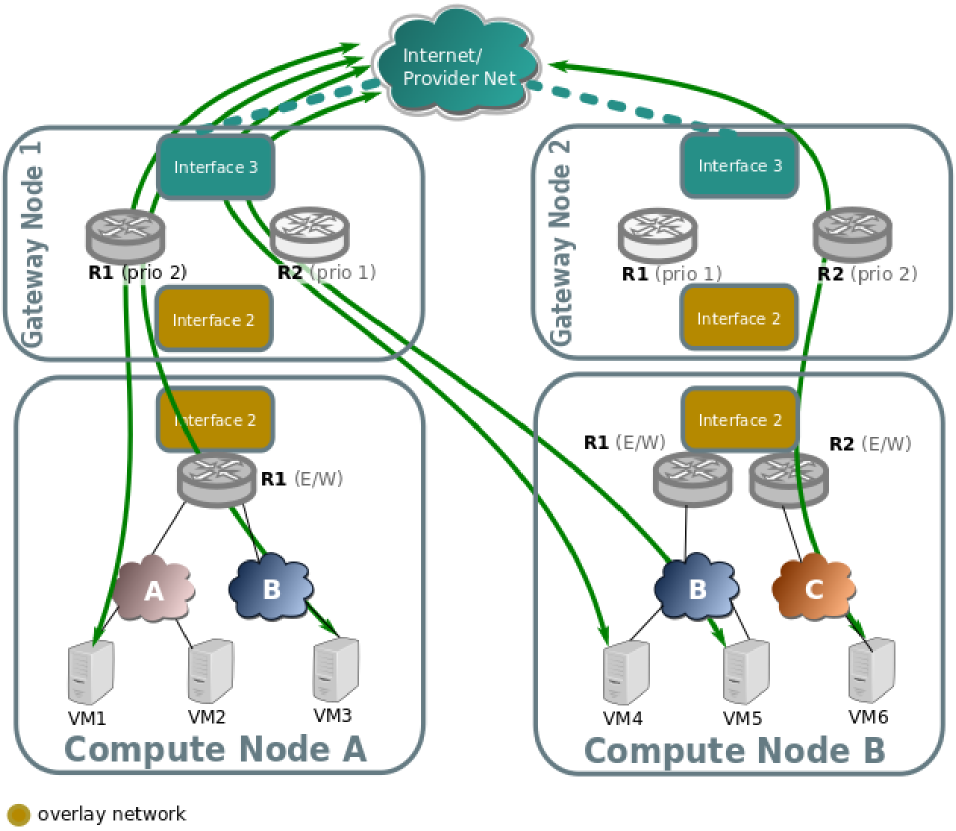
8.7.3 Routing

North/South

The different configurations are detailed in the *Reference architecture*

Non distributed FIP

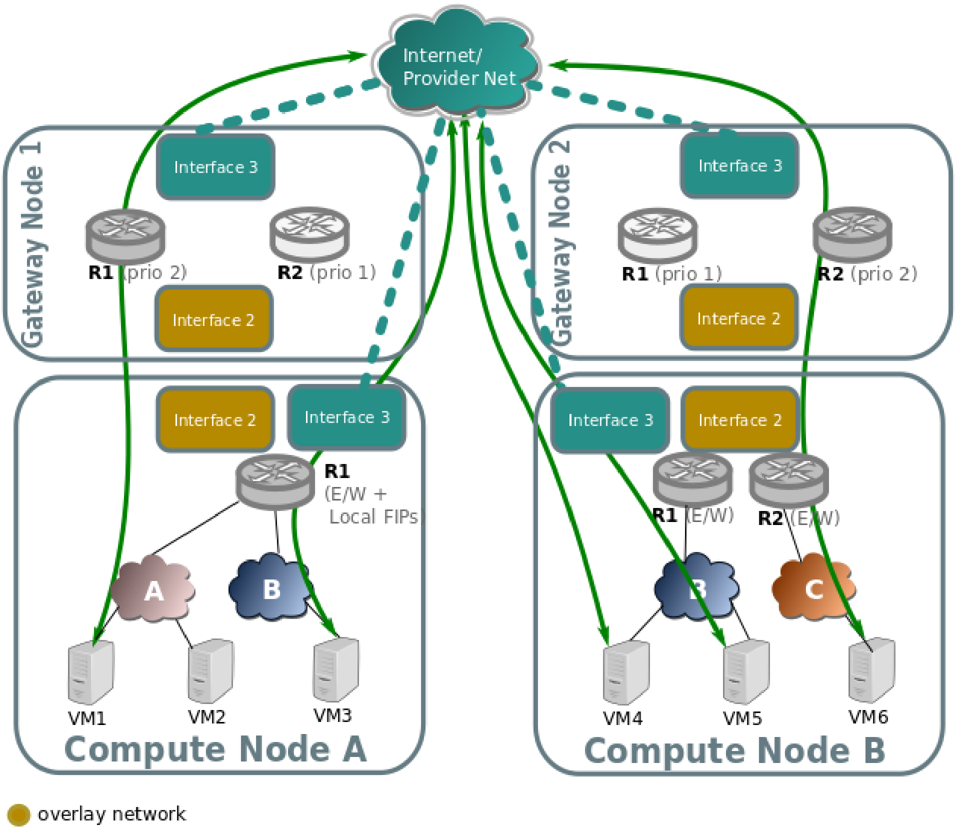
North/South traffic flows through the active chassis for each router for SNAT traffic, and also for FIPs.



Distributed Floating IP

In the following diagram we can see how VMs with no Floating IP (VM1, VM6) still communicate throught the gateway nodes using SNAT on the edge routers R1 and R2.

While VM3, VM4, and VM5 have an assigned floating IP, and its traffic flows directly through the local provider bridge/interface to the external network.



L3HA support

Ovn driver implements L3 high availability in a transparent way. You dont need to enable any config flags. As soon as you have more than one chassis capable of acting as an l3 gateway to the specific external network attached to the router it will schedule the router gateway port to multiple chassis, making use of the `gateway_chassis` column on OVNs `Logical_Router_Port` table.

In order to have external connectivity, either:

- Some gateway nodes have `ovn-cms-options` with the value `enable-chassis-as-gw` in Open_vSwitch tables external_ids column, or
- if no gateway node exists with the external ids column set with that value, then all nodes would be eligible to host gateway chassis.

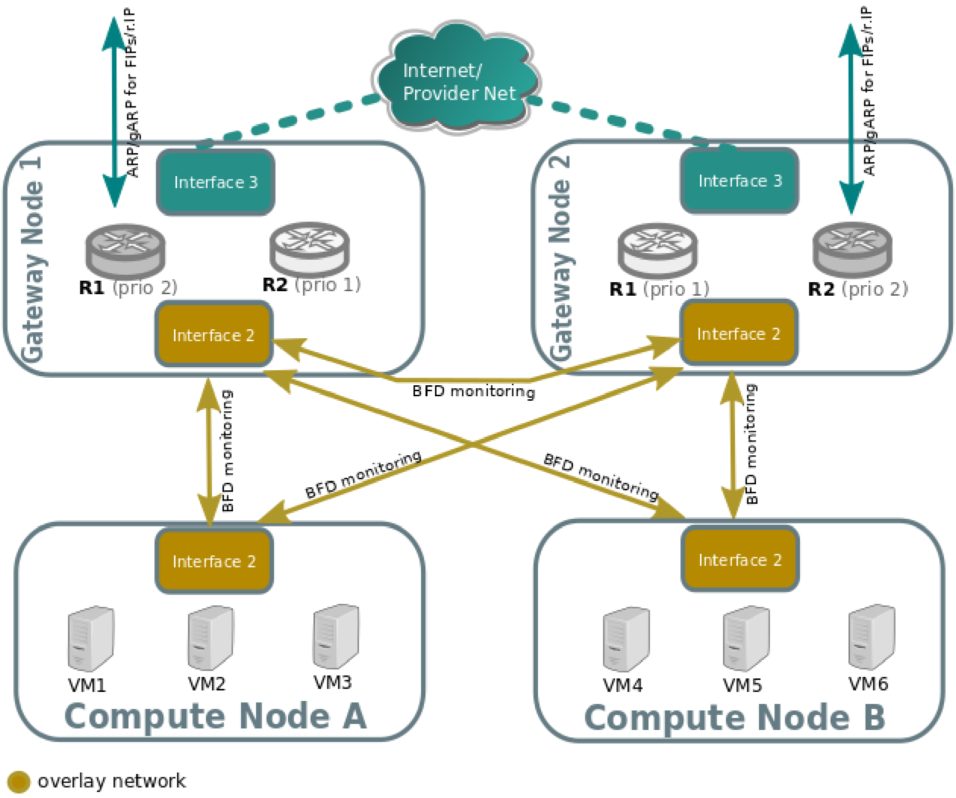
Example to how to enabled chassis to host gateways:

```
$ ovs-vsctl set open . external-ids:ovn-cms-options="enable-chassis-
↪as-gw"
```

At the low level, functionality is all implemented mostly by OpenFlow rules with bundle active_passive outputs. The ARP responder and router enablement/disablement is handled by ovn-controller. Gratuitous ARPs for FIPs and router external addresses are periodically sent by ovn-controller itself.

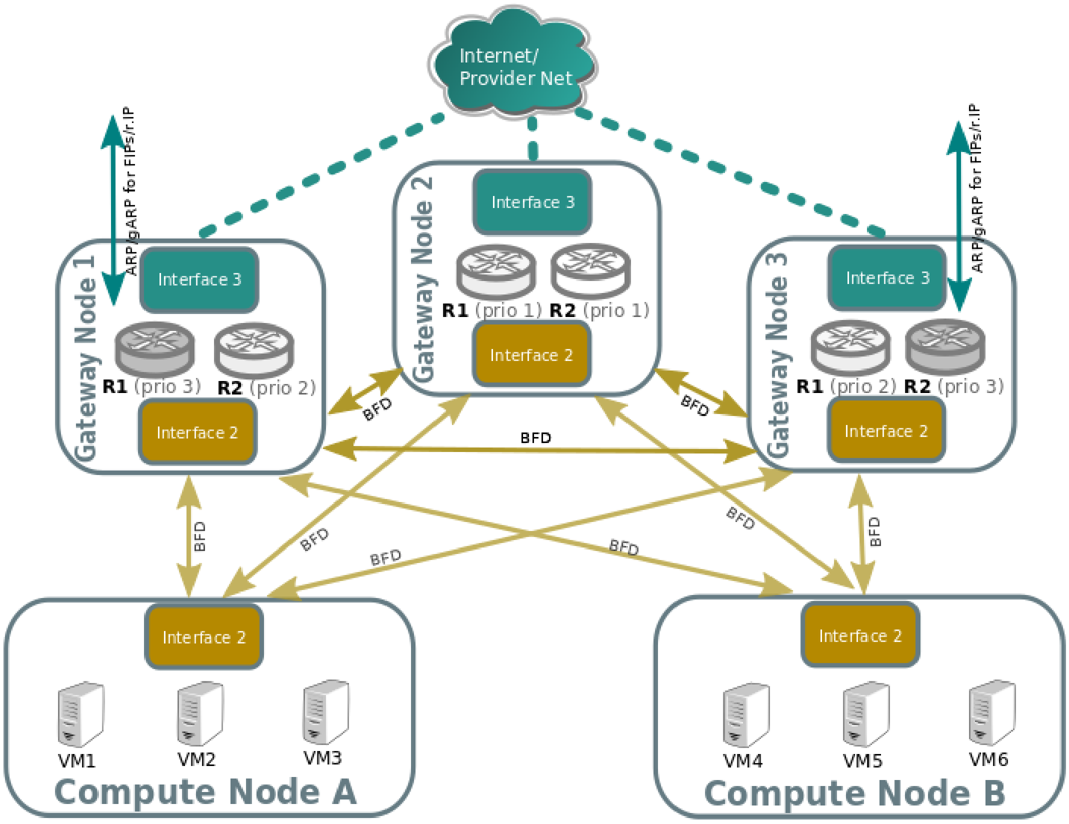
BFD monitoring

OVN monitors the availability of the chassis via the BFD protocol, which is encapsulated on top of the Geneve tunnels established from chassis to chassis.



Each chassis that is marked as a gateway chassis will monitor all the other gateway chassis in the deployment as well as compute node chassis, to let the gateways enable/disable routing of packets and ARP responses / announcements.

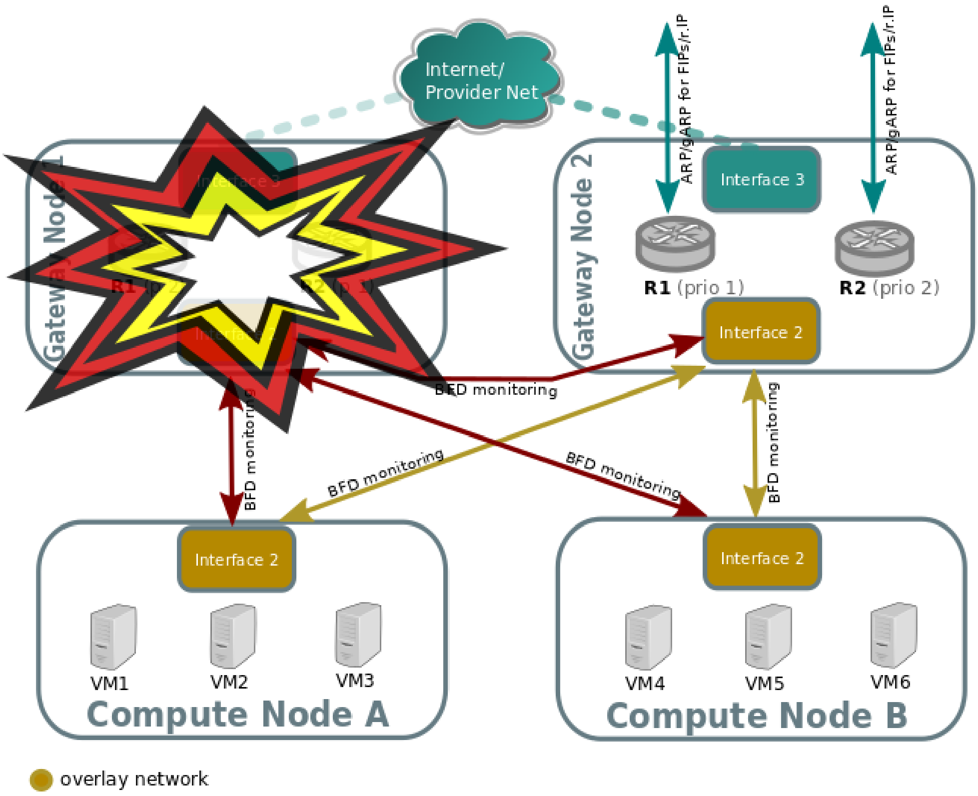
Each compute node chassis will monitor each gateway chassis via BFD to automatically steer external traffic (snat/dnat) through the active chassis for a given router.



The gateway nodes monitor each other in star topology. Compute nodes dont monitor each other because thats not necessary.

Failover (detected by BFD)

Look at the following example:



Compute nodes BFD monitoring of the gateway nodes will detect that tunnel endpoint going to gateway node 1 is down, so. So traffic output that needs to get into the external network through the router will be directed to the lower priority chassis for R1. R2 stays the same because Gateway Node 2 was already the highest priority chassis for R2.

Gateway node 2 will detect that tunnel endpoint to gateway node 1 is down, so it will become responsible for the external leg of R1, and its ovn-controller will populate flows for the external ARP responder, traffic forwarding (N/S) and periodic gratuitous ARPs.

Gateway node 2 will also bind the external port of the router (represented as a chassis-redirect port on the South Bound database).

If Gateway node 1 is still alive, failure over interface 2 will be detected because its not seeing any other nodes.

No mechanisms are still present to detect external network failure, so as good practice to detect network failure we recommend that all interfaces are handled over a single bonded interface with VLANs.

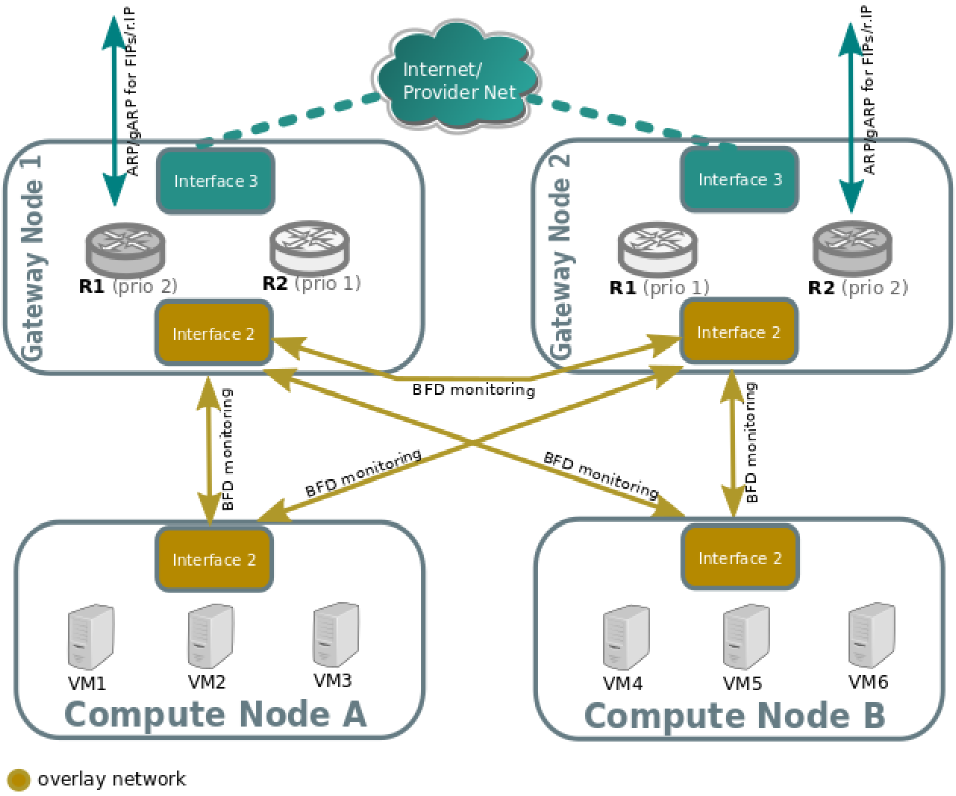
Supported failure modes are:

- gateway chassis becomes disconnected from network (tunneling interface)
- ovs-vswitchd is stopped (its responsible for BFD signaling)
- ovn-controller is stopped, as ovn-controller will remove himself as a registered chassis.

Note: As a side note, its also important to understand, that as for VRRP or CARP protocols, this detection mechanism only works for link failures, but not for routing failures.

Failback

L3HA behaviour is preemptive in OVN (at least for the time being) since that would balance back the routers to the original chassis, avoiding any of the gateway nodes becoming a bottleneck.

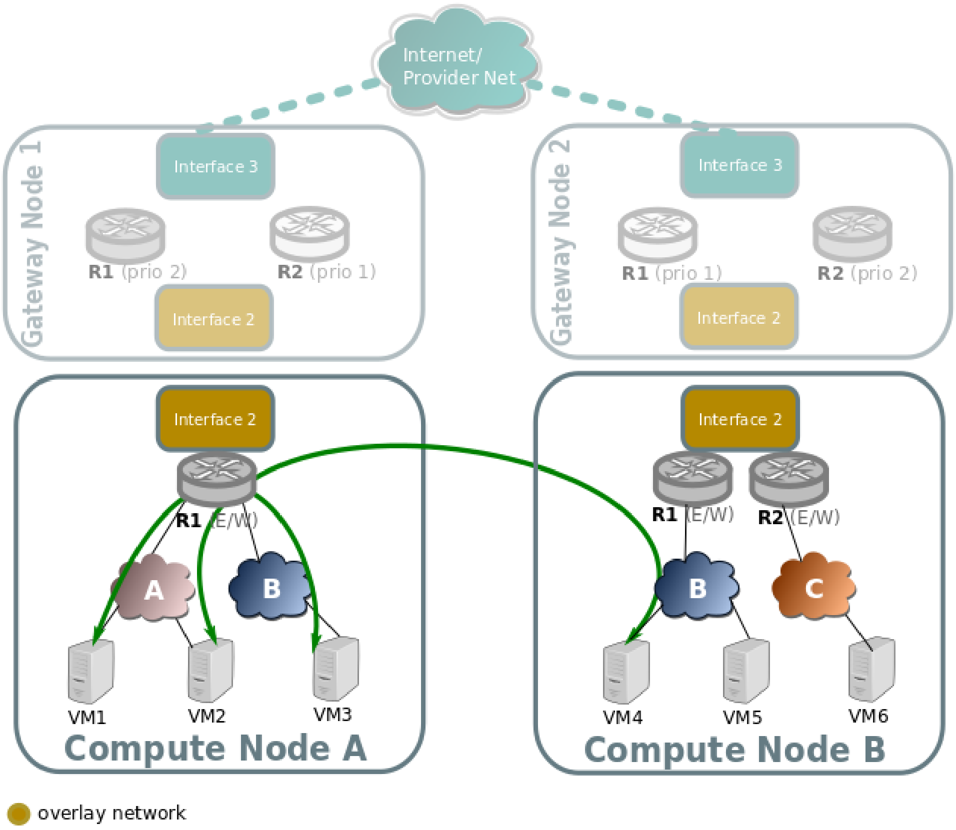


East/West

East/West traffic on ovn driver is completely distributed, that means that routing will happen internally on the compute nodes without the need to go through the gateway nodes.

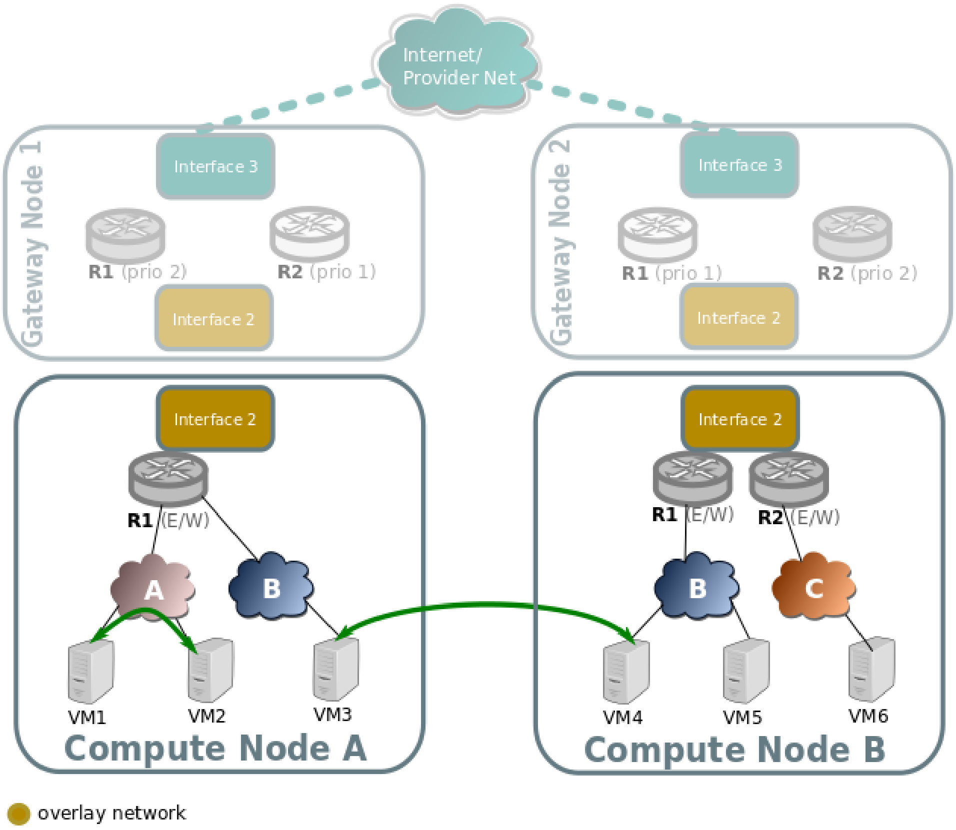
Traffic going through a virtual router, different subnets

Traffic going through a virtual router, and going from a virtual network/subnet to another will flow directly from compute to compute node encapsulated as usual, while all the routing operations like decreasing TTL or switching MAC addresses will be handled in OpenFlow at the source host of the packet.



Traffic across the same subnet

Traffic across a subnet will happen as described in the following diagram, although this kind of communication doesnt make use of routing at all (just encapsulation) its been included for completeness.



Traffic goes directly from instance to instance through br-int in the case of both instances living in the same host (VM1 and VM2), or via encapsulation when living on different hosts (VM3 and VM4).

8.7.4 IP Multicast: IGMP snooping configuration guide for OVN

How to enable it

In order to enable IGMP snooping with the OVN driver the following configuration needs to be set in the `/etc/neutron/neutron.conf` file of the controller nodes:

```
# OVN does reuse the OVS option, therefore the option group is [ovs]
[ovs]
igmp_snooping_enable = True
...
```

Upon restarting the Neutron service all existing networks (Logical_Switch, in OVN terms) will be updated in OVN to enable or disable IGMP snooping based on the `igmp_snooping_enable` configuration value.

Note: Currently the OVN driver does not configure IGMP querier in OVN so ovn-controller will not send IGMP group memberships IP querier to retrieve IGMP membership reports from active members.

OVN Database information

The `igmp_snooping_enable` configuration from Neutron is translated into the `mcast_snoop` option set in the `other_config` column from the `Logical_Switch` table in the OVN Northbound Database (`mcast_flood_unregistered` is always false):

```
$ ovn-nbctl list Logical_Switch
_uuid                : d6a2fbcd-aaa4-4b9e-8274-184238d66a15
other_config         : {mcast_flood_unregistered="false", mcast_snoop="true
↪"}
...
```

To find more information about the learnt IGMP groups by OVN use the command below (populated only when igmp_snooping_enable is True):

```
$ ovn-sbctl list IGMP_group
_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]
...
```

Note: Since IGMP querier is not yet supported in the OVN driver, restarting the ovn-controller service(s) will result in OVN unlearning the IGMP groups and broadcast all the multicast traffic. This behavior can impact when updating/upgrading the OVN services.

Extra information

When multicast IP traffic is sent to a multicast group address which is in the **224.0.0.X** range, the multicast traffic will be flooded, even when IGMP snooping is enabled. See the RFC 4541 session 2.1.2:

```
2) Packets with a destination IP (DIP) address in the 224.0.0.X range
   which are not IGMP must be forwarded on all ports.
```

The permutations from different configurations are:

- With IGMP snooping disabled: IP Multicast traffic flooded to all ports.
- With IGMP snooping enabled and multicast group address **not in** the 224.0.0.X range: IP Multicast traffic **is not** flooded.
- With IGMP snooping enabled and multicast group address **is in** the 224.0.0.X range: IP Multicast traffic **is** flooded.

8.7.5 OpenStack and OVN Tutorial

The OVN project documentation includes an in depth tutorial of using OVN with OpenStack.

OpenStack and OVN Tutorial

8.7.6 Reference architecture

The reference architecture defines the minimum environment necessary to deploy OpenStack with Open Virtual Network (OVN) integration for the Networking service in production with sufficient expectations of scale and performance. For evaluation purposes, you can deploy this environment using the *Installation Guide* or Vagrant. Any scaling or performance evaluations should use bare metal instead of virtual machines.

Layout

The reference architecture includes a minimum of four nodes.

The controller node contains the following components that provide enough functionality to launch basic instances:

- One network interface for management
- Identity service
- Image service
- Networking management with ML2 mechanism driver for OVN (control plane)
- Compute management (control plane)

The database node contains the following components:

- One network interface for management
- OVN northbound service (`ovn-northd`)
- Open vSwitch (OVS) database service (`ovsdb-server`) for the OVN northbound database (`ovnnb.db`)
- Open vSwitch (OVS) database service (`ovsdb-server`) for the OVN southbound database (`ovnsb.db`)

Note: For functional evaluation only, you can combine the controller and database nodes.

The two compute nodes contain the following components:

- Two or three network interfaces for management, overlay networks, and optionally provider networks
- Compute management (hypervisor)
- Hypervisor (KVM)
- OVN controller service (`ovn-controller`)
- OVS data plane service (`ovs-vswitchd`)

- OVS database service (`ovsdb-server`) with OVS local configuration (`conf.db`) database
- OVN metadata agent (`ovn-metadata-agent`)

The gateway nodes contain the following components:

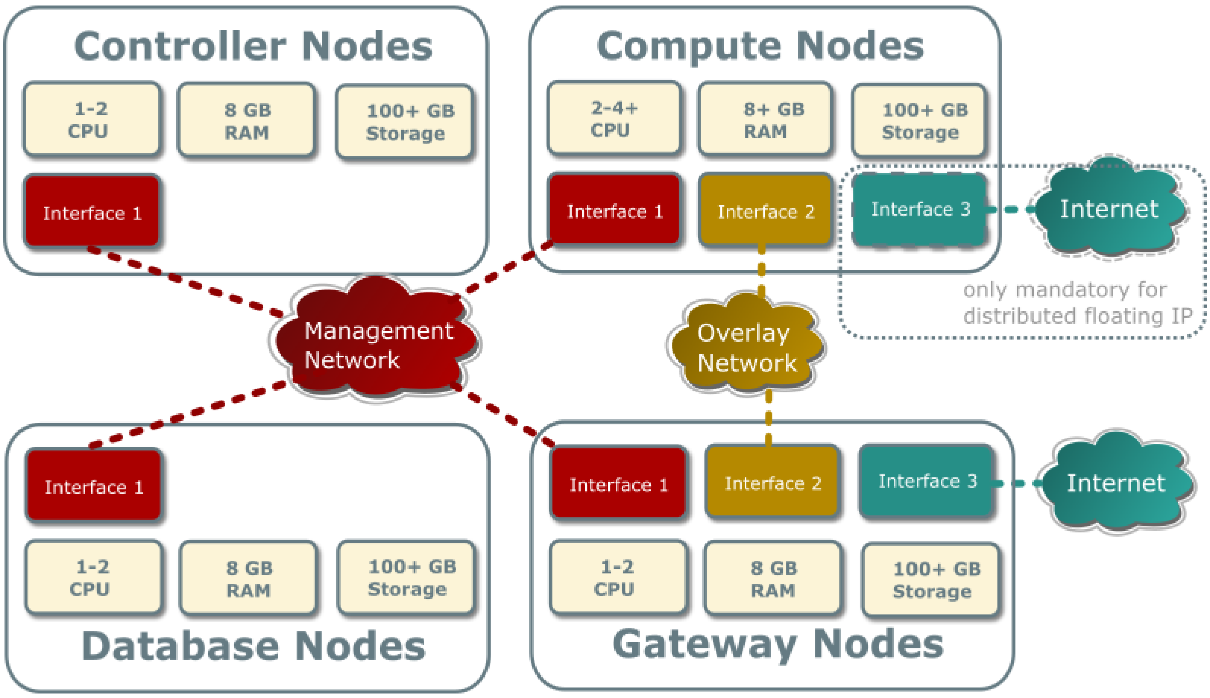
- Three network interfaces for management, overlay networks and provider networks.
- OVN controller service (`ovn-controller`)
- OVS data plane service (`ovs-vswitchd`)
- OVS database service (`ovsdb-server`) with OVS local configuration (`conf.db`) database

Note: Each OVN metadata agent provides metadata service locally on the compute nodes in a lightweight way. Each network being accessed by the instances of the compute node will have a corresponding metadata ovn-metadata-$net_uuid namespace, and inside an haproxy will funnel the requests to the ovn-metadata-agent over a unix socket.

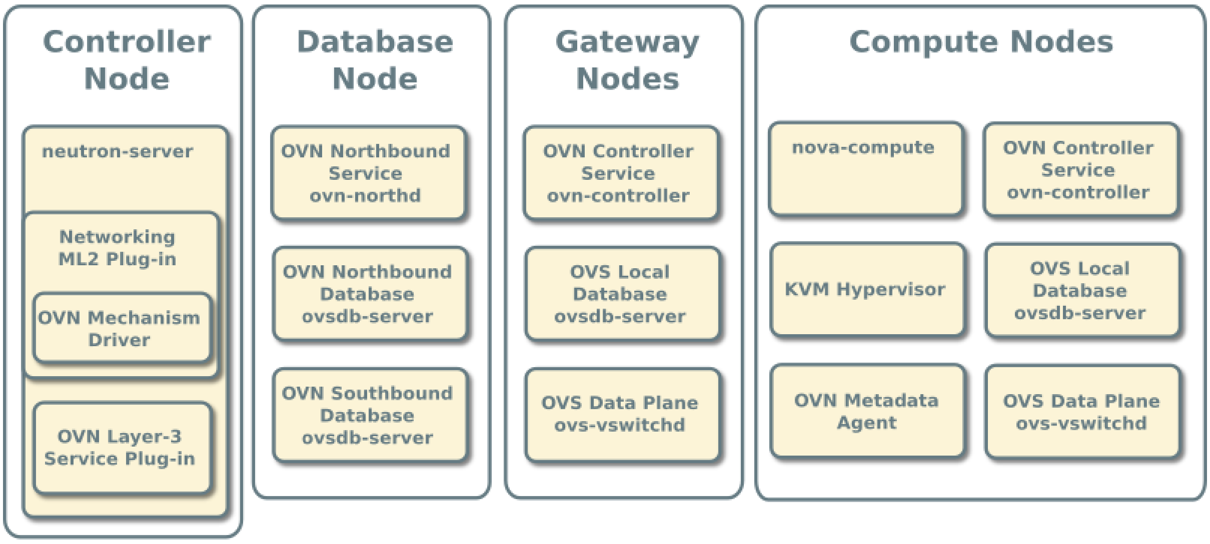
Such namespace can be very helpful for debug purposes to access the local instances on the compute node. If you login as root on such compute node you can execute:

ip netns ovn-metadata-$net_uuid exec ssh user@my.instance.ip.address

Hardware layout

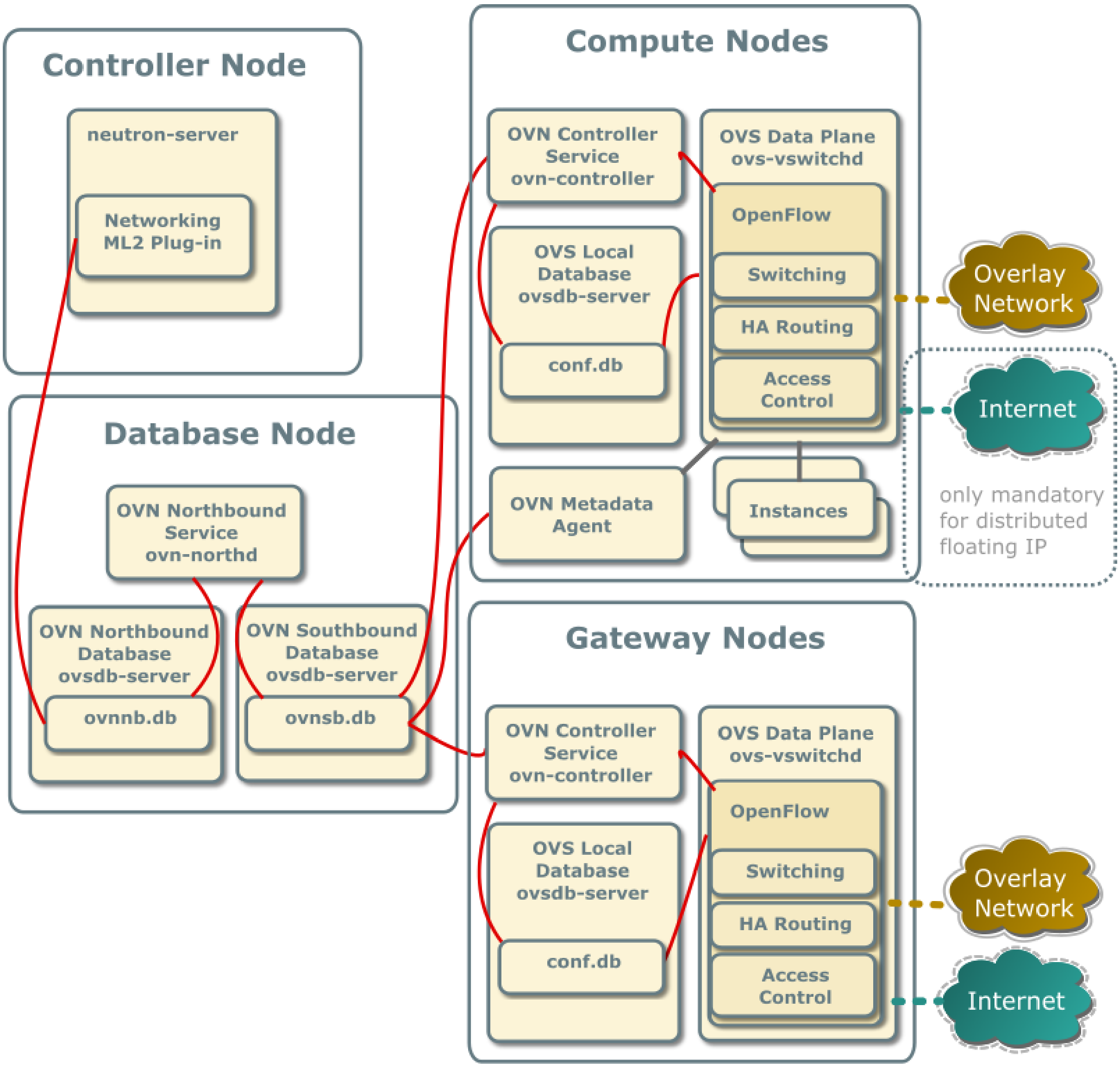


Service layout



Networking service with OVN integration

The reference architecture deploys the Networking service with OVN integration as described in the following scenarios:



With ovn driver, all the E/W traffic which traverses a virtual router is completely distributed, going from compute to compute node without passing through the gateway nodes.

N/S traffic that needs SNAT (without floating IPs) will always pass through the centralized gateway nodes, although, as soon as you have more than one gateway node ovn driver will make use of the HA capabilities of ovn.

Centralized Floating IPs

In this architecture, all the N/S router traffic (snat and floating IPs) goes through the gateway nodes.

The compute nodes dont need connectivity to the external network, although it could be provided if we wanted to have direct connectivity to such network from some instances.

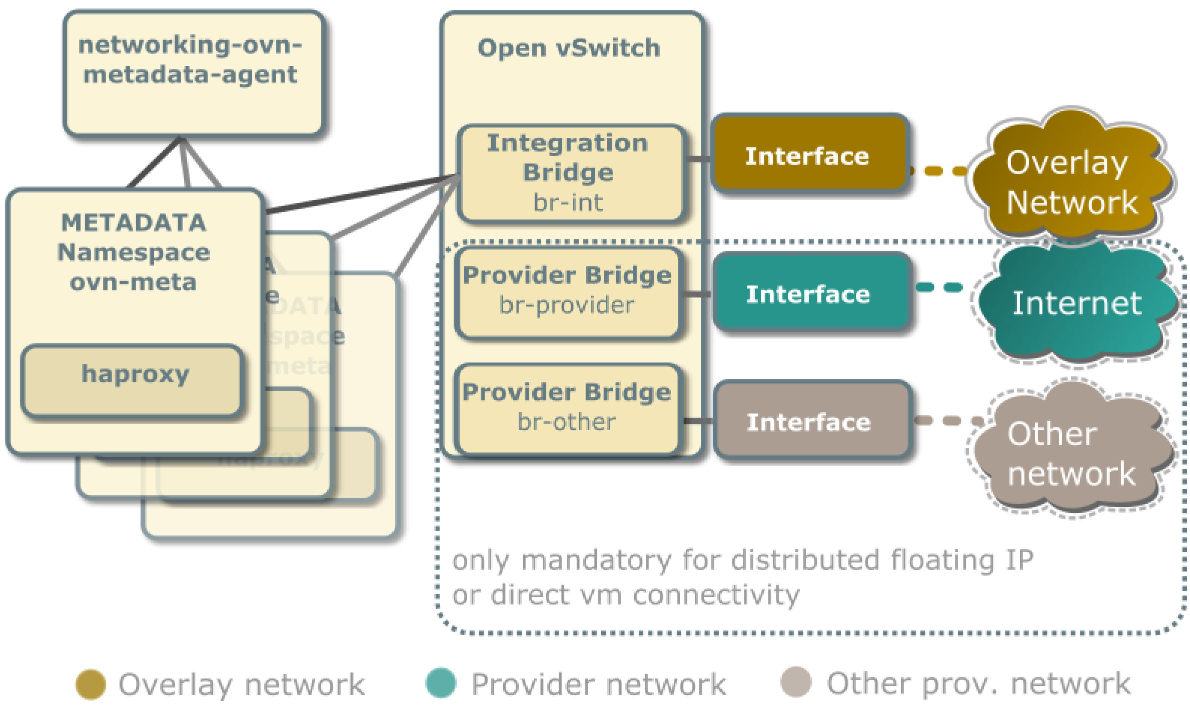
For external connectivity, gateway nodes have to set `ovn-cms-options` with `enable-chassis-as-gw` in Open_vSwitch tables external_ids column, for example:

```
$ ovs-vsctl set open . external-ids:ovn-cms-options="enable-chassis-as-gw"
```

Distributed Floating IPs (DVR)

In this architecture, the floating IP N/S traffic flows directly from/to the compute nodes through the specific provider network bridge. In this case compute nodes need connectivity to the external network.

Each compute node contains the following network components:



Note: The Networking service creates a unique network namespace for each virtual network that enables the metadata service.

Several external connections can be optionally created via provider bridges. Those can be used for direct vm connectivity to the specific networks or the use of distributed floating ips.

Accessing OVN database content

OVN stores configuration data in a collection of OVS database tables. The following commands show the contents of the most common database tables in the northbound and southbound databases. The example database output in this section uses these commands with various output filters.

```
$ ovn-nbctl list Logical_Switch
$ ovn-nbctl list Logical_Switch_Port
$ ovn-nbctl list ACL
$ ovn-nbctl list Address_Set
$ ovn-nbctl list Logical_Router
$ ovn-nbctl list Logical_Router_Port
$ ovn-nbctl list Gateway_Chassis

$ ovn-sbctl list Chassis
$ ovn-sbctl list Encap
```

(continues on next page)

(continued from previous page)

```
$ ovn-nbctl list Address_Set
$ ovn-sbctl lflow-list
$ ovn-sbctl list Multicast_Group
$ ovn-sbctl list Datapath_Binding
$ ovn-sbctl list Port_Binding
$ ovn-sbctl list MAC_Binding
$ ovn-sbctl list Gateway_Chassis
```

Note: By default, you must run these commands from the node containing the OVN databases.

Adding a compute node

When you add a compute node to the environment, the OVN controller service on it connects to the OVN southbound database and registers the node as a chassis.

```
_uuid                 : 9be8639d-1d0b-4e3d-9070-03a655073871
encaps                : [2fcefdf4-a5e7-43ed-b7b2-62039cc7e32e]
external_ids          : {ovn-bridge-mappings=""}
hostname              : "compute1"
name                  : "410ee302-850b-4277-8610-fa675d620cb7"
vtep_logical_switches: []
```

The `encaps` field value refers to tunnel endpoint information for the compute node.

```
_uuid                 : 2fcefdf4-a5e7-43ed-b7b2-62039cc7e32e
ip                    : "10.0.0.32"
options               : {}
type                  : geneve
```

Security Groups/Rules

When a Neutron Security Group is created, the equivalent Port Group in OVN (pg-<security_group_id> is created). This Port Group references Neutron SG id in its external_ids column.

When a Neutron Port is created, the equivalent Logical Port in OVN is added to those Port Groups associated to the Neutron Security Groups this port belongs to.

When a Neutron Port is deleted, the associated Logical Port in OVN is deleted. Since the schema includes a weak reference to the port, when the LSP gets deleted, it is automatically deleted from any Port Group entry where it was previously present.

Every time a security group rule is created, instead of figuring out the ports affected by its SG and inserting an ACL row which will be referenced by different Logical Switches, we just reference it from the associated Port Group.

OVN operations

1. Creating a security group will cause the OVN mechanism driver to create a port group in the Port_Group table of the northbound DB:

```
_uuid               : e96c5994-695d-4b9c-a17b-c7375ad281e2
acls                : [33c3c2d0-bc7b-421b-ace9-10884851521a, c22170ec-
↪da5d-4a59-b118-f7f0e370ebc4]
external_ids        : {"neutron:security_group_id"="ccbeffee-7b98-
↪4b6f-adf7-d42027ca6447"}
name                : pg_ccbeffee_7b98_4b6f_adf7_d42027ca6447
ports               : []
```

And it also creates the default ACLs for egress traffic in the ACL table of the northbound DB:

```
_uuid               : 33c3c2d0-bc7b-421b-ace9-10884851521a
action              : allow-related
direction           : from-lport
external_ids        : {"neutron:security_group_rule_id"="655b0d7e-
↪144e-4bd8-9243-10a261b91041"}
log                 : false
match               : "inport == @pg_ccbeffee_7b98_4b6f_adf7_
↪d42027ca6447 && ip4"
meter               : []
name                : []
priority            : 1002
severity            : []

_uuid               : c22170ec-da5d-4a59-b118-f7f0e370ebc4
action              : allow-related
direction           : from-lport
external_ids        : {"neutron:security_group_rule_id"="a303a34f-
↪5f19-494f-a9e2-e23f246bfcad"}
log                 : false
match               : "inport == @pg_ccbeffee_7b98_4b6f_adf7_
↪d42027ca6447 && ip6"
meter               : []
name                : []
priority            : 1002
severity            : []
```

Ports with no security groups

When a port doesnt belong to any Security Group and port security is enabled, we, by default, drop all the traffic to/from that port. In order to implement this through Port Groups, well create a special Port Group with a fixed name (`neutron_pg_drop`) which holds the ACLs to drop all the traffic.

This PG is created automatically once before neutron-server forks into workers.

Networks

Provider networks

A provider (external) network bridges instances to physical network infrastructure that provides layer-3 services. In most cases, provider networks implement layer-2 segmentation using VLAN IDs. A provider network maps to a provider bridge on each compute node that supports launching instances on the provider network. You can create more than one provider bridge, each one requiring a unique name and underlying physical network interface to prevent switching loops. Provider networks and bridges can use arbitrary names, but each mapping must reference valid provider network and bridge names. Each provider bridge can contain one `flat` (untagged) network and up to the maximum number of `vlan` (tagged) networks that the physical network infrastructure supports, typically around 4000.

Creating a provider network involves several commands at the host, OVS, and Networking service levels that yield a series of operations at the OVN level to create the virtual network components. The following example creates a `flat` provider network `provider` using the provider bridge `br-provider` and binds a subnet to it.

Create a provider network

1. On each compute node, create the provider bridge, map the provider network to it, and add the underlying physical or logical (typically a bond) network interface to it.

   ```
   # ovs-vsctl --may-exist add-br br-provider -- set bridge br-provider \
     protocols=OpenFlow13
   # ovs-vsctl set Open_vSwitch . external-ids:ovn-bridge-
   ↪mappings=provider:br-provider
   # ovs-vsctl --may-exist add-port br-provider INTERFACE_NAME
   ```

   Replace `INTERFACE_NAME` with the name of the underlying network interface.

   Note: These commands provide no output if successful.

2. On the controller node, source the administrative project credentials.

3. On the controller node, to enable this chassis to host gateway routers for external connectivity, set ovn-cms-options to enable-chassis-as-gw.

   ```
   # ovs-vsctl set Open_vSwitch . external-ids:ovn-cms-options="enable-
   ↪chassis-as-gw"
   ```

   Note: This command provide no output if successful.

4. On the controller node, create the provider network in the Networking service. In this case, instances and routers in other projects can use the network.

   ```
   $ openstack network create --external --share \
     --provider-physical-network provider --provider-network-type flat \
     provider
   +---------------------------+--------------------------------------+
   ```

   (continues on next page)

(continued from previous page)

```
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        | nova                                 |
| created_at                | 2016-06-15 15:50:37+00:00            |
| description               |                                      |
| id                        | 0243277b-4aa8-46d8-9e10-5c9ad5e01521 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| mtu                       | 1500                                 |
| name                      | provider                             |
| project_id                | b1ebf33664df402693f729090cfab861     |
| provider:network_type     | flat                                 |
| provider:physical_network | provider                             |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| router:external           | External                             |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   | 32a61337-c5a3-448a-a1e7-c11d6f062c21 |
| tags                      | []                                   |
| updated_at                | 2016-06-15 15:50:37+00:00            |
+---------------------------+--------------------------------------+
```

Note: The value of `--provider-physical-network` must refer to the provider network name in the mapping.

OVN operations

The OVN mechanism driver and OVN perform the following operations during creation of a provider network.

1. The mechanism driver translates the network into a logical switch in the OVN northbound database.

```
 _uuid                : 98edf19f-2dbc-4182-af9b-79cafa4794b6
 acls                 : []
 external_ids         : {"neutron:network_name"=provider}
 load_balancer        : []
 name                 : "neutron-e4abf6df-f8cf-49fd-85d4-3ea399f4d645"
 ports                : [92ee7c2f-cd22-4cac-a9d9-68a374dc7b17]

.. note::

   The ``neutron:network_name`` field in ``external_ids`` contains
   the network name and ``name`` contains the network UUID.
```

2. In addition, because the provider network is handled by a separate bridge, the following logical port is created in the OVN northbound database.

```
_uuid               : 92ee7c2f-cd22-4cac-a9d9-68a374dc7b17
addresses           : [unknown]
enabled             : []
external_ids        : {}
name                : "provnet-e4abf6df-f8cf-49fd-85d4-3ea399f4d645"
options             : {network_name=provider}
parent_name         : []
port_security       : []
tag                 : []
type                : localnet
up                  : false
```

3. The OVN northbound service translates these objects into datapath bindings, port bindings, and the appropriate multicast groups in the OVN southbound database.

   - Datapath bindings

```
_uuid               : f1f0981f-a206-4fac-b3a1-dc2030c9909f
external_ids        : {logical-switch="98edf19f-2dbc-4182-af9b-
↪79cafa4794b6"}
tunnel_key          : 109
```

   - Port bindings

```
_uuid               : 8427506e-46b5-41e5-a71b-a94a6859e773
chassis             : []
datapath            : f1f0981f-a206-4fac-b3a1-dc2030c9909f
logical_port        : "provnet-e4abf6df-f8cf-49fd-85d4-
↪3ea399f4d645"
mac                 : [unknown]
options             : {network_name=provider}
parent_port         : []
tag                 : []
tunnel_key          : 1
type                : localnet
```

   - Logical flows

```
Datapath: f1f0981f-a206-4fac-b3a1-dc2030c9909f  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 == "provnet-e4abf6df-f8cf-49fd-85d4-3ea399f4d645
↪"),
    action=(next;)
  table= 1(  ls_in_port_sec_ip), priority=    0, match=(1),
    action=(next;)
  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),
```

(continues on next page)

(continued from previous page)

```
    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_lb), priority=    0, match=(1),
    action=(next;)
  table= 8(     ls_in_stateful), priority=  100, match=(reg0[1]␣
↪== 1),
    action=(ct_commit; next;)
  table= 8(     ls_in_stateful), priority=  100, match=(reg0[2]␣
↪== 1),
    action=(ct_lb;)
  table= 8(     ls_in_stateful), priority=    0, match=(1),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=  100,
    match=(inport == "provnet-e4abf6df-f8cf-49fd-85d4-3ea399f4d645
↪"),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=    0, match=(1),
    action=(next;)
  table=10(      ls_in_l2_lkup), priority=  100, match=(eth.
↪mcast),
    action=(outport = "_MC_flood"; output;)
  table=10(      ls_in_l2_lkup), priority=    0, match=(1),
    action=(outport = "_MC_unknown"; output;)
Datapath: f1f0981f-a206-4fac-b3a1-dc2030c9909f  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_stateful), priority=  100, match=(reg0[1]␣
↪== 1),
    action=(ct_commit; next;)
  table= 5(    ls_out_stateful), priority=  100, match=(reg0[2]␣
↪== 1),
    action=(ct_lb;)
  table= 5(    ls_out_stateful), priority=    0, match=(1),
    action=(next;)
  table= 6( ls_out_port_sec_ip), priority=    0, match=(1),
    action=(next;)
  table= 7( ls_out_port_sec_l2), priority=  100, match=(eth.
↪mcast),
    action=(output;)
```

(continues on next page)

(continued from previous page)

```
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "provnet-e4abf6df-f8cf-49fd-85d4-
↪3ea399f4d645"),
    action=(output;)
```

- Multicast groups

```
_uuid                   : 0102f08d-c658-4d0a-a18a-ec8adcaddf4f
datapath                : f1f0981f-a206-4fac-b3a1-dc2030c9909f
name                    : _MC_unknown
ports                   : [8427506e-46b5-41e5-a71b-a94a6859e773]
tunnel_key              : 65534

_uuid                   : fbc38e51-ac71-4c57-a405-e6066e4c101e
datapath                : f1f0981f-a206-4fac-b3a1-dc2030c9909f
name                    : _MC_flood
ports                   : [8427506e-46b5-41e5-a71b-a94a6859e773]
tunnel_key              : 65535
```

Create a subnet on the provider network

The provider network requires at least one subnet that contains the IP address allocation available for instances, default gateway IP address, and metadata such as name resolution.

1. On the controller node, create a subnet bound to the provider network `provider`.

```
$ openstack subnet create --network provider --subnet-range \
  203.0.113.0/24 --allocation-pool start=203.0.113.101,end=203.0.113.
↪250 \
  --dns-nameserver 8.8.8.8,8.8.4.4 --gateway 203.0.113.1 provider-v4
  +-------------------+--------------------------------------+
  | Field             | Value                                |
  +-------------------+--------------------------------------+
  | allocation_pools  | 203.0.113.101-203.0.113.250          |
  | cidr              | 203.0.113.0/24                       |
  | created_at        | 2016-06-15 15:50:45+00:00            |
  | description       |                                      |
  | dns_nameservers   | 8.8.8.8, 8.8.4.4                     |
  | enable_dhcp       | True                                 |
  | gateway_ip        | 203.0.113.1                          |
  | host_routes       |                                      |
  | id                | 32a61337-c5a3-448a-a1e7-c11d6f062c21 |
  | ip_version        | 4                                    |
  | ipv6_address_mode | None                                 |
  | ipv6_ra_mode      | None                                 |
  | name              | provider-v4                          |
  | network_id        | 0243277b-4aa8-46d8-9e10-5c9ad5e01521 |
  | project_id        | b1ebf33664df402693f729090cfab861     |
  | subnetpool_id     | None                                 |
  | updated_at        | 2016-06-15 15:50:45+00:00            |
  +-------------------+--------------------------------------+
```

If using DHCP to manage instance IP addresses, adding a subnet causes a series of operations in the Networking service and OVN.

- The Networking service schedules the network on appropriate number of DHCP agents. The example environment contains three DHCP agents.
- Each DHCP agent spawns a network namespace with a `dnsmasq` process using an IP address from the subnet allocation.
- The OVN mechanism driver creates a logical switch port object in the OVN northbound database for each `dnsmasq` process.

OVN operations

The OVN mechanism driver and OVN perform the following operations during creation of a subnet on the provider network.

1. If the subnet uses DHCP for IP address management, create logical ports ports for each DHCP agent serving the subnet and bind them to the logical switch. In this example, the subnet contains two DHCP agents.

```
_uuid                : 5e144ab9-3e08-4910-b936-869bbbf254c8
addresses            : ["fa:16:3e:57:f9:ca 203.0.113.101"]
enabled              : true
external_ids         : {"neutron:port_name"=""}
name                 : "6ab052c2-7b75-4463-b34f-fd3426f61787"
options              : {}
parent_name          : []
port_security        : []
tag                  : []
type                 : ""
up                   : true

_uuid                : 38cf8b52-47c4-4e93-be8d-06bf71f6a7c9
addresses            : ["fa:16:3e:e0:eb:6d 203.0.113.102"]
enabled              : true
external_ids         : {"neutron:port_name"=""}
name                 : "94aee636-2394-48bc-b407-8224ab6bb1ab"
options              : {}
parent_name          : []
port_security        : []
tag                  : []
type                 : ""
up                   : true

_uuid                : 924500c4-8580-4d5f-a7ad-8769f6e58ff5
acls                 : []
external_ids         : {"neutron:network_name"=provider}
load_balancer        : []
name                 : "neutron-670efade-7cd0-4d87-8a04-27f366eb8941"
ports                : [38cf8b52-47c4-4e93-be8d-06bf71f6a7c9,
                        5e144ab9-3e08-4910-b936-869bbbf254c8,
                        a576b812-9c3e-4cfb-9752-5d8500b3adf9]
```

2. The OVN northbound service creates port bindings for these logical ports and adds them to the appropriate multicast group.

   - Port bindings

```
_uuid                 : 030024f4-61c3-4807-859b-07727447c427
chassis               : fc5ab9e7-bc28-40e8-ad52-2949358cc088
datapath              : bd0ab2b3-4cf4-4289-9529-ef430f6a89e6
logical_port          : "6ab052c2-7b75-4463-b34f-fd3426f61787"
mac                   : ["fa:16:3e:57:f9:ca 203.0.113.101"]
options               : {}
parent_port           : []
tag                   : []
tunnel_key            : 2
type                  : ""

_uuid                 : cc5bcd19-bcae-4e29-8cee-3ec8a8a75d46
chassis               : 6a9d0619-8818-41e6-abef-2f3d9a597c03
datapath              : bd0ab2b3-4cf4-4289-9529-ef430f6a89e6
logical_port          : "94aee636-2394-48bc-b407-8224ab6bb1ab"
mac                   : ["fa:16:3e:e0:eb:6d 203.0.113.102"]
options               : {}
parent_port           : []
tag                   : []
tunnel_key            : 3
type                  : ""
```

- Multicast groups

```
_uuid                 : 39b32ccd-fa49-4046-9527-13318842461e
datapath              : bd0ab2b3-4cf4-4289-9529-ef430f6a89e6
name                  : _MC_flood
ports                 : [030024f4-61c3-4807-859b-07727447c427,
                          904c3108-234d-41c0-b93c-116b7e352a75,
                          cc5bcd19-bcae-4e29-8cee-3ec8a8a75d46]
tunnel_key            : 65535
```

3. The OVN northbound service translates the logical ports into additional logical flows in the OVN southbound database.

```
Datapath: bd0ab2b3-4cf4-4289-9529-ef430f6a89e6  Pipeline: ingress
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "94aee636-2394-48bc-b407-8224ab6bb1ab"),
    action=(next;)
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "6ab052c2-7b75-4463-b34f-fd3426f61787"),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 203.0.113.101 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:57:f9:ca;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:57:f9:ca; arp.tpa = arp.spa;
            arp.spa = 203.0.113.101; outport = inport; inport = "";
            /* Allow sending out inport. */ output;)
  table= 9(      ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 203.0.113.102 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:e0:eb:6d;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:e0:eb:6d; arp.tpa = arp.spa;
            arp.spa = 203.0.113.102; outport = inport;
            inport = ""; /* Allow sending out inport. */ output;)
```

(continues on next page)

(continued from previous page)

```
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:57:f9:ca),
    action=(outport = "6ab052c2-7b75-4463-b34f-fd3426f61787"; output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:e0:eb:6d),
    action=(outport = "94aee636-2394-48bc-b407-8224ab6bb1ab"; output;)
Datapath: bd0ab2b3-4cf4-4289-9529-ef430f6a89e6  Pipeline: egress
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "6ab052c2-7b75-4463-b34f-fd3426f61787"),
    action=(output;)
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "94aee636-2394-48bc-b407-8224ab6bb1ab"),
    action=(output;)
```

4. For each compute node without a DHCP agent on the subnet:

   - The OVN controller service translates the logical flows into flows on the integration bridge `br-int`.

```
cookie=0x0, duration=22.303s, table=32, n_packets=0, n_bytes=0,
    idle_age=22, priority=100,reg7=0xffff,metadata=0x4
    actions=load:0x4->NXM_NX_TUN_ID[0..23],
        set_field:0xffff/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],
        output:5,output:4,resubmit(,33)
```

5. For each compute node with a DHCP agent on a subnet:

   - Creation of a DHCP network namespace adds two virtual switch ports. The first port connects the DHCP agent with `dnsmasq` process to the integration bridge and the second port patches the integration bridge to the provider bridge `br-provider`.

```
# ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:000022024a1dc045
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_
↪MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_
↪dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src␣
↪mod_tp_dst
 7(tap6ab052c2-7b): addr:00:00:00:00:10:7f
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 8(patch-br-int-to): addr:6a:8c:30:3f:d7:dd
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max

# ovs-ofctl -O OpenFlow13 show br-provider
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000080027137c4a
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS QUEUE_
↪STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(patch-provnet-0): addr:fa:42:c5:3f:d7:6f
```

(continues on next page)

(continued from previous page)

```
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
```

- The OVN controller service translates these logical flows into flows on the integration bridge.

```
cookie=0x0, duration=17.731s, table=0, n_packets=3, n_bytes=258,
    idle_age=16, priority=100,in_port=7
    actions=load:0x2->NXM_NX_REG5[],load:0x4->OXM_OF_METADATA[],
        load:0x2->NXM_NX_REG6[],resubmit(,16)
cookie=0x0, duration=17.730s, table=0, n_packets=15, n_bytes=954,
    idle_age=2, priority=100,in_port=8,vlan_tci=0x0000/0x1000
    actions=load:0x1->NXM_NX_REG5[],load:0x4->OXM_OF_METADATA[],
        load:0x1->NXM_NX_REG6[],resubmit(,16)
cookie=0x0, duration=17.730s, table=0, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,in_port=8,dl_vlan=0
    actions=strip_vlan,load:0x1->NXM_NX_REG5[],
        load:0x4->OXM_OF_METADATA[],load:0x1->NXM_NX_REG6[],
        resubmit(,16)
cookie=0x0, duration=17.732s, table=16, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,metadata=0x4,
        dl_src=01:00:00:00:00:00/01:00:00:00:00:00
    actions=drop
cookie=0x0, duration=17.732s, table=16, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,metadata=0x4,vlan_tci=0x1000/0x1000
    actions=drop
cookie=0x0, duration=17.732s, table=16, n_packets=3, n_bytes=258,
    idle_age=16, priority=50,reg6=0x2,metadata=0x4␣
↪actions=resubmit(,17)
cookie=0x0, duration=17.732s, table=16, n_packets=0, n_bytes=0,
    idle_age=17, priority=50,reg6=0x3,metadata=0x4␣
↪actions=resubmit(,17)
cookie=0x0, duration=17.732s, table=16, n_packets=15, n_bytes=954,
    idle_age=2, priority=50,reg6=0x1,metadata=0x4␣
↪actions=resubmit(,17)
cookie=0x0, duration=21.714s, table=17, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,18)
cookie=0x0, duration=21.714s, table=18, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,19)
cookie=0x0, duration=21.714s, table=19, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,20)
cookie=0x0, duration=21.714s, table=20, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,21)
cookie=0x0, duration=21.714s, table=21, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x1/0x1,metadata=0x4
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=21.714s, table=21, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x1/0x1,metadata=0x4
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=21.714s, table=21, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,22)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=21.714s, table=22, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,23)
cookie=0x0, duration=21.714s, table=23, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,24)
cookie=0x0, duration=21.714s, table=24, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x4/0x4,metadata=0x4
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=21.714s, table=24, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x4/0x4,metadata=0x4
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=21.714s, table=24, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x2/0x2,metadata=0x4
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=21.714s, table=24, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x2/0x2,metadata=0x4
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=21.714s, table=24, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,25)
cookie=0x0, duration=21.714s, table=25, n_packets=15, n_bytes=954,
    idle_age=6, priority=100,reg6=0x1,metadata=0x4␣
↪actions=resubmit(,26)
cookie=0x0, duration=21.714s, table=25, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,arp,metadata=0x4,
        arp_tpa=203.0.113.101,arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:f9:5d:f3,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ef95df3->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a81264->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,
↪32)
cookie=0x0, duration=21.714s, table=25, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,arp,metadata=0x4,
        arp_tpa=203.0.113.102,arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:f0:a5:9f,
        load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ef0a59f->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a81265->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,
↪32)
cookie=0x0, duration=21.714s, table=25, n_packets=3, n_bytes=258,
    idle_age=20, priority=0,metadata=0x4 actions=resubmit(,26)
cookie=0x0, duration=21.714s, table=26, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=100,metadata=0x4,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=load:0xffff->NXM_NX_REG7[],resubmit(,32)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=21.714s, table=26, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,metadata=0x4,dl_dst=fa:16:3e:f0:a5:9f
    actions=load:0x3->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=21.714s, table=26, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,metadata=0x4,dl_dst=fa:16:3e:f9:5d:f3
    actions=load:0x2->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=21.714s, table=26, n_packets=0, n_bytes=0,
    idle_age=21, priority=0,metadata=0x4
    actions=load:0xfffe->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=17.731s, table=33, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,reg7=0x2,metadata=0x4
    actions=load:0x2->NXM_NX_REG5[],resubmit(,34)
cookie=0x0, duration=118.126s, table=33, n_packets=0, n_bytes=0,
    idle_age=118, hard_age=17, priority=100,reg7=0xfffe,
↪metadata=0x4
    actions=load:0x1->NXM_NX_REG5[],load:0x1->NXM_NX_REG7[],
        resubmit(,34),load:0xfffe->NXM_NX_REG7[]
cookie=0x0, duration=118.126s, table=33, n_packets=18, n_
↪bytes=1212,
    idle_age=2, hard_age=17, priority=100,reg7=0xffff,metadata=0x4
    actions=load:0x2->NXM_NX_REG5[],load:0x2->NXM_NX_REG7[],
        resubmit(,34),load:0x1->NXM_NX_REG5[],load:0x1->NXM_NX_
↪REG7[],
        resubmit(,34),load:0xffff->NXM_NX_REG7[]
cookie=0x0, duration=17.730s, table=33, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,reg7=0x1,metadata=0x4
    actions=load:0x1->NXM_NX_REG5[],resubmit(,34)
cookie=0x0, duration=17.697s, table=33, n_packets=0, n_bytes=0,
    idle_age=17, priority=100,reg7=0x3,metadata=0x4
    actions=load:0x1->NXM_NX_REG7[],resubmit(,33)
cookie=0x0, duration=17.731s, table=34, n_packets=3, n_bytes=258,
    idle_age=16, priority=100,reg6=0x2,reg7=0x2,metadata=0x4
    actions=drop
cookie=0x0, duration=17.730s, table=34, n_packets=15, n_bytes=954,
    idle_age=2, priority=100,reg6=0x1,reg7=0x1,metadata=0x4
    actions=drop
cookie=0x0, duration=21.714s, table=48, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,49)
cookie=0x0, duration=21.714s, table=49, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,50)
cookie=0x0, duration=21.714s, table=50, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x1/0x1,metadata=0x4
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=21.714s, table=50, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x1/0x1,metadata=0x4
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=21.714s, table=50, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,51)
cookie=0x0, duration=21.714s, table=51, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,52)
cookie=0x0, duration=21.714s, table=52, n_packets=18, n_
↪bytes=1212,
```

(continues on next page)

(continued from previous page)

```
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,53)
cookie=0x0, duration=21.714s, table=53, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x4/0x4,metadata=0x4
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=21.714s, table=53, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x4/0x4,metadata=0x4
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=21.714s, table=53, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ipv6,reg0=0x2/0x2,metadata=0x4
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=21.714s, table=53, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,ip,reg0=0x2/0x2,metadata=0x4
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=21.714s, table=53, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,54)
cookie=0x0, duration=21.714s, table=54, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=0,metadata=0x4 actions=resubmit(,55)
cookie=0x0, duration=21.714s, table=55, n_packets=18, n_
↪bytes=1212,
    idle_age=6, priority=100,metadata=0x4,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=resubmit(,64)
cookie=0x0, duration=21.714s, table=55, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,reg7=0x3,metadata=0x4
    actions=resubmit(,64)
cookie=0x0, duration=21.714s, table=55, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,reg7=0x2,metadata=0x4
    actions=resubmit(,64)
cookie=0x0, duration=21.714s, table=55, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,reg7=0x1,metadata=0x4
    actions=resubmit(,64)
cookie=0x0, duration=21.712s, table=64, n_packets=15, n_bytes=954,
    idle_age=6, priority=100,reg7=0x3,metadata=0x4␣
↪actions=output:7
cookie=0x0, duration=21.711s, table=64, n_packets=3, n_bytes=258,
    idle_age=20, priority=100,reg7=0x1,metadata=0x4␣
↪actions=output:8
```

Self-service networks

A self-service (project) network includes only virtual components, thus enabling projects to manage them without additional configuration of the underlying physical network. The OVN mechanism driver supports Geneve and VLAN network types with a preference toward Geneve. Projects can choose to isolate self-service networks, connect two or more together via routers, or connect them to provider networks via routers with appropriate capabilities. Similar to provider networks, self-service networks can use arbitrary names.

Note: Similar to provider networks, self-service VLAN networks map to a unique bridge on each compute node that supports launching instances on those networks. Self-service VLAN networks also require several commands at the host and OVS levels. The following example assumes use of Geneve self-service networks.

Create a self-service network

Creating a self-service network involves several commands at the Networking service level that yield a series of operations at the OVN level to create the virtual network components. The following example creates a Geneve self-service network and binds a subnet to it. The subnet uses DHCP to distribute IP addresses to instances.

1. On the controller node, source the credentials for a regular (non-privileged) project. The following example uses the `demo` project.
2. On the controller node, create a self-service network in the Networking service.

```
$ openstack network create selfservice
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2016-06-09T15:42:41                  |
| description             |                                      |
| id                      | f49791f7-e653-4b43-99b1-0f5557c313e4 |
| ipv4_address_scope      | None                                 |
| ipv6_address_scope      | None                                 |
| mtu                     | 1442                                 |
| name                    | selfservice                          |
| port_security_enabled   | True                                 |
| project_id              | 1ef26f483b9d44e8ac0c97388d6cb609     |
| router_external         | Internal                             |
| shared                  | False                                |
| status                  | ACTIVE                               |
| subnets                 |                                      |
| tags                    | []                                   |
| updated_at              | 2016-06-09T15:42:41                  |
+-------------------------+--------------------------------------+
```

OVN operations

The OVN mechanism driver and OVN perform the following operations during creation of a self-service network.

1. The mechanism driver translates the network into a logical switch in the OVN northbound database.

```
uuid                   : 0ab40684-7cf8-4d6c-ae8b-9d9143762d37
acls                    : []
external_ids            : {"neutron:network_name"="selfservice"}
name                    : "neutron-d5aadceb-d8d6-41c8-9252-c5e0fe6c26a5"
ports                   : []
```

2. The OVN northbound service translates this object into new datapath bindings and logical flows in the OVN southbound database.

   - Datapath bindings

```
_uuid                : 0b214af6-8910-489c-926a-fd0ed16a8251
external_ids         : {logical-switch="15e2c80b-1461-4003-9869-
↪80416cd97de5"}
tunnel_key           : 5
```

   - Logical flows

```
Datapath: 0b214af6-8910-489c-926a-fd0ed16a8251  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= 1(  ls_in_port_sec_ip), priority=    0, match=(1),
    action=(next;)
  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_lb), priority=    0, match=(1),
    action=(next;)
  table= 8(     ls_in_stateful), priority=  100, match=(reg0[2]␣
↪== 1),
    action=(ct_lb;)
  table= 8(     ls_in_stateful), priority=  100, match=(reg0[1]␣
↪== 1),
    action=(ct_commit; next;)
  table= 8(     ls_in_stateful), priority=    0, match=(1),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=    0, match=(1),
    action=(next;)
  table=10(      ls_in_l2_lkup), priority=  100, match=(eth.
↪mcast),
    action=(outport = "_MC_flood"; output;)
Datapath: 0b214af6-8910-489c-926a-fd0ed16a8251  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;)
```

(continues on next page)

(continued from previous page)

```
  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_stateful), priority=  100, match=(reg0[1]␣
↪== 1),
    action=(ct_commit; next;)
  table= 5(    ls_out_stateful), priority=  100, match=(reg0[2]␣
↪== 1),
    action=(ct_lb;)
  table= 5(    ls_out_stateful), priority=    0, match=(1),
    action=(next;)
  table= 6( ls_out_port_sec_ip), priority=    0, match=(1),
    action=(next;)
  table= 7( ls_out_port_sec_l2), priority=  100, match=(eth.
↪mcast),
    action=(output;)
```

Note: These actions do not create flows on any nodes.

Create a subnet on the self-service network

A self-service network requires at least one subnet. In most cases, the environment provides suitable values for IP address allocation for instances, default gateway IP address, and metadata such as name resolution.

1. On the controller node, create a subnet bound to the self-service network `selfservice`.

```
$ openstack subnet create --network selfservice --subnet-range 192.
↪168.1.0/24 selfservice-v4
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.1.2-192.168.1.254            |
| cidr              | 192.168.1.0/24                       |
| created_at        | 2016-06-16 00:19:08+00:00            |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.1.1                          |
| headers           |                                      |
| host_routes       |                                      |
| id                | 8f027f25-0112-45b9-a1b9-2f8097c57219 |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | selfservice-v4                       |
| network_id        | 8ed4e43b-63ef-41ed-808b-b59f1120aec0 |
| project_id        | b1ebf33664df402693f729090cfab861     |
| subnetpool_id     | None                                 |
```

(continues on next page)

(continued from previous page)

```
| updated_at          | 2016-06-16 00:19:08+00:00            |
+-------------------+--------------------------------------+
```

OVN operations

The OVN mechanism driver and OVN perform the following operations during creation of a subnet on a self-service network.

1. If the subnet uses DHCP for IP address management, create logical ports ports for each DHCP agent serving the subnet and bind them to the logical switch. In this example, the subnet contains two DHCP agents.

```
_uuid                 : 1ed7c28b-dc69-42b8-bed6-46477bb8b539
addresses             : ["fa:16:3e:94:db:5e 192.168.1.2"]
enabled               : true
external_ids          : {"neutron:port_name"=""}
name                  : "0cfbbdca-ff58-4cf8-a7d3-77daaebe3056"
options               : {}
parent_name           : []
port_security         : []
tag                   : []
type                  : ""
up                    : true

_uuid                 : ae10a5e0-db25-4108-b06a-d2d5c127d9c4
addresses             : ["fa:16:3e:90:bd:f1 192.168.1.3"]
enabled               : true
external_ids          : {"neutron:port_name"=""}
name                  : "74930ace-d939-4bca-b577-fccba24c3fca"
options               : {}
parent_name           : []
port_security         : []
tag                   : []
type                  : ""
up                    : true

_uuid                 : 0ab40684-7cf8-4d6c-ae8b-9d9143762d37
acls                  : []
external_ids          : {"neutron:network_name"="selfservice"}
name                  : "neutron-d5aadceb-d8d6-41c8-9252-c5e0fe6c26a5"
ports                 : [1ed7c28b-dc69-42b8-bed6-46477bb8b539,
                         ae10a5e0-db25-4108-b06a-d2d5c127d9c4]
```

2. The OVN northbound service creates port bindings for these logical ports and adds them to the appropriate multicast group.

   - Port bindings

```
_uuid                 : 3e463ca0-951c-46fd-b6cf-05392fa3aa1f
chassis               : 6a9d0619-8818-41e6-abef-2f3d9a597c03
datapath              : 0b214af6-8910-489c-926a-fd0ed16a8251
logical_port          : "a203b410-97c1-4e4a-b0c3-558a10841c16"
mac                   : ["fa:16:3e:a1:dc:58 192.168.1.3"]
options               : {}
```

(continues on next page)

(continued from previous page)

```
parent_port          : []
tag                  : []
tunnel_key           : 2
type                 : ""

_uuid                : fa7b294d-2a62-45ae-8de3-a41c002de6de
chassis              : d63e8ae8-caf3-4a6b-9840-5c3a57febcac
datapath             : 0b214af6-8910-489c-926a-fd0ed16a8251
logical_port         : "39b23721-46f4-4747-af54-7e12f22b3397"
mac                  : ["fa:16:3e:1a:b4:23 192.168.1.2"]
options              : {}
parent_port          : []
tag                  : []
tunnel_key           : 1
type                 : ""
```

- Multicast groups

```
_uuid                : c08d0102-c414-4a47-98d9-dd3fa9f9901c
datapath             : 0b214af6-8910-489c-926a-fd0ed16a8251
name                 : _MC_flood
ports                : [3e463ca0-951c-46fd-b6cf-05392fa3aa1f,
                        fa7b294d-2a62-45ae-8de3-a41c002de6de]
tunnel_key           : 65535
```

3. The OVN northbound service translates the logical ports into logical flows in the OVN southbound database.

```
Datapath: 0b214af6-8910-489c-926a-fd0ed16a8251  Pipeline: ingress
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "39b23721-46f4-4747-af54-7e12f22b3397"),
    action=(next;)
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "a203b410-97c1-4e4a-b0c3-558a10841c16"),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 192.168.1.2 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:1a:b4:23;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:1a:b4:23; arp.tpa = arp.spa;
            arp.spa = 192.168.1.2; outport = inport;
            inport = ""; /* Allow sending out inport. */ output;)
  table= 9(      ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 192.168.1.3 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:a1:dc:58;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:a1:dc:58; arp.tpa = arp.spa;
            arp.spa = 192.168.1.3; outport = inport;
            inport = ""; /* Allow sending out inport. */ output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:a1:dc:58),
    action=(outport = "a203b410-97c1-4e4a-b0c3-558a10841c16"; output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:1a:b4:23),
    action=(outport = "39b23721-46f4-4747-af54-7e12f22b3397"; output;)
```

(continues on next page)

(continued from previous page)

```
Datapath: 0b214af6-8910-489c-926a-fd0ed16a8251  Pipeline: egress
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "39b23721-46f4-4747-af54-7e12f22b3397"),
    action=(output;)
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "a203b410-97c1-4e4a-b0c3-558a10841c16"),
    action=(output;)
```

4. For each compute node without a DHCP agent on the subnet:

   - The OVN controller service translates these objects into flows on the integration bridge `br-int`.

```
# ovs-ofctl dump-flows br-int
cookie=0x0, duration=9.054s, table=32, n_packets=0, n_bytes=0,
    idle_age=9, priority=100,reg7=0xffff,metadata=0x5
    actions=load:0x5->NXM_NX_TUN_ID[0..23],
        set_field:0xffff/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],
        output:4,output:3
```

5. For each compute node with a DHCP agent on the subnet:

   - Creation of a DHCP network namespace adds a virtual switch ports that connects the DHCP agent with the `dnsmasq` process to the integration bridge.

```
# ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:000022024a1dc045
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_
↪MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_
↪dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src␣
↪mod_tp_dst
 9(tap39b23721-46): addr:00:00:00:00:b0:5d
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
```

   - The OVN controller service translates these objects into flows on the integration bridge.

```
cookie=0x0, duration=21.074s, table=0, n_packets=8, n_bytes=648,
    idle_age=11, priority=100,in_port=9
    actions=load:0x2->NXM_NX_REG5[],load:0x5->OXM_OF_METADATA[],
        load:0x1->NXM_NX_REG6[],resubmit(,16)
cookie=0x0, duration=21.076s, table=16, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,metadata=0x5,
        dl_src=01:00:00:00:00:00/01:00:00:00:00:00
    actions=drop
cookie=0x0, duration=21.075s, table=16, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,metadata=0x5,vlan_tci=0x1000/0x1000
    actions=drop
cookie=0x0, duration=21.076s, table=16, n_packets=0, n_bytes=0,
    idle_age=21, priority=50,reg6=0x2,metadata=0x5
    actions=resubmit(,17)
cookie=0x0, duration=21.075s, table=16, n_packets=8, n_bytes=648,
```

(continues on next page)

(continued from previous page)

```
    idle_age=11, priority=50,reg6=0x1,metadata=0x5
    actions=resubmit(,17)
cookie=0x0, duration=21.075s, table=17, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5
    actions=resubmit(,18)
cookie=0x0, duration=21.076s, table=18, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5
    actions=resubmit(,19)
cookie=0x0, duration=21.076s, table=19, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5
    actions=resubmit(,20)
cookie=0x0, duration=21.075s, table=20, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5
    actions=resubmit(,21)
cookie=0x0, duration=5.398s, table=21, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x1/0x1,metadata=0x5
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=5.398s, table=21, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x1/0x1,metadata=0x5
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=5.398s, table=22, n_packets=6, n_bytes=508,
    idle_age=2, priority=0,metadata=0x5
    actions=resubmit(,23)
cookie=0x0, duration=5.398s, table=23, n_packets=6, n_bytes=508,
    idle_age=2, priority=0,metadata=0x5
    actions=resubmit(,24)
cookie=0x0, duration=5.398s, table=24, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x4/0x4,metadata=0x5
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=5.398s, table=24, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x4/0x4,metadata=0x5
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=5.398s, table=24, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x2/0x2,metadata=0x5
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=5.398s, table=24, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x2/0x2,metadata=0x5
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=5.399s, table=24, n_packets=6, n_bytes=508,
    idle_age=2, priority=0,metadata=0x5 actions=resubmit(,25)
cookie=0x0, duration=5.398s, table=25, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,arp,metadata=0x5,
        arp_tpa=192.168.1.2,arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:82:8b:0e,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163e828b0e->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80102->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=5.378s, table=25, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,arp,metadata=0x5,arp_tpa=192.168.1.3,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:d5:00:02,load:0x2->NXM_OF_ARP_OP[],
```

(continues on next page)

(continued from previous page)

```
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ed50002->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80103->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=5.399s, table=25, n_packets=6, n_bytes=508,
    idle_age=2, priority=0,metadata=0x5
    actions=resubmit(,26)
cookie=0x0, duration=5.399s, table=26, n_packets=6, n_bytes=508,
    idle_age=2, priority=100,metadata=0x5,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=load:0xffff->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=5.398s, table=26, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,metadata=0x5,dl_dst=fa:16:3e:d5:00:02
    actions=load:0x2->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=5.398s, table=26, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,metadata=0x5,dl_dst=fa:16:3e:82:8b:0e
    actions=load:0x1->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=21.038s, table=32, n_packets=0, n_bytes=0,
    idle_age=21, priority=100,reg7=0x2,metadata=0x5
    actions=load:0x5->NXM_NX_TUN_ID[0..23],
        set_field:0x2/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],
↪output:4
cookie=0x0, duration=21.038s, table=32, n_packets=8, n_bytes=648,
    idle_age=11, priority=100,reg7=0xffff,metadata=0x5
    actions=load:0x5->NXM_NX_TUN_ID[0..23],
        set_field:0xffff/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],
        output:4,resubmit(,33)
cookie=0x0, duration=5.397s, table=33, n_packets=12, n_bytes=1016,
    idle_age=2, priority=100,reg7=0xffff,metadata=0x5
    actions=load:0x1->NXM_NX_REG7[],resubmit(,34),
       load:0xffff->NXM_NX_REG7[]
cookie=0x0, duration=5.397s, table=33, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,reg7=0x1,metadata=0x5
    actions=resubmit(,34)
cookie=0x0, duration=21.074s, table=34, n_packets=8, n_bytes=648,
    idle_age=11, priority=100,reg6=0x1,reg7=0x1,metadata=0x5
    actions=drop
cookie=0x0, duration=21.076s, table=48, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5 actions=resubmit(,49)
cookie=0x0, duration=21.075s, table=49, n_packets=8, n_bytes=648,
    idle_age=11, priority=0,metadata=0x5 actions=resubmit(,50)
cookie=0x0, duration=5.398s, table=50, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x1/0x1,metadata=0x5
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=5.398s, table=50, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x1/0x1,metadata=0x5
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=5.398s, table=50, n_packets=6, n_bytes=508,
    idle_age=3, priority=0,metadata=0x5
    actions=resubmit(,51)
cookie=0x0, duration=5.398s, table=51, n_packets=6, n_bytes=508,
    idle_age=3, priority=0,metadata=0x5
```

(continues on next page)

(continued from previous page)

```
    actions=resubmit(,52)
cookie=0x0, duration=5.398s, table=52, n_packets=6, n_bytes=508,
    idle_age=3, priority=0,metadata=0x5
    actions=resubmit(,53)
cookie=0x0, duration=5.399s, table=53, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x4/0x4,metadata=0x5
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=5.398s, table=53, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x4/0x4,metadata=0x5
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=5.398s, table=53, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ip,reg0=0x2/0x2,metadata=0x5
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=5.398s, table=53, n_packets=0, n_bytes=0,
    idle_age=5, priority=100,ipv6,reg0=0x2/0x2,metadata=0x5
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=5.398s, table=53, n_packets=6, n_bytes=508,
    idle_age=3, priority=0,metadata=0x5
    actions=resubmit(,54)
cookie=0x0, duration=5.398s, table=54, n_packets=6, n_bytes=508,
    idle_age=3, priority=0,metadata=0x5
    actions=resubmit(,55)
cookie=0x0, duration=5.398s, table=55, n_packets=6, n_bytes=508,
    idle_age=3, priority=100,metadata=0x5,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=resubmit(,64)
cookie=0x0, duration=5.398s, table=55, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,reg7=0x1,metadata=0x5
    actions=resubmit(,64)
cookie=0x0, duration=5.398s, table=55, n_packets=0, n_bytes=0,
    idle_age=5, priority=50,reg7=0x2,metadata=0x5
    actions=resubmit(,64)
cookie=0x0, duration=5.397s, table=64, n_packets=6, n_bytes=508,
    idle_age=3, priority=100,reg7=0x1,metadata=0x5
    actions=output:9
```

Routers

Routers

Routers pass traffic between layer-3 networks.

Create a router

1. On the controller node, source the credentials for a regular (non-privileged) project. The following example uses the `demo` project.
2. On the controller node, create router in the Networking service.

```
$ openstack router create router
+-----------------------+--------------------------------------+
| Field                 | Value                                |
```

(continues on next page)

(continued from previous page)

```
+-----------------------+--------------------------------------+
| admin_state_up        | UP                                   |
| description           |                                      |
| external_gateway_info | null                                 |
| headers               |                                      |
| id                    | 24addfcd-5506-405d-a59f-003644c3d16a |
| name                  | router                               |
| project_id            | b1ebf33664df402693f729090cfab861     |
| routes                |                                      |
| status                | ACTIVE                               |
+-----------------------+--------------------------------------+
```

OVN operations

The OVN mechanism driver and OVN perform the following operations when creating a router.

1. The OVN mechanism driver translates the router into a logical router object in the OVN northbound database.

```
_uuid                : 1c2e340d-dac9-496b-9e86-1065f9dab752
default_gw           : []
enabled              : []
external_ids         : {"neutron:router_name"="router"}
name                 : "neutron-a24fd760-1a99-4eec-9f02-24bb284ff708"
ports                : []
static_routes        : []
```

2. The OVN northbound service translates this object into logical flows and datapath bindings in the OVN southbound database.

   - Datapath bindings

```
_uuid                : 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa
external_ids         : {logical-router="1c2e340d-dac9-496b-9e86-
↪1065f9dab752"}
tunnel_key           : 3
```

   - Logical flows

```
Datapath: 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa  Pipeline: ingress
  table= 0(    lr_in_admission), priority=  100,
    match=(vlan.present || eth.src[40]),
    action=(drop;)
  table= 1(     lr_in_ip_input), priority=  100,
    match=(ip4.mcast || ip4.src == 255.255.255.255 ||
           ip4.src == 127.0.0.0/8 || ip4.dst == 127.0.0.0/8 ||
           ip4.src == 0.0.0.0/8 || ip4.dst == 0.0.0.0/8),
    action=(drop;)
  table= 1(     lr_in_ip_input), priority=   50, match=(ip4.
↪mcast),
    action=(drop;)
  table= 1(     lr_in_ip_input), priority=   50, match=(eth.
↪bcast),
    action=(drop;)
```

(continues on next page)

(continued from previous page)

```
  table= 1(     lr_in_ip_input), priority=   30,
    match=(ip4 && ip.ttl == {0, 1}), action=(drop;)
  table= 1(     lr_in_ip_input), priority=    0, match=(1),
    action=(next;)
  table= 2(       lr_in_unsnat), priority=    0, match=(1),
    action=(next;)
  table= 3(         lr_in_dnat), priority=    0, match=(1),
    action=(next;)
  table= 5(  lr_in_arp_resolve), priority=    0, match=(1),
    action=(get_arp(outport, reg0); next;)
  table= 6(  lr_in_arp_request), priority=  100,
    match=(eth.dst == 00:00:00:00:00:00),
    action=(arp { eth.dst = ff:ff:ff:ff:ff:ff; arp.spa = reg1;
            arp.op = 1; output; };)
  table= 6(  lr_in_arp_request), priority=    0, match=(1),
    action=(output;)
Datapath: 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa  Pipeline: egress
  table= 0(        lr_out_snat), priority=    0, match=(1),
    action=(next;)
```

3. The OVN controller service on each compute node translates these objects into flows on the integration bridge `br-int`.

```
# ovs-ofctl dump-flows br-int
cookie=0x0, duration=6.402s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x5,vlan_tci=0x1000/0x1000
    actions=drop
cookie=0x0, duration=6.402s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x5,
    dl_src=01:00:00:00:00:00/01:00:00:00:00:00
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_dst=127.0.0.0/8
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_dst=0.0.0.0/8
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_dst=224.0.0.0/4
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,ip,metadata=0x5,nw_dst=224.0.0.0/4
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_src=255.255.255.255
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_src=127.0.0.0/8
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x5,nw_src=0.0.0.0/8
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,arp,metadata=0x5,arp_op=2
    actions=push:NXM_NX_REG0[],push:NXM_OF_ETH_SRC[],
        push:NXM_NX_ARP_SHA[],push:NXM_OF_ARP_SPA[],
```

(continues on next page)

(continued from previous page)

```
        pop:NXM_NX_REG0[],pop:NXM_OF_ETH_SRC[],
        controller(userdata=00.00.00.01.00.00.00.00),
        pop:NXM_OF_ETH_SRC[],pop:NXM_NX_REG0[]
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,metadata=0x5,dl_dst=ff:ff:ff:ff:ff:ff
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=30,ip,metadata=0x5,nw_ttl=0
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=30,ip,metadata=0x5,nw_ttl=1
    actions=drop
cookie=0x0, duration=6.402s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x5
    actions=resubmit(,18)
cookie=0x0, duration=6.402s, table=18, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x5
    actions=resubmit(,19)
cookie=0x0, duration=6.402s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x5
    actions=resubmit(,20)
cookie=0x0, duration=6.402s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x5
    actions=resubmit(,32)
cookie=0x0, duration=6.402s, table=48, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x5
    actions=resubmit(,49)
```

Attach a self-service network to the router

Self-service networks, particularly subnets, must interface with a router to enable connectivity with other self-service and provider networks.

1. On the controller node, add the self-service network subnet `selfservice-v4` to the router `router`.

```
$ openstack router add subnet router selfservice-v4
```

Note: This command provides no output.

OVN operations

The OVN mechanism driver and OVN perform the following operations when adding a subnet as an interface on a router.

1. The OVN mechanism driver translates the operation into logical objects and devices in the OVN northbound database and performs a series of operations on them.

   - Create a logical port.

```
_uuid               : 4c9e70b1-fff0-4d0d-af8e-42d3896eb76f
addresses           : ["fa:16:3e:0c:55:62 192.168.1.1"]
enabled             : true
external_ids        : {"neutron:port_name"=""}
name                : "5b72d278-5b16-44a6-9aa0-9e513a429506"
options             : {router-port="lrp-5b72d278-5b16-44a6-9aa0-
↪9e513a429506"}
parent_name         : []
port_security       : []
tag                 : []
type                : router
up                  : false
```

- Add the logical port to logical switch.

```
_uuid               : 0ab40684-7cf8-4d6c-ae8b-9d9143762d37
acls                : []
external_ids        : {"neutron:network_name"="selfservice"}
name                : "neutron-d5aadceb-d8d6-41c8-9252-
↪c5e0fe6c26a5"
ports               : [1ed7c28b-dc69-42b8-bed6-46477bb8b539,
                       4c9e70b1-fff0-4d0d-af8e-42d3896eb76f,
                       ae10a5e0-db25-4108-b06a-d2d5c127d9c4]
```

- Create a logical router port object.

```
_uuid               : f60ccb93-7b3d-4713-922c-37104b7055dc
enabled             : []
external_ids        : {}
mac                 : "fa:16:3e:0c:55:62"
name                : "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506"
network             : "192.168.1.1/24"
peer                : []
```

- Add the logical router port to the logical router object.

```
_uuid               : 1c2e340d-dac9-496b-9e86-1065f9dab752
default_gw          : []
enabled             : []
external_ids        : {"neutron:router_name"="router"}
name                : "neutron-a24fd760-1a99-4eec-9f02-
↪24bb284ff708"
ports               : [f60ccb93-7b3d-4713-922c-37104b7055dc]
static_routes       : []
```

2. The OVN northbound service translates these objects into logical flows, datapath bindings, and the appropriate multicast groups in the OVN southbound database.

   - Logical flows in the logical router datapath

```
Datapath: 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa  Pipeline: ingress
  table= 0(     lr_in_admission), priority=   50,
    match=((eth.mcast || eth.dst == fa:16:3e:0c:55:62) &&
           inport == "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506"),
    action=(next;)
  table= 1(      lr_in_ip_input), priority=  100,
```

(continues on next page)

(continued from previous page)

```
    match=(ip4.src == {192.168.1.1, 192.168.1.255}), action=(drop;
↪)
  table= 1(     lr_in_ip_input), priority=   90,
    match=(ip4.dst == 192.168.1.1 && icmp4.type == 8 &&
           icmp4.code == 0),
    action=(ip4.dst = ip4.src; ip4.src = 192.168.1.1; ip.ttl =␣
↪255;
            icmp4.type = 0;
            inport = ""; /* Allow sending out inport. */ next; )
  table= 1(     lr_in_ip_input), priority=   90,
    match=(inport == "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506" &&
           arp.tpa == 192.168.1.1 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:0c:55:62;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:0c:55:62; arp.tpa = arp.spa;
            arp.spa = 192.168.1.1;
            outport = "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506";
            inport = ""; /* Allow sending out inport. */ output;)
  table= 1(     lr_in_ip_input), priority=   60,
    match=(ip4.dst == 192.168.1.1), action=(drop;)
  table= 4(   lr_in_ip_routing), priority=   24,
    match=(ip4.dst == 192.168.1.0/255.255.255.0),
    action=(ip.ttl--; reg0 = ip4.dst; reg1 = 192.168.1.1;
            eth.src = fa:16:3e:0c:55:62;
            outport = "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506";
            next;)
Datapath: 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa  Pipeline: egress
  table= 1(    lr_out_delivery), priority=  100,
    match=(outport == "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506),
    action=(output;)
```

- Logical flows in the logical switch datapath

```
Datapath: 611d35e8-b1e1-442c-bc07-7c6192ad6216  Pipeline: ingress
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "5b72d278-5b16-44a6-9aa0-9e513a429506"),
    action=(next;)
  table= 3(      ls_in_pre_acl), priority=  110,
    match=(ip && inport == "5b72d278-5b16-44a6-9aa0-9e513a429506
↪"),
    action=(next;)
  table= 9(      ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 192.168.1.1 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:0c:55:62;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:0c:55:62; arp.tpa = arp.spa;
            arp.spa = 192.168.1.1; outport = inport;
            inport = ""; /* Allow sending out inport. */ output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:fa:76:8f),
    action=(outport = "f112b99a-8ccc-4c52-8733-7593fa0966ea";␣
↪output;)
Datapath: 611d35e8-b1e1-442c-bc07-7c6192ad6216  Pipeline: egress
  table= 1(     ls_out_pre_acl), priority=  110,
    match=(ip && outport == "f112b99a-8ccc-4c52-8733-7593fa0966ea
↪"),
```

(continues on next page)

(continued from previous page)

```
    action=(next;)
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "f112b99a-8ccc-4c52-8733-7593fa0966ea"),
    action=(output;)
```

- Port bindings

```
_uuid                : 0f86395b-a0d8-40fd-b22c-4c9e238a7880
chassis              : []
datapath             : 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa
logical_port         : "lrp-5b72d278-5b16-44a6-9aa0-9e513a429506"
mac                  : []
options              : {peer="5b72d278-5b16-44a6-9aa0-9e513a429506
↪"}
parent_port          : []
tag                  : []
tunnel_key           : 1
type                 : patch

_uuid                : 8d95ab8c-c2ea-4231-9729-7ecbfc2cd676
chassis              : []
datapath             : 4aef86e4-e54a-4c83-bb27-d65c670d4b51
logical_port         : "5b72d278-5b16-44a6-9aa0-9e513a429506"
mac                  : ["fa:16:3e:0c:55:62 192.168.1.1"]
options              : {peer="lrp-5b72d278-5b16-44a6-9aa0-
↪9e513a429506"}
parent_port          : []
tag                  : []
tunnel_key           : 3
type                 : patch
```

- Multicast groups

```
_uuid                : 4a6191aa-d8ac-4e93-8306-b0d8fbbe4e35
datapath             : 4aef86e4-e54a-4c83-bb27-d65c670d4b51
name                 : _MC_flood
ports                : [8d95ab8c-c2ea-4231-9729-7ecbfc2cd676,
                        be71fac3-9f04-41c9-9951-f3f7f1fa1ec5,
                        da5c1269-90b7-4df2-8d76-d4575754b02d]
tunnel_key           : 65535
```

In addition, if the self-service network contains ports with IP addresses (typically instances or DHCP servers), OVN creates a logical flow for each port, similar to the following example.

```
Datapath: 4a7485c6-a1ef-46a5-b57c-5ddb6ac15aaa  Pipeline: ingress
  table= 5(  lr_in_arp_resolve), priority=  100,
    match=(outport == "lrp-f112b99a-8ccc-4c52-8733-7593fa0966ea" &&
           reg0 == 192.168.1.11),
    action=(eth.dst = fa:16:3e:b6:91:70; next;)
```

3. On each compute node, the OVN controller service creates patch ports, similar to the following example.

```
7(patch-f112b99a-): addr:4e:01:91:2a:73:66
    config:     0
```

(continues on next page)

(continued from previous page)

```
    state:      0
    speed: 0 Mbps now, 0 Mbps max
8(patch-lrp-f112b): addr:be:9d:7b:31:bb:87
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
```

4. On all compute nodes, the OVN controller service creates the following additional flows:

```
cookie=0x0, duration=6.667s, table=0, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,in_port=8
    actions=load:0x9->OXM_OF_METADATA[],load:0x1->NXM_NX_REG6[],
        resubmit(,16)
cookie=0x0, duration=6.667s, table=0, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,in_port=7
    actions=load:0x7->OXM_OF_METADATA[],load:0x4->NXM_NX_REG6[],
        resubmit(,16)
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x4,metadata=0x7
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x1,metadata=0x9,
        dl_dst=fa:16:3e:fa:76:8f
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x1,metadata=0x9,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x9,nw_src=192.168.1.1
    actions=drop
cookie=0x0, duration=6.673s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x9,nw_src=192.168.1.255
    actions=drop
cookie=0x0, duration=6.673s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,arp,reg6=0x1,metadata=0x9,
        arp_tpa=192.168.1.1,arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:fa:76:8f,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163efa768f->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80101->NXM_OF_ARP_SPA[],load:0x1->NXM_NX_REG7[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=6.673s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,icmp,metadata=0x9,nw_dst=192.168.1.1,
        icmp_type=8,icmp_code=0
    actions=move:NXM_OF_IP_SRC[]->NXM_OF_IP_DST[],mod_nw_src:192.168.
↪1.1,
        load:0xff->NXM_NX_IP_TTL[],load:0->NXM_OF_ICMP_TYPE[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,18)
cookie=0x0, duration=6.674s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=60,ip,metadata=0x9,nw_dst=192.168.1.1
    actions=drop
cookie=0x0, duration=6.674s, table=20, n_packets=0, n_bytes=0,
    idle_age=6, priority=24,ip,metadata=0x9,nw_dst=192.168.1.0/24
```

(continues on next page)

(continued from previous page)

```
    actions=dec_ttl(),move:NXM_OF_IP_DST[]->NXM_NX_REG0[],
        load:0xc0a80101->NXM_NX_REG1[],mod_dl_src:fa:16:3e:fa:76:8f,
        load:0x1->NXM_NX_REG7[],resubmit(,21)
cookie=0x0, duration=6.674s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg0=0xc0a80103,reg7=0x1,metadata=0x9
    actions=mod_dl_dst:fa:16:3e:d5:00:02,resubmit(,22)
cookie=0x0, duration=6.674s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg0=0xc0a80102,reg7=0x1,metadata=0x9
    actions=mod_dl_dst:fa:16:3e:82:8b:0e,resubmit(,22)
cookie=0x0, duration=6.673s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg0=0xc0a8010b,reg7=0x1,metadata=0x9
    actions=mod_dl_dst:fa:16:3e:b6:91:70,resubmit(,22)
cookie=0x0, duration=6.673s, table=25, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,arp,metadata=0x7,arp_tpa=192.168.1.1,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:fa:76:8f,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163efa768f->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80101->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=6.674s, table=26, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,metadata=0x7,dl_dst=fa:16:3e:fa:76:8f
    actions=load:0x4->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=6.667s, table=33, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x4,metadata=0x7
    actions=resubmit(,34)
cookie=0x0, duration=6.667s, table=33, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x1,metadata=0x9
    actions=resubmit(,34)
cookie=0x0, duration=6.667s, table=34, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg6=0x4,reg7=0x4,metadata=0x7
    actions=drop
cookie=0x0, duration=6.667s, table=34, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg6=0x1,reg7=0x1,metadata=0x9
    actions=drop
cookie=0x0, duration=6.674s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,ipv6,reg7=0x4,metadata=0x7
    actions=resubmit(,50)
cookie=0x0, duration=6.673s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,ip,reg7=0x4,metadata=0x7
    actions=resubmit(,50)
cookie=0x0, duration=6.673s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x1,metadata=0x9
    actions=resubmit(,64)
cookie=0x0, duration=6.673s, table=55, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg7=0x4,metadata=0x7
    actions=resubmit(,64)
cookie=0x0, duration=6.667s, table=64, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x4,metadata=0x7
    actions=output:7
cookie=0x0, duration=6.667s, table=64, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x1,metadata=0x9
    actions=output:8
```

5. On compute nodes not containing a port on the network, the OVN controller also creates additional flows.

```
cookie=0x0, duration=6.673s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x7,
        dl_src=01:00:00:00:00:00/01:00:00:00:00:00
    actions=drop
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x7,vlan_tci=0x1000/0x1000
    actions=drop
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x2,metadata=0x7
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=16, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg6=0x1,metadata=0x7
    actions=resubmit(,17)
cookie=0x0, duration=6.674s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,ip,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70,nw_src=192.168.1.11
    actions=resubmit(,18)
cookie=0x0, duration=6.674s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,udp,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70,nw_src=0.0.0.0,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=resubmit(,18)
cookie=0x0, duration=6.674s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,ip,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70
    actions=drop
cookie=0x0, duration=6.673s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,ipv6,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70
    actions=drop
cookie=0x0, duration=6.670s, table=17, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,18)
cookie=0x0, duration=6.674s, table=18, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,arp,reg6=0x3,metadata=0x7,
        dl_src=fa:16:3e:b6:91:70,arp_spa=192.168.1.11,
        arp_sha=fa:16:3e:b6:91:70
    actions=resubmit(,19)
cookie=0x0, duration=6.673s, table=18, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,icmp6,reg6=0x3,metadata=0x7,icmp_type=135,
        icmp_code=0
    actions=drop
cookie=0x0, duration=6.673s, table=18, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,icmp6,reg6=0x3,metadata=0x7,icmp_type=136,
        icmp_code=0
    actions=drop
cookie=0x0, duration=6.673s, table=18, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,arp,reg6=0x3,metadata=0x7
    actions=drop
cookie=0x0, duration=6.673s, table=18, n_packets=0, n_bytes=0,
```

(continues on next page)

(continued from previous page)

```
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,19)
cookie=0x0, duration=6.673s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,icmp6,metadata=0x7,icmp_type=136,icmp_
↪code=0
    actions=resubmit(,20)
cookie=0x0, duration=6.673s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,icmp6,metadata=0x7,icmp_type=135,icmp_
↪code=0
    actions=resubmit(,20)
cookie=0x0, duration=6.674s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=6.670s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=6.674s, table=19, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,20)
cookie=0x0, duration=6.673s, table=20, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,21)
cookie=0x0, duration=6.674s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x1/0x1,metadata=0x7
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=6.670s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x1/0x1,metadata=0x7
    actions=ct(table=22,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=6.674s, table=21, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,22)
cookie=0x0, duration=6.674s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=-new+est-rel-inv+trk,
↪metadata=0x7
    actions=resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=-new-est+rel-inv+trk,
↪metadata=0x7
    actions=resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=+inv+trk,metadata=0x7
    actions=drop
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,icmp6,metadata=0x7,icmp_type=135,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,icmp6,metadata=0x7,icmp_type=136,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=6.674s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,udp,reg6=0x3,metadata=0x7,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.674s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,udp,reg6=0x3,metadata=0x7,
```

(continues on next page)

(continued from previous page)

```
        nw_dst=192.168.1.0/24,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,ct_state=+new+trk,ipv6,reg6=0x3,
↪metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,ct_state=+new+trk,ip,reg6=0x3,
↪metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.674s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2001,ip,reg6=0x3,metadata=0x7
    actions=drop
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=2001,ipv6,reg6=0x3,metadata=0x7
    actions=drop
cookie=0x0, duration=6.674s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=1,ipv6,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=1,ip,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=6.673s, table=22, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,23)
cookie=0x0, duration=6.673s, table=23, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,24)
cookie=0x0, duration=6.674s, table=24, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x2/0x2,metadata=0x7
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=6.674s, table=24, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x2/0x2,metadata=0x7
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,25)
cookie=0x0, duration=6.673s, table=24, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x4/0x4,metadata=0x7
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=6.670s, table=24, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x4/0x4,metadata=0x7
    actions=ct(table=25,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=6.674s, table=24, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,25)
cookie=0x0, duration=6.673s, table=25, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,arp,metadata=0x7,arp_tpa=192.168.1.11,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:b6:91:70,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163eb69170->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a8010b->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=6.670s, table=25, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,arp,metadata=0x7,arp_tpa=192.168.1.3,arp_
↪op=1
```

(continues on next page)

(continued from previous page)

```
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:d5:00:02,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ed50002->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80103->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=6.670s, table=25, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,arp,metadata=0x7,arp_tpa=192.168.1.2,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:82:8b:0e,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163e828b0e->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80102->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=6.674s, table=25, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,26)
cookie=0x0, duration=6.674s, table=26, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x7,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=load:0xffff->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=6.674s, table=26, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,metadata=0x7,dl_dst=fa:16:3e:d5:00:02
    actions=load:0x2->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=6.673s, table=26, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,metadata=0x7,dl_dst=fa:16:3e:b6:91:70
    actions=load:0x3->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=6.670s, table=26, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,metadata=0x7,dl_dst=fa:16:3e:82:8b:0e
    actions=load:0x1->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=6.674s, table=32, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x3,metadata=0x7
    actions=load:0x7->NXM_NX_TUN_ID[0..23],
        set_field:0x3/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],output:3
cookie=0x0, duration=6.673s, table=32, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x2,metadata=0x7
    actions=load:0x7->NXM_NX_TUN_ID[0..23],
        set_field:0x2/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],output:3
cookie=0x0, duration=6.670s, table=32, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,reg7=0x1,metadata=0x7
    actions=load:0x7->NXM_NX_TUN_ID[0..23],
        set_field:0x1/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],output:5
cookie=0x0, duration=6.674s, table=48, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,49)
cookie=0x0, duration=6.674s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,icmp6,metadata=0x7,icmp_type=135,icmp_
↪code=0
```

(continues on next page)

(continued from previous page)

```
    actions=resubmit(,50)
cookie=0x0, duration=6.673s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=110,icmp6,metadata=0x7,icmp_type=136,icmp_
↪code=0
    actions=resubmit(,50)
cookie=0x0, duration=6.674s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=6.673s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=6.674s, table=49, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,50)
cookie=0x0, duration=6.674s, table=50, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x1/0x1,metadata=0x7
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=6.673s, table=50, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x1/0x1,metadata=0x7
    actions=ct(table=51,zone=NXM_NX_REG5[0..15])
cookie=0x0, duration=6.673s, table=50, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,51)
cookie=0x0, duration=6.670s, table=51, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,52)
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=+inv+trk,metadata=0x7
    actions=drop
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=-new+est-rel-inv+trk,
↪metadata=0x7
    actions=resubmit(,53)
cookie=0x0, duration=6.673s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,ct_state=-new-est+rel-inv+trk,
↪metadata=0x7
    actions=resubmit(,53)
cookie=0x0, duration=6.673s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,icmp6,metadata=0x7,icmp_type=136,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=6.673s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=65535,icmp6,metadata=0x7,icmp_type=135,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,ct_state=+new+trk,ip,reg7=0x3,
↪metadata=0x7,
        nw_src=192.168.1.11
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.670s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,ct_state=+new+trk,ip,reg7=0x3,
↪metadata=0x7,
        nw_src=192.168.1.11
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.670s, table=52, n_packets=0, n_bytes=0,
```

(continues on next page)

(continued from previous page)

```
    idle_age=6, priority=2002,udp,reg7=0x3,metadata=0x7,
        nw_src=192.168.1.0/24,tp_src=67,tp_dst=68
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.670s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=2002,ct_state=+new+trk,ipv6,reg7=0x3,
        metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.673s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=2001,ip,reg7=0x3,metadata=0x7
    actions=drop
cookie=0x0, duration=6.673s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=2001,ipv6,reg7=0x3,metadata=0x7
    actions=drop
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=1,ip,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=1,ipv6,metadata=0x7
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=6.674s, table=52, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,53)
cookie=0x0, duration=6.674s, table=53, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x4/0x4,metadata=0x7
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=6.674s, table=53, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x4/0x4,metadata=0x7
    actions=ct(table=54,zone=NXM_NX_REG5[0..15],nat)
cookie=0x0, duration=6.673s, table=53, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ipv6,reg0=0x2/0x2,metadata=0x7
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=6.673s, table=53, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,ip,reg0=0x2/0x2,metadata=0x7
    actions=ct(commit,zone=NXM_NX_REG5[0..15]),resubmit(,54)
cookie=0x0, duration=6.674s, table=53, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,54)
cookie=0x0, duration=6.674s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,ip,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70,nw_dst=255.255.255.255
    actions=resubmit(,55)
cookie=0x0, duration=6.673s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,ip,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70,nw_dst=192.168.1.11
    actions=resubmit(,55)
cookie=0x0, duration=6.673s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=90,ip,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70,nw_dst=224.0.0.0/4
    actions=resubmit(,55)
cookie=0x0, duration=6.670s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,ip,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70
    actions=drop
cookie=0x0, duration=6.670s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=80,ipv6,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70
```

(continues on next page)

(continued from previous page)

```
    actions=drop
cookie=0x0, duration=6.674s, table=54, n_packets=0, n_bytes=0,
    idle_age=6, priority=0,metadata=0x7
    actions=resubmit(,55)
cookie=0x0, duration=6.673s, table=55, n_packets=0, n_bytes=0,
    idle_age=6, priority=100,metadata=0x7,
        dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
    actions=resubmit(,64)
cookie=0x0, duration=6.674s, table=55, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:b6:91:70
    actions=resubmit(,64)
cookie=0x0, duration=6.673s, table=55, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg7=0x1,metadata=0x7
    actions=resubmit(,64)
cookie=0x0, duration=6.670s, table=55, n_packets=0, n_bytes=0,
    idle_age=6, priority=50,reg7=0x2,metadata=0x7
    actions=resubmit(,64)
```

6. On compute nodes containing a port on the network, the OVN controller also creates an additional flow.

```
cookie=0x0, duration=13.358s, table=52, n_packets=0, n_bytes=0,
    idle_age=13, priority=2002,ct_state=+new+trk,ipv6,reg7=0x3,
        metadata=0x7,ipv6_src=::
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
```

Instances

Launching an instance causes the same series of operations regardless of the network. The following example uses the `provider` provider network, `cirros` image, `m1.tiny` flavor, `default` security group, and `mykey` key.

Launch an instance on a provider network

1. On the controller node, source the credentials for a regular (non-privileged) project. The following example uses the `demo` project.

2. On the controller node, launch an instance using the UUID of the provider network.

```
$ openstack server create --flavor m1.tiny --image cirros \
  --nic net-id=0243277b-4aa8-46d8-9e10-5c9ad5e01521 \
  --security-group default --key-name mykey provider-instance
+--------------------------------------+------------------------------
↪-----------------+
| Property                             | Value                        ␣
↪                 |
+--------------------------------------+------------------------------
↪-----------------+
| OS-DCF:diskConfig                    | MANUAL                       ␣
↪                 |
| OS-EXT-AZ:availability_zone          | nova                         ␣
↪                 |
```

(continues on next page)

(continued from previous page)

```
| 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                            | hdF4LMQqC5PB                  ␣
↪                  |
| config_drive                         |                               ␣
↪                  |
| created                              | 2015-09-17T21:58:18Z          ␣
↪                  |
| flavor                               | m1.tiny (1)                   ␣
↪                  |
| hostId                               |                               ␣
↪                  |
| id                                   | 181c52ba-aebc-4c32-a97d-
↪2e8e82e4eaaf          |
| image                                | cirros (38047887-61a7-41ea-
↪9b49-27987d5e8bb9) |
| key_name                             | mykey                         ␣
↪                  |
| metadata                             | {}                            ␣
↪                  |
| name                                 | provider-instance             ␣
↪                  |
| os-extended-volumes:volumes_attached | []                            ␣
↪                  |
| progress                             | 0                             ␣
↪                  |
| security_groups                      | default                       ␣
↪                  |
| status                               | BUILD                         ␣
↪                  |
| tenant_id                            |␣
↪f5b2ccaa75ac413591f12fcaa096aa5c         |
| updated                              | 2015-09-17T21:58:18Z          ␣
↪                  |
| user_id                              |␣
↪684286a9079845359882afc3aa5011fb         |
+--------------------------------------+------------------------------
↪-----------------+
```

OVN operations

The OVN mechanism driver and OVN perform the following operations when launching an instance.

1. The OVN mechanism driver creates a logical port for the instance.

```
_uuid                 : cc891503-1259-47a1-9349-1c0293876664
addresses             : ["fa:16:3e:1c:ca:6a 203.0.113.103"]
enabled               : true
external_ids          : {"neutron:port_name"=""}
name                  : "cafd4862-c69c-46e4-b3d2-6141ce06b205"
options               : {}
parent_name           : []
port_security         : ["fa:16:3e:1c:ca:6a 203.0.113.103"]
tag                   : []
type                  : ""
up                    : true
```

2. The OVN mechanism driver updates the appropriate Address Set entry with the address of this instance:

```
_uuid                 : d0becdea-e1ed-48c4-9afc-e278cdef4629
addresses             : ["203.0.113.103"]
external_ids          : {"neutron:security_group_name"=default}
name                  : "as_ip4_90a78a43_b549_4bee_8822_21fcccab58dc"
```

3. The OVN mechanism driver creates ACL entries for this port and any other ports in the project.

```
_uuid                 : f8d27bfc-4d74-4e73-8fac-c84585443efd
action                : drop
direction             : from-lport
external_ids          : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                   : false
match                 : "inport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip"
priority              : 1001

_uuid                 : a61d0068-b1aa-4900-9882-e0671d1fc131
action                : allow
direction             : to-lport
external_ids          : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                   : false
match                 : "outport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip4 && ip4.src == 203.0.113.0/24 && udp && udp.
↪src == 67 && udp.dst == 68"
priority              : 1002

_uuid                 : a5a787b8-7040-4b63-a20a-551bd73eb3d1
action                : allow-related
direction             : from-lport
external_ids          : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                   : false
match                 : "inport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip6"
```

(continues on next page)

(continued from previous page)

```
priority            : 1002

_uuid               : 7b3f63b8-e69a-476c-ad3d-37de043232b2
action              : allow-related
direction           : to-lport
external_ids        : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                 : false
match               : "outport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip4 && ip4.src = $as_ip4_90a78a43_b5649_4bee_8822_
↪21fcccab58dc"
priority            : 1002

_uuid               : 36dbb1b1-cd30-4454-a0bf-923646eb7c3f
action              : allow
direction           : from-lport
external_ids        : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                 : false
match               : "inport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip4 && (ip4.dst == 255.255.255.255 || ip4.dst ==␣
↪203.0.113.0/24) && udp && udp.src == 68 && udp.dst == 67"
priority            : 1002

_uuid               : 05a92f66-be48-461e-a7f1-b07bfbd3e667
action              : allow-related
direction           : from-lport
external_ids        : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                 : false
match               : "inport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip4"
priority            : 1002

_uuid               : 37f18377-d6c3-4c44-9e4d-2170710e50ff
action              : drop
direction           : to-lport
external_ids        : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                 : false
match               : "outport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip"
priority            : 1001

_uuid               : 6d4db3cf-c1f1-4006-ad66-ae582a6acd21
action              : allow-related
direction           : to-lport
external_ids        : {"neutron:lport"="cafd4862-c69c-46e4-b3d2-
↪6141ce06b205"}
log                 : false
match               : "outport == \"cafd4862-c69c-46e4-b3d2-
↪6141ce06b205\" && ip6 && ip6.src = $as_ip6_90a78a43_b5649_4bee_8822_
↪21fcccab58dc"
priority            : 1002
```

4. The OVN mechanism driver updates the logical switch information with the UUIDs of these

objects.

```
_uuid                : 924500c4-8580-4d5f-a7ad-8769f6e58ff5
acls                 : [05a92f66-be48-461e-a7f1-b07bfbd3e667,
                        36dbb1b1-cd30-4454-a0bf-923646eb7c3f,
                        37f18377-d6c3-4c44-9e4d-2170710e50ff,
                        7b3f63b8-e69a-476c-ad3d-37de043232b2,
                        a5a787b8-7040-4b63-a20a-551bd73eb3d1,
                        a61d0068-b1aa-4900-9882-e0671d1fc131,
                        f8d27bfc-4d74-4e73-8fac-c84585443efd]
external_ids         : {"neutron:network_name"=provider}
name                 : "neutron-670efade-7cd0-4d87-8a04-27f366eb8941"
ports                : [38cf8b52-47c4-4e93-be8d-06bf71f6a7c9,
                        5e144ab9-3e08-4910-b936-869bbbf254c8,
                        a576b812-9c3e-4cfb-9752-5d8500b3adf9,
                        cc891503-1259-47a1-9349-1c0293876664]
```

5. The OVN northbound service creates port bindings for the logical ports and adds them to the appropriate multicast group.

   - Port bindings

```
_uuid                : e73e3fcd-316a-4418-bbd5-a8a42032b1c3
chassis              : fc5ab9e7-bc28-40e8-ad52-2949358cc088
datapath             : bd0ab2b3-4cf4-4289-9529-ef430f6a89e6
logical_port         : "cafd4862-c69c-46e4-b3d2-6141ce06b205"
mac                  : ["fa:16:3e:1c:ca:6a 203.0.113.103"]
options              : {}
parent_port          : []
tag                  : []
tunnel_key           : 4
type                 : ""
```

   - Multicast groups

```
_uuid                : 39b32ccd-fa49-4046-9527-13318842461e
datapath             : bd0ab2b3-4cf4-4289-9529-ef430f6a89e6
name                 : _MC_flood
ports                : [030024f4-61c3-4807-859b-07727447c427,
                        904c3108-234d-41c0-b93c-116b7e352a75,
                        cc5bcd19-bcae-4e29-8cee-3ec8a8a75d46,
                        e73e3fcd-316a-4418-bbd5-a8a42032b1c3]
tunnel_key           : 65535
```

6. The OVN northbound service translates the Address Set change into the new Address Set in the OVN southbound database.

```
_uuid                : 2addbee3-7084-4fff-8f7b-15b1efebdaff
addresses            : ["203.0.113.103"]
name                 : "as_ip4_90a78a43_b549_4bee_8822_21fcccab58dc"
```

7. The OVN northbound service translates the ACL and logical port objects into logical flows in the OVN southbound database.

```
Datapath: bd0ab2b3-4cf4-4289-9529-ef430f6a89e6  Pipeline: ingress
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
```

(continues on next page)

(continued from previous page)

```
          eth.src == {fa:16:3e:1c:ca:6a}),
    action=(next;)
  table= 1(  ls_in_port_sec_ip), priority=   90,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
           eth.src == fa:16:3e:1c:ca:6a && ip4.src == {203.0.113.103}
↪),
    action=(next;)
  table= 1(  ls_in_port_sec_ip), priority=   90,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
           eth.src == fa:16:3e:1c:ca:6a && ip4.src == 0.0.0.0 &&
           ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst ==␣
↪67),
    action=(next;)
  table= 1(  ls_in_port_sec_ip), priority=   80,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
           eth.src == fa:16:3e:1c:ca:6a && ip),
    action=(drop;)
  table= 2(  ls_in_port_sec_nd), priority=   90,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
           eth.src == fa:16:3e:1c:ca:6a &&
           arp.sha == fa:16:3e:1c:ca:6a && (arp.spa == 203.0.113.103␣
↪)),
    action=(next;)
  table= 2(  ls_in_port_sec_nd), priority=   80,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
           (arp || nd)),
    action=(drop;)
  table= 3(      ls_in_pre_acl), priority=  110,
    match=(nd),
    action=(next;)
  table= 3(      ls_in_pre_acl), priority=  100,
    match=(ip),
    action=(reg0[0] = 1; next;)
  table= 6(          ls_in_acl), priority=65535,
    match=(ct.inv),
    action=(drop;)
  table= 6(          ls_in_acl), priority=65535,
    match=(nd),
    action=(next;)
  table= 6(          ls_in_acl), priority=65535,
    match=(ct.est && !ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 6(          ls_in_acl), priority=65535,
    match=(!ct.est && ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 6(          ls_in_acl), priority= 2002,
    match=(ct.new && (inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205"
           && ip6)),
    action=(reg0[1] = 1; next;)
  table= 6(          ls_in_acl), priority= 2002,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip4 &&
           (ip4.dst == 255.255.255.255 || ip4.dst == 203.0.113.0/24) &
↪&
           udp && udp.src == 68 && udp.dst == 67),
    action=(reg0[1] = 1; next;)
  table= 6(          ls_in_acl), priority= 2002,
```

(continues on next page)

(continued from previous page)

```
    match=(ct.new && (inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205
↪" &&
           ip4)),
    action=(reg0[1] = 1; next;)
  table= 6(           ls_in_acl), priority= 2001,
    match=(inport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip),
    action=(drop;)
  table= 6(           ls_in_acl), priority=    1,
    match=(ip),
    action=(reg0[1] = 1; next;)
  table= 9(       ls_in_arp_rsp), priority=   50,
    match=(arp.tpa == 203.0.113.103 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:1c:ca:6a;
            arp.op = 2; /* ARP reply */ arp.tha = arp.sha;
            arp.sha = fa:16:3e:1c:ca:6a; arp.tpa = arp.spa;
            arp.spa = 203.0.113.103; outport = inport;
            inport = ""; /* Allow sending out inport. */ output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:1c:ca:6a),
    action=(outport = "cafd4862-c69c-46e4-b3d2-6141ce06b205"; output;)
Datapath: bd0ab2b3-4cf4-4289-9529-ef430f6a89e6  Pipeline: egress
  table= 1(     ls_out_pre_acl), priority=  110,
    match=(nd),
    action=(next;)
  table= 1(     ls_out_pre_acl), priority=  100,
    match=(ip),
    action=(reg0[0] = 1; next;)
  table= 4(          ls_out_acl), priority=65535,
    match=(!ct.est && ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 4(          ls_out_acl), priority=65535,
    match=(ct.est && !ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 4(          ls_out_acl), priority=65535,
    match=(ct.inv),
    action=(drop;)
  table= 4(          ls_out_acl), priority=65535,
    match=(nd),
    action=(next;)
  table= 4(          ls_out_acl), priority= 2002,
    match=(ct.new &&
           (outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip6 &
↪&
            ip6.src == $as_ip6_90a78a43_b549_4bee_8822_21fccab58dc)),
    action=(reg0[1] = 1; next;)
  table= 4(          ls_out_acl), priority= 2002,
    match=(ct.new &&
           (outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip4 &
↪&
            ip4.src == $as_ip4_90a78a43_b549_4bee_8822_21fccab58dc)),
    action=(reg0[1] = 1; next;)
  table= 4(          ls_out_acl), priority= 2002,
    match=(outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip4 &&
           ip4.src == 203.0.113.0/24 && udp && udp.src == 67 &&
           udp.dst == 68),
    action=(reg0[1] = 1; next;)
```

(continues on next page)

(continued from previous page)

```
table= 4(        ls_out_acl), priority= 2001,
  match=(outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" && ip),
  action=(drop;)
table= 4(        ls_out_acl), priority=    1,
  match=(ip),
  action=(reg0[1] = 1; next;)
table= 6( ls_out_port_sec_ip), priority=   90,
  match=(outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
         eth.dst == fa:16:3e:1c:ca:6a &&
         ip4.dst == {255.255.255.255, 224.0.0.0/4, 203.0.113.103}),
  action=(next;)
table= 6( ls_out_port_sec_ip), priority=   80,
  match=(outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
         eth.dst == fa:16:3e:1c:ca:6a && ip),
  action=(drop;)
table= 7( ls_out_port_sec_l2), priority=   50,
  match=(outport == "cafd4862-c69c-46e4-b3d2-6141ce06b205" &&
         eth.dst == {fa:16:3e:1c:ca:6a}),
  action=(output;)
```

8. The OVN controller service on each compute node translates these objects into flows on the integration bridge `br-int`. Exact flows depend on whether the compute node containing the instance also contains a DHCP agent on the subnet.

   - On the compute node containing the instance, the Compute service creates a port that connects the instance to the integration bridge and OVN creates the following flows:

```
# ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:000022024a1dc045
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_
↪MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_
↪dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src␣
↪mod_tp_dst
 9(tapcafd4862-c6): addr:fe:16:3e:1c:ca:6a
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
```

```
cookie=0x0, duration=184.992s, table=0, n_packets=175, n_
↪bytes=15270,
    idle_age=15, priority=100,in_port=9
    actions=load:0x3->NXM_NX_REG5[],load:0x4->OXM_OF_METADATA[],
        load:0x4->NXM_NX_REG6[],resubmit(,16)
cookie=0x0, duration=191.687s, table=16, n_packets=175, n_
↪bytes=15270,
    idle_age=15, priority=50,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=resubmit(,17)
cookie=0x0, duration=191.687s, table=17, n_packets=2, n_bytes=684,
    idle_age=112, priority=90,udp,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a,nw_src=0.0.0.0,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
```

(continues on next page)

(continued from previous page)

```
    actions=resubmit(,18)
cookie=0x0, duration=191.687s, table=17, n_packets=146, n_
↪bytes=12780,
    idle_age=20, priority=90,ip,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a,nw_src=203.0.113.103
    actions=resubmit(,18)
cookie=0x0, duration=191.687s, table=17, n_packets=17, n_
↪bytes=1386,
    idle_age=92, priority=80,ipv6,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=191.687s, table=17, n_packets=0, n_bytes=0,
    idle_age=191, priority=80,ip,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=191.687s, table=18, n_packets=10, n_
↪bytes=420,
    idle_age=15, priority=90,arp,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a,arp_spa=203.0.113.103,
        arp_sha=fa:16:3e:1c:ca:6a
    actions=resubmit(,19)
cookie=0x0, duration=191.687s, table=18, n_packets=0, n_bytes=0,
    idle_age=191, priority=80,icmp6,reg6=0x4,metadata=0x4,
        icmp_type=136,icmp_code=0
    actions=drop
cookie=0x0, duration=191.687s, table=18, n_packets=0, n_bytes=0,
    idle_age=191, priority=80,icmp6,reg6=0x4,metadata=0x4,
        icmp_type=135,icmp_code=0
    actions=drop
cookie=0x0, duration=191.687s, table=18, n_packets=0, n_bytes=0,
    idle_age=191, priority=80,arp,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=75.033s, table=19, n_packets=0, n_bytes=0,
    idle_age=75, priority=110,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=75.032s, table=19, n_packets=0, n_bytes=0,
    idle_age=75, priority=110,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=75.032s, table=19, n_packets=34, n_
↪bytes=5170,
    idle_age=49, priority=100,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=75.032s, table=19, n_packets=0, n_bytes=0,
    idle_age=75, priority=100,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=13, n_
↪bytes=1118,
```

(continues on next page)

(continued from previous page)

```
    idle_age=49, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x4
    actions=resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x4
    actions=resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,ct_state=+inv+trk,metadata=0x4
    actions=drop
cookie=0x0, duration=75.033s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=2002,ct_state=+new+trk,ipv6,reg6=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=15, n_
↪bytes=1816,
    idle_age=49, priority=2002,ct_state=+new+trk,ip,reg6=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=2002,udp,reg6=0x4,metadata=0x4,
        nw_dst=203.0.113.0/24,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=2002,udp,reg6=0x4,metadata=0x4,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=75.033s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=2001,ip,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=2001,ipv6,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=75.032s, table=22, n_packets=6, n_bytes=2236,
    idle_age=54, priority=1,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=75.032s, table=22, n_packets=0, n_bytes=0,
    idle_age=75, priority=1,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=67.064s, table=25, n_packets=0, n_bytes=0,
    idle_age=67, priority=50,arp,metadata=0x4,arp_tpa=203.0.113.
↪103,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:1c:ca:6a,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ed63dca->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a81268->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=75.033s, table=26, n_packets=19, n_
↪bytes=2776,
    idle_age=44, priority=50,metadata=0x4,dl_dst=fa:16:3e:1c:ca:6a
    actions=load:0x4->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=221031.310s, table=33, n_packets=72, n_
↪bytes=6292,
```

(continues on next page)

(continued from previous page)

```
    idle_age=20, hard_age=65534, priority=100,reg7=0x3,
↪metadata=0x4
    actions=load:0x1->NXM_NX_REG7[],resubmit(,33)
cookie=0x0, duration=184.992s, table=34, n_packets=2, n_bytes=684,
    idle_age=112, priority=100,reg6=0x4,reg7=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=75.034s, table=49, n_packets=0, n_bytes=0,
    idle_age=75, priority=110,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=75.033s, table=49, n_packets=0, n_bytes=0,
    idle_age=75, priority=110,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=75.033s, table=49, n_packets=38, n_
↪bytes=6566,
    idle_age=49, priority=100,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=75.033s, table=49, n_packets=0, n_bytes=0,
    idle_age=75, priority=100,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x4
    actions=resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=13, n_
↪bytes=1118,
    idle_age=49, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x4
    actions=resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=65535,ct_state=+inv+trk,metadata=0x4
    actions=drop
cookie=0x0, duration=75.034s, table=52, n_packets=4, n_bytes=1538,
    idle_age=54, priority=2002,udp,reg7=0x4,metadata=0x4,
        nw_src=203.0.113.0/24,tp_src=67,tp_dst=68
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=2002,ct_state=+new+trk,ip,reg7=0x4,
        metadata=0x4,nw_src=203.0.113.103
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=2.041s, table=52, n_packets=0, n_bytes=0,
    idle_age=2, priority=2002,ct_state=+new+trk,ipv6,reg7=0x4,
        metadata=0x4,ipv6_src=::2/::2
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=2, n_bytes=698,
    idle_age=54, priority=2001,ip,reg7=0x4,metadata=0x4
    actions=drop
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=75.033s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=2001,ipv6,reg7=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=75.034s, table=52, n_packets=0, n_bytes=0,
    idle_age=75, priority=1,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=75.033s, table=52, n_packets=19, n_
↪bytes=3212,
    idle_age=49, priority=1,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=75.034s, table=54, n_packets=17, n_
↪bytes=2656,
    idle_age=49, priority=90,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=203.0.113.103
    actions=resubmit(,55)
cookie=0x0, duration=75.033s, table=54, n_packets=0, n_bytes=0,
    idle_age=75, priority=90,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=255.255.255.255
    actions=resubmit(,55)
cookie=0x0, duration=75.033s, table=54, n_packets=0, n_bytes=0,
    idle_age=75, priority=90,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=224.0.0.0/4
    actions=resubmit(,55)
cookie=0x0, duration=75.033s, table=54, n_packets=0, n_bytes=0,
    idle_age=75, priority=80,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=75.033s, table=54, n_packets=0, n_bytes=0,
    idle_age=75, priority=80,ipv6,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=75.033s, table=55, n_packets=21, n_
↪bytes=2860,
    idle_age=44, priority=50,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=resubmit(,64)
cookie=0x0, duration=184.992s, table=64, n_packets=166, n_
↪bytes=15088,
    idle_age=15, priority=100,reg7=0x4,metadata=0x4
    actions=output:9
```

- For each compute node that only contains a DHCP agent on the subnet, OVN creates the following flows:

```
cookie=0x0, duration=189.649s, table=16, n_packets=0, n_bytes=0,
    idle_age=189, priority=50,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=resubmit(,17)
cookie=0x0, duration=189.650s, table=17, n_packets=0, n_bytes=0,
    idle_age=189, priority=90,udp,reg6=0x4,metadata=0x4,
        dl_src=fa:14:3e:1c:ca:6a,nw_src=0.0.0.0,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=resubmit(,18)
cookie=0x0, duration=189.649s, table=17, n_packets=0, n_bytes=0,
    idle_age=189, priority=90,ip,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a,nw_src=203.0.113.103
```

(continues on next page)

(continued from previous page)

```
    actions=resubmit(,18)
cookie=0x0, duration=189.650s, table=17, n_packets=0, n_bytes=0,
    idle_age=189, priority=80,ipv6,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=189.650s, table=17, n_packets=0, n_bytes=0,
    idle_age=189, priority=80,ip,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=189.650s, table=18, n_packets=0, n_bytes=0,
    idle_age=189, priority=90,arp,reg6=0x4,metadata=0x4,
        dl_src=fa:16:3e:1c:ca:6a,arp_spa=203.0.113.103,
        arp_sha=fa:16:3e:1c:ca:6a
    actions=resubmit(,19)
cookie=0x0, duration=189.650s, table=18, n_packets=0, n_bytes=0,
    idle_age=189, priority=80,icmp6,reg6=0x4,metadata=0x4,
        icmp_type=136,icmp_code=0
    actions=drop
cookie=0x0, duration=189.650s, table=18, n_packets=0, n_bytes=0,
    idle_age=189, priority=80,icmp6,reg6=0x4,metadata=0x4,
        icmp_type=135,icmp_code=0
    actions=drop
cookie=0x0, duration=189.649s, table=18, n_packets=0, n_bytes=0,
    idle_age=189, priority=80,arp,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=79.452s, table=19, n_packets=0, n_bytes=0,
    idle_age=79, priority=110,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=79.450s, table=19, n_packets=0, n_bytes=0,
    idle_age=79, priority=110,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=79.452s, table=19, n_packets=0, n_bytes=0,
    idle_age=79, priority=100,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=79.450s, table=19, n_packets=18, n_
↪bytes=3164,
    idle_age=57, priority=100,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=79.450s, table=22, n_packets=6, n_bytes=510,
    idle_age=57, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x4
    actions=resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x4
    actions=resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,23)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,ct_state=+inv+trk,metadata=0x4
    actions=drop
cookie=0x0, duration=79.453s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,ct_state=+new+trk,ipv6,reg6=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,ct_state=+new+trk,ip,reg6=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,udp,reg6=0x4,metadata=0x4,
        nw_dst=203.0.113.0/24,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,udp,reg6=0x4,metadata=0x4,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=79.452s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2001,ip,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=2001,ipv6,reg6=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=79.450s, table=22, n_packets=0, n_bytes=0,
    idle_age=79, priority=1,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=79.450s, table=22, n_packets=12, n_
↪bytes=2654,
    idle_age=57, priority=1,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=71.483s, table=25, n_packets=0, n_bytes=0,
    idle_age=71, priority=50,arp,metadata=0x4,arp_tpa=203.0.113.
↪103,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:1c:ca:6a,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163ed63dca->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a81268->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],load:0->NXM_NX_REG6[],
        load:0->NXM_OF_IN_PORT[],resubmit(,32)
cookie=0x0, duration=79.450s, table=26, n_packets=8, n_bytes=1258,
    idle_age=57, priority=50,metadata=0x4,dl_dst=fa:16:3e:1c:ca:6a
    actions=load:0x4->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=182.952s, table=33, n_packets=74, n_
↪bytes=7040,
    idle_age=18, priority=100,reg7=0x4,metadata=0x4
    actions=load:0x1->NXM_NX_REG7[],resubmit(,33)
cookie=0x0, duration=79.451s, table=49, n_packets=0, n_bytes=0,
    idle_age=79, priority=110,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=79.450s, table=49, n_packets=0, n_bytes=0,
```

(continues on next page)

(continued from previous page)

```
    idle_age=79, priority=110,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=79.450s, table=49, n_packets=18, n_
↪bytes=3164,
    idle_age=57, priority=100,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=79.450s, table=49, n_packets=0, n_bytes=0,
    idle_age=79, priority=100,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x4
    actions=resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=6, n_bytes=510,
    idle_age=57, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x4
    actions=resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,icmp6,metadata=0x4,icmp_type=135,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,icmp6,metadata=0x4,icmp_type=136,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=65535,ct_state=+inv+trk,metadata=0x4
    actions=drop
cookie=0x0, duration=79.452s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,udp,reg7=0x4,metadata=0x4,
        nw_src=203.0.113.0/24,tp_src=67,tp_dst=68
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=2002,ct_state=+new+trk,ip,reg7=0x4,
        metadata=0x4,nw_src=203.0.113.103
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=71.483s, table=52, n_packets=0, n_bytes=0,
    idle_age=71, priority=2002,ct_state=+new+trk,ipv6,reg7=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=2001,ipv6,reg7=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=79.450s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=2001,ip,reg7=0x4,metadata=0x4
    actions=drop
cookie=0x0, duration=79.453s, table=52, n_packets=0, n_bytes=0,
    idle_age=79, priority=1,ipv6,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=79.450s, table=52, n_packets=12, n_
↪bytes=2654,
    idle_age=57, priority=1,ip,metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=79.452s, table=54, n_packets=0, n_bytes=0,
    idle_age=79, priority=90,ip,reg7=0x4,metadata=0x4,
```

(continues on next page)

(continued from previous page)

```
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=255.255.255.255
    actions=resubmit(,55)
cookie=0x0, duration=79.452s, table=54, n_packets=0, n_bytes=0,
    idle_age=79, priority=90,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=203.0.113.103
    actions=resubmit(,55)
cookie=0x0, duration=79.452s, table=54, n_packets=0, n_bytes=0,
    idle_age=79, priority=90,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a,nw_dst=224.0.0.0/4
    actions=resubmit(,55)
cookie=0x0, duration=79.450s, table=54, n_packets=0, n_bytes=0,
    idle_age=79, priority=80,ip,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=79.450s, table=54, n_packets=0, n_bytes=0,
    idle_age=79, priority=80,ipv6,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=drop
cookie=0x0, duration=79.450s, table=55, n_packets=0, n_bytes=0,
    idle_age=79, priority=50,reg7=0x4,metadata=0x4,
        dl_dst=fa:16:3e:1c:ca:6a
    actions=resubmit(,64)
```

Launch an instance on a self-service network

To launch an instance on a self-service network, follow the same steps as *launching an instance on the provider network*, but using the UUID of the self-service network.

OVN operations

The OVN mechanism driver and OVN perform the following operations when launching an instance.

1. The OVN mechanism driver creates a logical port for the instance.

```
_uuid                : c754d1d2-a7fb-4dd0-b14c-c076962b06b9
addresses            : ["fa:16:3e:15:7d:13 192.168.1.5"]
enabled              : true
external_ids         : {"neutron:port_name"=""}
name                 : "eaf36f62-5629-4ec4-b8b9-5e562c40e7ae"
options              : {}
parent_name          : []
port_security        : ["fa:16:3e:15:7d:13 192.168.1.5"]
tag                  : []
type                 : ""
up                   : true
```

2. The OVN mechanism driver updates the appropriate Address Set object(s) with the address of the new instance:

```
_uuid                : d0becdea-e1ed-48c4-9afc-e278cdef4629
addresses            : ["192.168.1.5", "203.0.113.103"]
external_ids         : {"neutron:security_group_name"=default}
name                 : "as_ip4_90a78a43_b549_4bee_8822_21fcccab58dc"
```

3. The OVN mechanism driver creates ACL entries for this port and any other ports in the project.

```
_uuid                : 00ecbe8f-c82a-4e18-b688-af2a1941cff7
action               : allow
direction            : from-lport
external_ids         : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                  : false
match                : "inport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip4 && (ip4.dst == 255.255.255.255 || ip4.dst ==␣
↪192.168.1.0/24) && udp && udp.src == 68 && udp.dst == 67"
priority             : 1002

_uuid                : 2bf5b7ed-008e-4676-bba5-71fe58897886
action               : allow-related
direction            : from-lport
external_ids         : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                  : false
match                : "inport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip4"
priority             : 1002

_uuid                : 330b4e27-074f-446a-849b-9ab0018b65c5
action               : allow
direction            : to-lport
external_ids         : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                  : false
match                : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip4 && ip4.src == 192.168.1.0/24 && udp && udp.
↪src == 67 && udp.dst == 68"
priority             : 1002

_uuid                : 683f52f2-4be6-4bd7-a195-6c782daa7840
action               : allow-related
direction            : from-lport
external_ids         : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                  : false
match                : "inport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip6"
priority             : 1002

_uuid                : 8160f0b4-b344-43d5-bbd4-ca63a71aa4fc
action               : drop
direction            : to-lport
external_ids         : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                  : false
match                : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip"
priority             : 1001

_uuid                : 97c6b8ca-14ea-4812-8571-95d640a88f4f
action               : allow-related
direction            : to-lport
```

(continues on next page)

(continued from previous page)

```
external_ids        : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                 : false
match               : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip6"
priority            : 1002

_uuid               : 9cfd8eb5-5daa-422e-8fe8-bd22fd7fa826
action              : allow-related
direction           : to-lport
external_ids        : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                 : false
match               : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip4 && ip4.src == 0.0.0.0/0 && icmp4"
priority            : 1002

_uuid               : f72c2431-7a64-4cea-b84a-118bdc761be2
action              : drop
direction           : from-lport
external_ids        : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                 : false
match               : "inport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip"
priority            : 1001

_uuid               : f94133fa-ed27-4d5e-a806-0d528e539cb3
action              : allow-related
direction           : to-lport
external_ids        : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                 : false
match               : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip4 && ip4.src == $as_ip4_90a78a43_b549_4bee_8822_
↪21fcccab58dc"
priority            : 1002

_uuid               : 7f7a92ff-b7e9-49b0-8be0-0dc388035df3
action              : allow-related
direction           : to-lport
external_ids        : {"neutron:lport"="eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae"}
log                 : false
match               : "outport == \"eaf36f62-5629-4ec4-b8b9-
↪5e562c40e7ae\" && ip6 && ip6.src == $as_ip4_90a78a43_b549_4bee_8822_
↪21fcccab58dc"
priority            : 1002
```

4. The OVN mechanism driver updates the logical switch information with the UUIDs of these objects.

```
_uuid               : 15e2c80b-1461-4003-9869-80416cd97de5
acls                : [00ecbe8f-c82a-4e18-b688-af2a1941cff7,
                       2bf5b7ed-008e-4676-bba5-71fe58897886,
                       330b4e27-074f-446a-849b-9ab0018b65c5,
```

(continues on next page)

(continued from previous page)

```
                        683f52f2-4be6-4bd7-a195-6c782daa7840,
                        7f7a92ff-b7e9-49b0-8be0-0dc388035df3,
                        8160f0b4-b344-43d5-bbd4-ca63a71aa4fc,
                        97c6b8ca-14ea-4812-8571-95d640a88f4f,
                        9cfd8eb5-5daa-422e-8fe8-bd22fd7fa826,
                        f72c2431-7a64-4cea-b84a-118bdc761be2,
                        f94133fa-ed27-4d5e-a806-0d528e539cb3]
external_ids        : {"neutron:network_name"="selfservice"}
name                : "neutron-6cc81cae-8c5f-4c09-aaf2-35d0aa95c084"
ports               : [2df457a5-f71c-4a2f-b9ab-d9e488653872,
                        67c2737c-b380-492b-883b-438048b48e56,
                        c754d1d2-a7fb-4dd0-b14c-c076962b06b9]
```

5. With address sets, it is no longer necessary for the OVN mechanism driver to create separate ACLs for other instances in the project. That is handled automagically via address sets.

6. The OVN northbound service translates the updated Address Set object(s) into updated Address Set objects in the OVN southbound database:

```
_uuid               : 2addbee3-7084-4fff-8f7b-15b1efebdaff
addresses           : ["192.168.1.5", "203.0.113.103"]
name                : "as_ip4_90a78a43_b549_4bee_8822_21fcccab58dc"
```

7. The OVN northbound service adds a Port Binding for the new Logical Switch Port object:

```
_uuid               : 7a558e7b-ed7a-424f-a0cf-ab67d2d832d7
chassis             : b67d6da9-0222-4ab1-a852-ab2607610bf8
datapath            : 3f6e16b5-a03a-48e5-9b60-7b7a0396c425
logical_port        : "e9cb7857-4cb1-4e91-aae5-165a7ab5b387"
mac                 : ["fa:16:3e:b6:91:70 192.168.1.5"]
options             : {}
parent_port         : []
tag                 : []
tunnel_key          : 3
type                : ""
```

8. The OVN northbound service updates the flooding multicast group for the logical datapath with the new port binding:

```
_uuid               : c08d0102-c414-4a47-98d9-dd3fa9f9901c
datapath            : 0b214af6-8910-489c-926a-fd0ed16a8251
name                : _MC_flood
ports               : [3e463ca0-951c-46fd-b6cf-05392fa3aa1f,
                        794a6f03-7941-41ed-b1c6-0e00c1e18da0,
                        fa7b294d-2a62-45ae-8de3-a41c002de6de]
tunnel_key          : 65535
```

9. The OVN northbound service adds Logical Flows based on the updated Address Set, ACL and Logical_Switch_Port objects:

```
Datapath: 3f6e16b5-a03a-48e5-9b60-7b7a0396c425  Pipeline: ingress
  table= 0(  ls_in_port_sec_l2), priority=   50,
    match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
      eth.src == {fa:16:3e:b6:a3:54}),
    action=(next;)
```

(continues on next page)

(continued from previous page)

```
table= 1(  ls_in_port_sec_ip), priority=   90,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
    eth.src == fa:16:3e:b6:a3:54 && ip4.src == 0.0.0.0 &&
    ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst == 67),
  action=(next;)
table= 1(  ls_in_port_sec_ip), priority=   90,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
    eth.src == fa:16:3e:b6:a3:54 && ip4.src == {192.168.1.5}),
  action=(next;)
table= 1(  ls_in_port_sec_ip), priority=   80,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
    eth.src == fa:16:3e:b6:a3:54 && ip),
  action=(drop;)
table= 2(  ls_in_port_sec_nd), priority=   90,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
    eth.src == fa:16:3e:b6:a3:54 && arp.sha == fa:16:3e:b6:a3:54 &&
    (arp.spa == 192.168.1.5 )),
  action=(next;)
table= 2(  ls_in_port_sec_nd), priority=   80,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
    (arp || nd)),
  action=(drop;)
table= 3(      ls_in_pre_acl), priority=  110, match=(nd),
  action=(next;)
table= 3(      ls_in_pre_acl), priority=  100, match=(ip),
  action=(reg0[0] = 1; next;)
table= 6(          ls_in_acl), priority=65535,
  match=(!ct.est && ct.rel && !ct.new && !ct.inv),
  action=(next;)
table= 6(          ls_in_acl), priority=65535,
  match=(ct.est && !ct.rel && !ct.new && !ct.inv),
  action=(next;)
table= 6(          ls_in_acl), priority=65535, match=(ct.inv),
  action=(drop;)
table= 6(          ls_in_acl), priority=65535, match=(nd),
  action=(next;)
table= 6(          ls_in_acl), priority= 2002,
  match=(ct.new && (inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387
↪" &&
    ip6)),
  action=(reg0[1] = 1; next;)
table= 6(          ls_in_acl), priority= 2002,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip4 &&
    (ip4.dst == 255.255.255.255 || ip4.dst == 192.168.1.0/24) &&
    udp && udp.src == 68 && udp.dst == 67),
  action=(reg0[1] = 1; next;)
table= 6(          ls_in_acl), priority= 2002,
  match=(ct.new && (inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387
↪" &&
    ip4)),
  action=(reg0[1] = 1; next;)
table= 6(          ls_in_acl), priority= 2001,
  match=(inport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip),
  action=(drop;)
table= 6(          ls_in_acl), priority=    1, match=(ip),
  action=(reg0[1] = 1; next;)
```

(continues on next page)

(continued from previous page)

```
  table= 9(   ls_in_arp_nd_rsp), priority=   50,
    match=(arp.tpa == 192.168.1.5 && arp.op == 1),
    action=(eth.dst = eth.src; eth.src = fa:16:3e:b6:a3:54; arp.op =␣
↪2; /* ARP reply */ arp.tha = arp.sha; arp.sha = fa:16:3e:b6:a3:54;␣
↪arp.tpa = arp.spa; arp.spa = 192.168.1.5; outport = inport; inport␣
↪= ""; /* Allow sending out inport. */ output;)
  table=10(      ls_in_l2_lkup), priority=   50,
    match=(eth.dst == fa:16:3e:b6:a3:54),
    action=(outport = "e9cb7857-4cb1-4e91-aae5-165a7ab5b387"; output;)
Datapath: 3f6e16b5-a03a-48e5-9b60-7b7a0396c425  Pipeline: egress
  table= 1(     ls_out_pre_acl), priority=  110, match=(nd),
    action=(next;)
  table= 1(     ls_out_pre_acl), priority=  100, match=(ip),
    action=(reg0[0] = 1; next;)
  table= 4(         ls_out_acl), priority=65535, match=(nd),
    action=(next;)
  table= 4(         ls_out_acl), priority=65535,
    match=(!ct.est && ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 4(         ls_out_acl), priority=65535,
    match=(ct.est && !ct.rel && !ct.new && !ct.inv),
    action=(next;)
  table= 4(         ls_out_acl), priority=65535, match=(ct.inv),
    action=(drop;)
  table= 4(         ls_out_acl), priority= 2002,
    match=(ct.new &&
      (outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip6 &&
      ip6.src == $as_ip6_90a78a43_b549_4bee_8822_21fcccab58dc)),
    action=(reg0[1] = 1; next;)
  table= 4(         ls_out_acl), priority= 2002,
    match=(ct.new &&
      (outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip4 &&
      ip4.src == $as_ip4_90a78a43_b549_4bee_8822_21fcccab58dc)),
    action=(reg0[1] = 1; next;)
  table= 4(         ls_out_acl), priority= 2002,
    match=(outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip4 &&
      ip4.src == 192.168.1.0/24 && udp && udp.src == 67 && udp.dst ==␣
↪68),
    action=(reg0[1] = 1; next;)
  table= 4(         ls_out_acl), priority= 2001,
    match=(outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" && ip),
    action=(drop;)
  table= 4(         ls_out_acl), priority=    1, match=(ip),
    action=(reg0[1] = 1; next;)
  table= 6( ls_out_port_sec_ip), priority=   90,
    match=(outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
      eth.dst == fa:16:3e:b6:a3:54 &&
      ip4.dst == {255.255.255.255, 224.0.0.0/4, 192.168.1.5}),
    action=(next;)
  table= 6( ls_out_port_sec_ip), priority=   80,
    match=(outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
      eth.dst == fa:16:3e:b6:a3:54 && ip),
    action=(drop;)
  table= 7( ls_out_port_sec_l2), priority=   50,
    match=(outport == "e9cb7857-4cb1-4e91-aae5-165a7ab5b387" &&
      eth.dst == {fa:16:3e:b6:a3:54}),
```

(continues on next page)

(continued from previous page)

```
    action=(output;)
```

10. The OVN controller service on each compute node translates these objects into flows on the integration bridge `br-int`. Exact flows depend on whether the compute node containing the instance also contains a DHCP agent on the subnet.

    - On the compute node containing the instance, the Compute service creates a port that connects the instance to the integration bridge and OVN creates the following flows:

    ```
    # ovs-ofctl show br-int
    OFPT_FEATURES_REPLY (xid=0x2): dpid:000022024a1dc045
    n_tables:254, n_buffers:256
    capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_
    ↪MATCH_IP
    actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_
    ↪dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src␣
    ↪mod_tp_dst
     12(tapeaf36f62-56): addr:fe:16:3e:15:7d:13
         config:     0
         state:      0
         current:    10MB-FD COPPER
    ```

    ```
    cookie=0x0, duration=179.460s, table=0, n_packets=122, n_
    ↪bytes=10556,
        idle_age=1, priority=100,in_port=12
        actions=load:0x4->NXM_NX_REG5[],load:0x5->OXM_OF_METADATA[],
            load:0x3->NXM_NX_REG6[],resubmit(,16)
    cookie=0x0, duration=187.408s, table=16, n_packets=122, n_
    ↪bytes=10556,
        idle_age=1, priority=50,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13
        actions=resubmit(,17)
    cookie=0x0, duration=187.408s, table=17, n_packets=2, n_bytes=684,
        idle_age=84, priority=90,udp,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13,nw_src=0.0.0.0,nw_dst=255.255.
    ↪255.255,
            tp_src=68,tp_dst=67
        actions=resubmit(,18)
    cookie=0x0, duration=187.408s, table=17, n_packets=98, n_
    ↪bytes=8276,
        idle_age=1, priority=90,ip,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13,nw_src=192.168.1.5
        actions=resubmit(,18)
    cookie=0x0, duration=187.408s, table=17, n_packets=17, n_
    ↪bytes=1386,
        idle_age=55, priority=80,ipv6,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13
        actions=drop
    cookie=0x0, duration=187.408s, table=17, n_packets=0, n_bytes=0,
        idle_age=187, priority=80,ip,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13
        actions=drop
    cookie=0x0, duration=187.408s, table=18, n_packets=5, n_bytes=210,
        idle_age=10, priority=90,arp,reg6=0x3,metadata=0x5,
            dl_src=fa:16:3e:15:7d:13,arp_spa=192.168.1.5,
    ```

(continues on next page)

(continued from previous page)

```
    arp_sha=fa:16:3e:15:7d:13
    actions=resubmit(,19)
cookie=0x0, duration=187.408s, table=18, n_packets=0, n_bytes=0,
    idle_age=187, priority=80,icmp6,reg6=0x3,metadata=0x5,
        icmp_type=135,icmp_code=0
    actions=drop
cookie=0x0, duration=187.408s, table=18, n_packets=0, n_bytes=0,
    idle_age=187, priority=80,icmp6,reg6=0x3,metadata=0x5,
        icmp_type=136,icmp_code=0
    actions=drop
cookie=0x0, duration=187.408s, table=18, n_packets=0, n_bytes=0,
    idle_age=187, priority=80,arp,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=33, n_
↪bytes=4081,
    idle_age=0, priority=100,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=100,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=47.068s, table=22, n_packets=15, n_
↪bytes=1392,
    idle_age=0, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x5
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x5
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=+inv+trk,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,ct_state=+new+trk,ipv6,reg6=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=16, n_
↪bytes=1922,
    idle_age=2, priority=2002,ct_state=+new+trk,ip,reg6=0x3,
        metadata=0x5
```

(continues on next page)

(continued from previous page)

```
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,udp,reg6=0x3,metadata=0x5,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,udp,reg6=0x3,metadata=0x5,
        nw_dst=192.168.1.0/24,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.069s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ipv6,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ip,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=2, n_bytes=767,
    idle_age=27, priority=1,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=1,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=179.457s, table=25, n_packets=2, n_bytes=84,
    idle_age=33, priority=50,arp,metadata=0x5,arp_tpa=192.168.1.5,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:15:7d:13,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163e157d13->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80105->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,
↪32)
cookie=0x0, duration=187.408s, table=26, n_packets=50, n_
↪bytes=4806,
    idle_age=1, priority=50,metadata=0x5,dl_dst=fa:16:3e:15:7d:13
    actions=load:0x3->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=469.575s, table=33, n_packets=74, n_
↪bytes=7040,
    idle_age=305, priority=100,reg7=0x4,metadata=0x4
    actions=load:0x1->NXM_NX_REG7[],resubmit(,33)
cookie=0x0, duration=179.460s, table=34, n_packets=2, n_bytes=684,
    idle_age=84, priority=100,reg6=0x3,reg7=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.069s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=47.068s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=47.068s, table=49, n_packets=34, n_
↪bytes=4455,
    idle_age=0, priority=100,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=47.068s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=100,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=47.069s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=+inv+trk,metadata=0x5
    actions=drop
cookie=0x0, duration=47.069s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=22, n_
↪bytes=2000,
    idle_age=0, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x5
    actions=resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x5
    actions=resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,ct_state=+new+trk,ip,reg7=0x3,
        metadata=0x5,nw_src=192.168.1.5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,ct_state=+new+trk,ip,reg7=0x3,
        metadata=0x5,nw_src=203.0.113.103
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=3, n_bytes=1141,
    idle_age=27, priority=2002,udp,reg7=0x3,metadata=0x5,
        nw_src=192.168.1.0/24,tp_src=67,tp_dst=68
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=39.497s, table=52, n_packets=0, n_bytes=0,
    idle_age=39, priority=2002,ct_state=+new+trk,ipv6,reg7=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ip,reg7=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ipv6,reg7=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=52, n_packets=9, n_bytes=1314,
    idle_age=2, priority=1,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=47.068s, table=52, n_packets=0, n_bytes=0,
    idle_age=47, priority=1,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=47.068s, table=54, n_packets=23, n_
↪bytes=2945,
    idle_age=0, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=192.168.1.11
    actions=resubmit(,55)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=47.068s, table=54, n_packets=0, n_bytes=0,
    idle_age=47, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=255.255.255.255
    actions=resubmit(,55)
cookie=0x0, duration=47.068s, table=54, n_packets=0, n_bytes=0,
    idle_age=47, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=224.0.0.0/4
    actions=resubmit(,55)
cookie=0x0, duration=47.068s, table=54, n_packets=0, n_bytes=0,
    idle_age=47, priority=80,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=47.068s, table=54, n_packets=0, n_bytes=0,
    idle_age=47, priority=80,ipv6,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=47.068s, table=55, n_packets=25, n_
↪bytes=3029,
    idle_age=0, priority=50,reg7=0x3,metadata=0x7,
        dl_dst=fa:16:3e:15:7d:13
    actions=resubmit(,64)
cookie=0x0, duration=179.460s, table=64, n_packets=116, n_
↪bytes=10623,
    idle_age=1, priority=100,reg7=0x3,metadata=0x5
    actions=output:12
```

- For each compute node that only contains a DHCP agent on the subnet, OVN creates the following flows:

```
cookie=0x0, duration=192.587s, table=16, n_packets=0, n_bytes=0,
    idle_age=192, priority=50,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13
    actions=resubmit(,17)
cookie=0x0, duration=192.587s, table=17, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,ip,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13,nw_src=192.168.1.5
    actions=resubmit(,18)
cookie=0x0, duration=192.587s, table=17, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,udp,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13,nw_src=0.0.0.0,
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=resubmit(,18)
cookie=0x0, duration=192.587s, table=17, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,ipv6,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=192.587s, table=17, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,ip,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=192.587s, table=18, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,arp,reg6=0x3,metadata=0x5,
        dl_src=fa:16:3e:15:7d:13,arp_spa=192.168.1.5,
        arp_sha=fa:16:3e:15:7d:13
    actions=resubmit(,19)
cookie=0x0, duration=192.587s, table=18, n_packets=0, n_bytes=0,
```

(continues on next page)

(continued from previous page)

```
    idle_age=192, priority=80,arp,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=192.587s, table=18, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,icmp6,reg6=0x3,metadata=0x5,
        icmp_type=135,icmp_code=0
    actions=drop
cookie=0x0, duration=192.587s, table=18, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,icmp6,reg6=0x3,metadata=0x5,
        icmp_type=136,icmp_code=0
    actions=drop
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=33, n_
↪bytes=4081,
    idle_age=0, priority=100,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=47.068s, table=19, n_packets=0, n_bytes=0,
    idle_age=47, priority=100,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,20)
cookie=0x0, duration=47.068s, table=22, n_packets=15, n_
↪bytes=1392,
    idle_age=0, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x5
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x5
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=65535,ct_state=+inv+trk,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,ct_state=+new+trk,ipv6,reg6=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=16, n_
↪bytes=1922,
    idle_age=2, priority=2002,ct_state=+new+trk,ip,reg6=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,udp,reg6=0x3,metadata=0x5,
```

(continues on next page)

(continued from previous page)

```
        nw_dst=255.255.255.255,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2002,udp,reg6=0x3,metadata=0x5,
        nw_dst=192.168.1.0/24,tp_src=68,tp_dst=67
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.069s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ipv6,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=2001,ip,reg6=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=47.068s, table=22, n_packets=2, n_bytes=767,
    idle_age=27, priority=1,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=47.068s, table=22, n_packets=0, n_bytes=0,
    idle_age=47, priority=1,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,23)
cookie=0x0, duration=179.457s, table=25, n_packets=2, n_bytes=84,
    idle_age=33, priority=50,arp,metadata=0x5,arp_tpa=192.168.1.5,
        arp_op=1
    actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],
        mod_dl_src:fa:16:3e:15:7d:13,load:0x2->NXM_OF_ARP_OP[],
        move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],
        load:0xfa163e157d13->NXM_NX_ARP_SHA[],
        move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],
        load:0xc0a80105->NXM_OF_ARP_SPA[],
        move:NXM_NX_REG6[]->NXM_NX_REG7[],
        load:0->NXM_NX_REG6[],load:0->NXM_OF_IN_PORT[],resubmit(,
↪32)
cookie=0x0, duration=192.587s, table=26, n_packets=61, n_
↪bytes=5607,
    idle_age=6, priority=50,metadata=0x5,dl_dst=fa:16:3e:15:7d:13
    actions=load:0x3->NXM_NX_REG7[],resubmit(,32)
cookie=0x0, duration=184.640s, table=32, n_packets=61, n_
↪bytes=5607,
    idle_age=6, priority=100,reg7=0x3,metadata=0x5
    actions=load:0x5->NXM_NX_TUN_ID[0..23],
        set_field:0x3/0xffffffff->tun_metadata0,
        move:NXM_NX_REG6[0..14]->NXM_NX_TUN_METADATA0[16..30],
↪output:4
cookie=0x0, duration=47.069s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=135,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=47.068s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=110,icmp6,metadata=0x5,icmp_type=136,
        icmp_code=0
    actions=resubmit(,50)
cookie=0x0, duration=47.068s, table=49, n_packets=34, n_
↪bytes=4455,
    idle_age=0, priority=100,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
cookie=0x0, duration=47.068s, table=49, n_packets=0, n_bytes=0,
    idle_age=47, priority=100,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[0],resubmit(,50)
```

(continues on next page)

(continued from previous page)

```
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=65535,ct_state=+inv+trk,
        metadata=0x5
    actions=drop
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=65535,ct_state=-new-est+rel-inv+trk,
        metadata=0x5
    actions=resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=27, n_
↪bytes=2316,
    idle_age=6, priority=65535,ct_state=-new+est-rel-inv+trk,
        metadata=0x5
    actions=resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2002,ct_state=+new+trk,icmp,reg7=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2002,ct_state=+new+trk,ipv6,reg7=0x3,
        metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2002,udp,reg7=0x3,metadata=0x5,
        nw_src=192.168.1.0/24,tp_src=67,tp_dst=68
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2002,ct_state=+new+trk,ip,reg7=0x3,
        metadata=0x5,nw_src=203.0.113.103
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,50)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2001,ip,reg7=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=2001,ipv6,reg7=0x3,metadata=0x5
    actions=drop
cookie=0x0, duration=192.587s, table=52, n_packets=25, n_
↪bytes=2604,
    idle_age=6, priority=1,ip,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=192.587s, table=52, n_packets=0, n_bytes=0,
    idle_age=192, priority=1,ipv6,metadata=0x5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=192.587s, table=54, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=224.0.0.0/4
    actions=resubmit(,55)
cookie=0x0, duration=192.587s, table=54, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=255.255.255.255
    actions=resubmit(,55)
cookie=0x0, duration=192.587s, table=54, n_packets=0, n_bytes=0,
    idle_age=192, priority=90,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13,nw_dst=192.168.1.5
    actions=resubmit(,55)
cookie=0x0, duration=192.587s, table=54, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,ipv6,reg7=0x3,metadata=0x5,
```

(continues on next page)

(continued from previous page)

```
        dl_dst=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=192.587s, table=54, n_packets=0, n_bytes=0,
    idle_age=192, priority=80,ip,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13
    actions=drop
cookie=0x0, duration=192.587s, table=55, n_packets=0, n_bytes=0,
    idle_age=192, priority=50,reg7=0x3,metadata=0x5,
        dl_dst=fa:16:3e:15:7d:13
    actions=resubmit(,64)
```

- For each compute node that contains neither the instance nor a DHCP agent on the subnet, OVN creates the following flows:

```
cookie=0x0, duration=189.763s, table=52, n_packets=0, n_bytes=0,
    idle_age=189, priority=2002,ct_state=+new+trk,ipv6,reg7=0x4,
        metadata=0x4
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
cookie=0x0, duration=189.763s, table=52, n_packets=0, n_bytes=0,
    idle_age=189, priority=2002,ct_state=+new+trk,ip,reg7=0x4,
        metadata=0x4,nw_src=192.168.1.5
    actions=load:0x1->NXM_NX_REG0[1],resubmit(,53)
```

8.7.7 DPDK Support in OVN

Configuration Settings

The following configuration parameter needs to be set in the Neutron ML2 plugin configuration file under the ovn section to enable DPDK support.

vhost_sock_dir This is the directory path in which vswitch daemon in all the compute nodes creates the virtio socket. Follow the instructions in INSTALL.DPDK.md in openvswitch source tree to know how to configure DPDK support in vswitch daemons.

Configuration Settings in compute hosts

Compute nodes configured with OVS DPDK should set the datapath_type as netdev for the integration bridge (managed by OVN) and all other bridges if connected to the integration bridge via patch ports. The below command can be used to set the datapath_type.

```
$ sudo ovs-vsctl set Bridge br-int datapath_type=netdev
```

8.7.8 Troubleshooting

The following section describe common problems that you might encounter after/during the installation of the OVN ML2 driver with Devstack and possible solutions to these problems.

Launching VMs failure

Disable AppArmor

Using Ubuntu you might encounter libvirt permission errors when trying to create OVS ports after launching a VM (from the nova compute log). Disabling AppArmor might help with this problem, check out https://help.ubuntu.com/community/AppArmor for instructions on how to disable it.

Multi-Node setup not working

Geneve kernel module not supported

By default OVN creates tunnels between compute nodes using the Geneve protocol. Older kernels (< 3.18) dont support the Geneve module and hence tunneling cant work. You can check it with this command lsmod | grep openvswitch (geneve should show up in the result list)

For more information about which upstream Kernel version is required for support of each tunnel type, see the answer to Why do tunnels not work when using a kernel module other than the one packaged with Open vSwitch? in the OVS FAQ.

MTU configuration

This problem is not unique to OVN but is amplified due to the possible larger size of geneve header compared to other common tunneling protocols (VXLAN). If you are using VMs as compute nodes make sure that you either lower the MTU size on the virtual interface or enable fragmentation on it.

8.7.9 SR-IOV guide for OVN

The purpose of this page is to describe how SR-IOV works with OVN. Prior to reading this document, it is recommended to first read *the basic guide for SR-IOV*.

External ports

In order for SR-IOV to work with the Neutron driver we are leveraging the external ports feature from the OVN project. When virtual machines are booted on hypervisors supporting SR-IOV nics, the local ovn-controllers are unable to reply to the VMs DHCP, internal DNS, IPv6 router solicitation requests, etc since the hypervisor is bypassed in the SR-IOV case. OVN then introduced the idea of having `external` ports which are able to reply on behalf of those VM ports external to the hypervisor that they are running on.

The OVN Neutron driver will create a port of the type `external` for ports with the following VNICs set:

- direct

- direct-physical
- macvtap

Also, ports of the type `external` will be scheduled on the gateway nodes (controller or networker nodes) in HA mode by the OVN Neutron driver. Check the *OVN Database information* section for more information.

Environment setup for OVN SR-IOV

There are a very few differences between setting up an environment for SR-IOV for the OVS and OVN Neutron drivers. As mentioned at the beginning of this document, the instructions from the *the basic guide for SR-IOV* are required for getting SR-IOV working with the OVN driver.

The only differences required for an OVN deployment are:

- When configuring the `mechanism_drivers` in the *ml2_conf.ini* file we should specify `ovn` driver instead of the `openvswitch` driver
- Disabling the Neutron DHCP agent
- Deploying the OVN Metadata agent on the gateway nodes (controller or networker nodes)

OVN Database information

Before getting into the ports information, the previous sections talks about **gateway nodes**, the OVN Neutron driver identifies a gateway node by the `ovn-cms-options=enable-chassis-as-gw` and `ovn-bridge-mappings` options in the external_ids column from the `Chassis` table in the OVN Southbound database:

```
$ ovn-sbctl list Chassis
_uuid                : 12b13aff-a821-4cde-a4ac-d9cf8e2c91bc
external_ids         : {ovn-cms-options=enable-chassis-as-gw, ovn-bridge-
↪mappings="public:br-ex", ...}
hostname             : controller-0
name                 : "1a462946-ccfd-46a6-8abf-9dca9eb558fb"
...
```

For more information about both of these options, please take a look at the ovn-controller documentation.

These options can be set by running the following command locally on each gateway node (note, the `ovn-bridge-mappings` will need to be adapted to your environment):

```
$ ovs-vsctl set Open_vSwitch . external-ids:ovn-cms-options=\"enable-
↪chassis-as-gw\" external-ids:ovn-bridge-mappings=\"public:br-ex\"
```

As mentioned in the External ports section, every time a Neutron port with a certain VNIC is created the OVN driver will create a port of the type `external` in the OVN Northbound database. These ports can be found by issuing the following command:

```
$ ovn-nbctl find Logical_Switch_Port type=external
_uuid                : 105e83ae-252d-401b-a1a7-8d28ec28a359
ha_chassis_group     : [43047e7b-4c78-4984-9788-6263fcc69885]
type                 : external
...
```

The `ha_chassis_group` column indicates which HA Chassis Group that port belongs to, to find that group do:

```
# The UUID is the one from the ha_chassis_group column from
# the Logical_Switch_Port table
$ ovn-nbctl list HA_Chassis_Group 43047e7b-4c78-4984-9788-6263fcc69885
_uuid                : 43047e7b-4c78-4984-9788-6263fcc69885
external_ids         : {}
ha_chassis           : [3005bf84-fc95-4361-866d-bfa1c980adc8, 72c7671e-dd48-
↪4100-9741-c47221672961]
name                 : default_ha_chassis_group
```

Note: For now, the OVN driver only has one HA Chassis Group created called `default_ha_chassis_group`. All `external` ports in the system will belong to this group.

The chassis that are members of the `default_ha_chassis_group` HA Chassis Group are listed in the `ha_chassis` column. Those are the gateway nodes (controller or networker nodes) in the deployment and its where the `external` ports will be scheduled. In order to find which gateway node the external ports are scheduled on use the following command:

```
# The UUIDs are the UUID members of the HA Chassis Group
# (ha_chassis column from the HA_Chassis_Group table)
$ ovn-nbctl list HA_Chassis 3005bf84-fc95-4361-866d-bfa1c980adc8 72c7671e-
↪dd48-4100-9741-c47221672961
_uuid                : 3005bf84-fc95-4361-866d-bfa1c980adc8
chassis_name         : "1a462946-ccfd-46a6-8abf-9dca9eb558fb"
external_ids         : {}
priority             : 32767

_uuid                : 72c7671e-dd48-4100-9741-c47221672961
chassis_name         : "a0cb9d55-a6da-4f84-857f-d4b674088c8c"
external_ids         : {}
priority             : 32766
```

Note the `priority` column from the previous command, the chassis with the highest `priority` from that list is the chassis that will have the external ports scheduled on it. In our example above, the chassis with the UUID `1a462946-ccfd-46a6-8abf-9dca9eb558fb` is the one.

Whenever the chassis with the highest priority goes down, the ports will be automatically scheduled on the next chassis with the highest priority which is alive. So, the external ports are HA out of the box.

Known limitations

The current SR-IOV implementation for the OVN Neutron driver has a few known limitations that should be addressed in the future:

1. At the moment, **all** external ports will be scheduled on a single gateway node since theres only one HA Chassis Group for all of those ports.
2. Routing on VLAN tenant network will not work with SR-IOV. This is because the external ports are not being co-located with the logical routers gateway ports, for more information take a look at bug #1875852.

8.7.10 Router Availability Zones guide for OVN

The purpose of this page is to describe how the router availability zones works with OVN. Prior to reading this document, it is recommended to first read *ML2/OVS driver Availability Zones guide*.

How to configure it

Different from the ML2/OVS driver for Neutron the availability zones for the OVN driver is not configured via a configuration file. Since ML2/OVN does not rely on an external agent such as the L3 agent, certain nodes (e.g gateway/networker node) wont have any Neutron configuration file present. For this reason, OVN uses the local OVSDB for configuring the availability zones that instance of `ovn-controller` running on that hypervisor belongs to.

The configuration is done via the `ovn-cms-options` entry in *external_ids* column of the local *Open_vSwitch* table:

```
$ ovs-vsctl set Open_vSwitch . external-ids:ovn-cms-options="enable-
↪chassis-as-gw,availability-zones=az-0:az-1:az-2"
```

The above command is adding two configurations to the `ovn-cms-options` option, the `enable-chassis-as-gw` option which tells the OVN driver that this is a gateway/networker node and the `availability-zones` option specifying three availability zones: **az-0**, **az-1** and **az-2**.

Note that, the syntax used to specify the availability zones is the `availability-zones` word, followed by an equal sign (=) and a **colon** separated list of the availability zones that this local `ovn-controller` instance belongs to.

To confirm the specific `ovn-controller` availability zones, check the **Availability Zone** column in the output of the command below:

```
$ openstack network agent list
+--------------------------------------+-------------------------------+----
↪------------+-------------------+-------+-------+----------------+
| ID                                   | Agent Type                    |␣
↪Host          | Availability Zone | Alive | State | Binary         |
+--------------------------------------+-------------------------------+----
↪------------+-------------------+-------+-------+----------------+
| 2d1924b2-99a4-4c6c-a4f2-0be64c0cec8c | OVN Controller Gateway agent |␣
↪gateway-host-0 | az0, az1, az2      | :-)   | UP    | ovn-controller |
+--------------------------------------+-------------------------------+----
↪------------+-------------------+-------+-------+----------------+
```

Note: If you know the UUID of the agent the **openstack network agent show <UUID>** command can also be used.

To confirm the availability zones defined in the system as a whole:

```
$ openstack availability zone list
+-----------+-------------+
| Zone Name | Zone Status |
+-----------+-------------+
| az0       | available   |
| az1       | available   |
```

(continues on next page)

(continued from previous page)

```
| az2         | available   |
+-----------+-------------+
```

Using router availability zones

In order to create a router with availability zones the `--availability-zone-hint` should be passed to the create command, note that this parameter can be specified multiple times in case the router belongs to more than one availability zone. For example:

```
$ openstack router create --availability-zone-hint az-0 --availability-
↪zone-hint az-1 router-0
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints | az-0, az-1                           |
| availability_zones      |                                      |
| created_at              | 2020-06-04T08:29:33Z                 |
| description             |                                      |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| id                      | 8fd6d01a-57ad-4e91-a788-ebe48742d000 |
| name                    | router-0                             |
| project_id              | 2a364ced6c084888be0919450629de1c     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    |                                      |
| updated_at              | 2020-06-04T08:29:33Z                 |
+-------------------------+--------------------------------------+
```

Its also possible to set the default availability zones via the */etc/neutron/neutron.conf* configuration file:

```
[DEFAULT]
default_availability_zones = az-0,az-2
...
```

When scheduling the gateway ports of a router, the OVN driver will take into consideration the router availability zones and make sure that the ports are scheduled on the nodes belonging to those availability zones.

Note that in the router object we have two attributes related to availability zones: `availability_zones` and `availability_zone_hints`:

```
| availability_zone_hints | az-0, az-1                           |
| availability_zones      |                                      |
```

This distinction makes more sense in the **ML2/OVS** driver which relies on the L3 agent for its router placement (see the *ML2/OVS driver Availability Zones guide* for more information). In **ML2/OVN** the `ovn-controller` service will be running on all nodes of the cluster so the `availability_zone_hints` will always match the `availability_zones` attribute.

OVN Database information

In order to check the availability zones of a router via the OVN Northbound database, one can look for the `neutron:availability_zone_hints` key in the `external_ids` column for its entry in the `Logical_Router` table:

```
$ ovn-nbctl list Logical_Router
_uuid               : 4df68f1e-17dd-4b9a-848d-b6152ae19203
external_ids        : {"neutron:availability_zone_hints"="az-0,az-1",
↪"neutron:gw_port_id"="", "neutron:revision_number"="1", "neutron:router_
↪name"=router-0}
name                : neutron-8fd6d01a-57ad-4e91-a788-ebe48742d000
...
```

To check the availability zones of the Chassis, look at the `ovn-cms-options` key in the `other_config` column (or `external_ids` for an older version of OVN) of the `Chassis` table in the OVN Southbound database:

```
$ ovn-sbctl list Chassis
_uuid               : abaa9f07-9988-40c0-bd1a-8d8326af08b0
name                : "2d1924b2-99a4-4c6c-a4f2-0be64c0cec8c"
other_config        : {..., ovn-cms-options="enable-chassis-as-gw,
↪availability-zones=az-0:az-1:az-2"}
...
```

As mentioned in the *Using router availability zones* section, the scheduling of the gateway router ports will take into consideration the availability zones that the router belongs to. We can confirm this behavior by looking in the `Gateway_Chassis` table from the OVN Southbound database:

```
$ ovn-sbctl list Gateway_Chassis
_uuid               : ac61b70f-ff51-43d9-830b-f9bc6d74090a
chassis_name        : "2d1924b2-99a4-4c6c-a4f2-0be64c0cec8c"
external_ids        : {}
name                : lrp-5a40eeca-5233-4029-a470-9018aa8b3de9_2d1924b2-
↪99a4-4c6c-a4f2-0be64c0cec8c
options             : {}
priority            : 2

_uuid               : c1b7763b-1784-4e5a-a948-853662faeddc
chassis_name        : "1cde2542-69f9-4598-b20b-d4f68304deb0"
external_ids        : {}
name                : lrp-5a40eeca-5233-4029-a470-9018aa8b3de9_1cde2542-
↪69f9-4598-b20b-d4f68304deb0
options             : {}
priority            : 1
```

Each entry on this table represents an instance of the gateway port (L3 HA, for more information see *Routing in OVN*), the `chassis_name` column indicates which Chassis that port instance is scheduled onto. If we co-relate each entry and their `chassis_name` we will see that this port has been only scheduled to Chassis matching with the routers availability zones.

8.8 Archived Contents

Note: Contents here have been moved from the unified version of Administration Guide. They will be merged into the Networking Guide gradually.

8.8.1 Introduction to Networking

The Networking service, code-named neutron, provides an API that lets you define network connectivity and addressing in the cloud. The Networking service enables operators to leverage different networking technologies to power their cloud networking. The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and NAT to edge firewalls, and IPsec VPN.

For a detailed description of the Networking API abstractions and their attributes, see the OpenStack Networking API v2.0 Reference.

Note: If you use the Networking service, do not run the Compute `nova-network` service (like you do in traditional Compute deployments). When you configure networking, see the Compute-related topics in this Networking section.

Networking API

Networking is a virtual network service that provides a powerful API to define the network connectivity and IP addressing that devices from other services, such as Compute, use.

The Compute API has a virtual server abstraction to describe computing resources. Similarly, the Networking API has virtual network, subnet, and port abstractions to describe networking resources.

Resource	Description
Network	An isolated L2 segment, analogous to VLAN in the physical networking world.
Subnet	A block of v4 or v6 IP addresses and associated configuration state.
Port	A connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. Also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

Networking resources

To configure rich network topologies, you can create and configure networks and subnets and instruct other OpenStack services like Compute to attach virtual devices to ports on these networks.

In particular, Networking supports each project having multiple private networks and enables projects to choose their own IP addressing scheme, even if those IP addresses overlap with those that other projects use.

The Networking service:

- Enables advanced cloud networking use cases, such as building multi-tiered web applications and enabling migration of applications to the cloud without changing IP addresses.
- Offers flexibility for administrators to customize network offerings.
- Enables developers to extend the Networking API. Over time, the extended functionality becomes part of the core Networking API.

Configure SSL support for networking API

OpenStack Networking supports SSL for the Networking API server. By default, SSL is disabled but you can enable it in the `neutron.conf` file.

Set these options to configure SSL:

`use_ssl = True` Enables SSL on the networking API server.

`ssl_cert_file = PATH_TO_CERTFILE` Certificate file that is used when you securely start the Networking API server.

`ssl_key_file = PATH_TO_KEYFILE` Private key file that is used when you securely start the Networking API server.

`ssl_ca_file = PATH_TO_CAFILE` Optional. CA certificate file that is used when you securely start the Networking API server. This file verifies connecting clients. Set this option when API clients must authenticate to the API server by using SSL certificates that are signed by a trusted CA.

`tcp_keepidle = 600` The value of TCP_KEEPIDLE, in seconds, for each server socket when starting the API server. Not supported on OS X.

`retry_until_window = 30` Number of seconds to keep retrying to listen.

`backlog = 4096` Number of backlog requests with which to configure the socket.

Firewall-as-a-Service (FWaaS) overview

For information on Firewall-as-a-Service (FWaaS), please consult the *Networking Guide*.

Allowed-address-pairs

`Allowed-address-pairs` enables you to specify mac_address and ip_address(cidr) pairs that pass through a port regardless of subnet. This enables the use of protocols such as VRRP, which floats an IP address between two instances to enable fast data plane failover.

Note: Currently, only the ML2, Open vSwitch, and VMware NSX plug-ins support the allowed-address-pairs extension.

Basic allowed-address-pairs operations.

- Create a port with a specified allowed address pair:

```
$ openstack port create port1 --allowed-address \
ip-address=<IP_CIDR>[,mac_address=<MAC_ADDRESS]
```

- Update a port by adding allowed address pairs:

```
$ openstack port set PORT_UUID --allowed-address \
ip-address=<IP_CIDR>[,mac_address=<MAC_ADDRESS]
```

Virtual-Private-Network-as-a-Service (VPNaaS)

The VPNaaS extension enables OpenStack projects to extend private networks across the internet.

VPNaaS is a service. It is a parent object that associates a VPN with a specific subnet and router. Only one VPN service object can be created for each router and each subnet. However, each VPN service object can have any number of IP security connections.

The Internet Key Exchange (IKE) policy specifies the authentication and encryption algorithms to use during phase one and two negotiation of a VPN connection. The IP security policy specifies the authentication and encryption algorithm and encapsulation mode to use for the established VPN connection. Note that you cannot update the IKE and IPSec parameters for live tunnels.

You can set parameters for site-to-site IPsec connections, including peer CIDRs, MTU, authentication mode, peer address, DPD settings, and status.

The current implementation of the VPNaaS extension provides:

- Site-to-site VPN that connects two private networks.
- Multiple VPN connections per project.
- IKEv1 policy support with 3des, aes-128, aes-256, or aes-192 encryption.
- IPSec policy support with 3des, aes-128, aes-192, or aes-256 encryption, sha1 authentication, ESP, AH, or AH-ESP transform protocol, and tunnel or transport mode encapsulation.
- Dead Peer Detection (DPD) with hold, clear, restart, disabled, or restart-by-peer actions.

The VPNaaS driver plugin can be configured in the neutron configuration file. You can then enable the service.

8.8.2 Networking architecture

Before you deploy Networking, it is useful to understand the Networking services and how they interact with the OpenStack components.

Overview

Networking is a standalone component in the OpenStack modular architecture. It is positioned alongside OpenStack components such as Compute, Image service, Identity, or Dashboard. Like those components, a deployment of Networking often involves deploying several services to a variety of hosts.

The Networking server uses the neutron-server daemon to expose the Networking API and enable administration of the configured Networking plug-in. Typically, the plug-in requires access to a database for persistent storage (also similar to other OpenStack services).

If your deployment uses a controller host to run centralized Compute components, you can deploy the Networking server to that same host. However, Networking is entirely standalone and can be deployed to a dedicated host. Depending on your configuration, Networking can also include the following agents:

Agent	Description
plug-in agent (`neutron-*-agent`)	Runs on each hypervisor to perform local vSwitch configuration. The agent that runs, depends on the plug-in that you use. Certain plug-ins do not require an agent.
dhcp agent (`neutron-dhcp-agent`)	Provides DHCP services to project networks. Required by certain plug-ins.
l3 agent (`neutron-l3-agent`)	Provides L3/NAT forwarding to provide external network access for VMs on project networks. Required by certain plug-ins.
metering agent (`neutron-metering-agent`)	Provides L3 traffic metering for project networks.

These agents interact with the main neutron process through RPC (for example, RabbitMQ or Qpid) or through the standard Networking API. In addition, Networking integrates with OpenStack components in a number of ways:

- Networking relies on the Identity service (keystone) for the authentication and authorization of all API requests.
- Compute (nova) interacts with Networking through calls to its standard API. As part of creating a VM, the `nova-compute` service communicates with the Networking API to plug each virtual NIC on the VM into a particular network.
- The dashboard (horizon) integrates with the Networking API, enabling administrators and project users to create and manage network services through a web-based GUI.

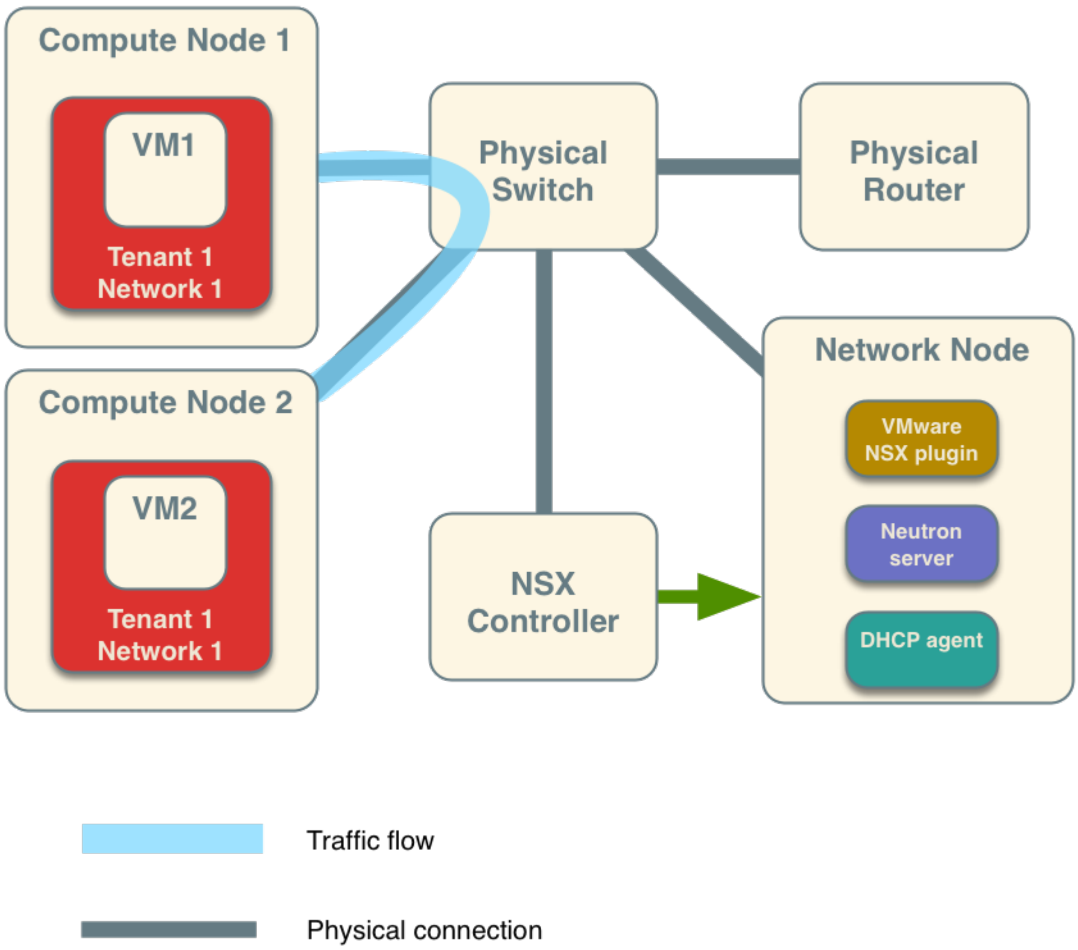
VMware NSX integration

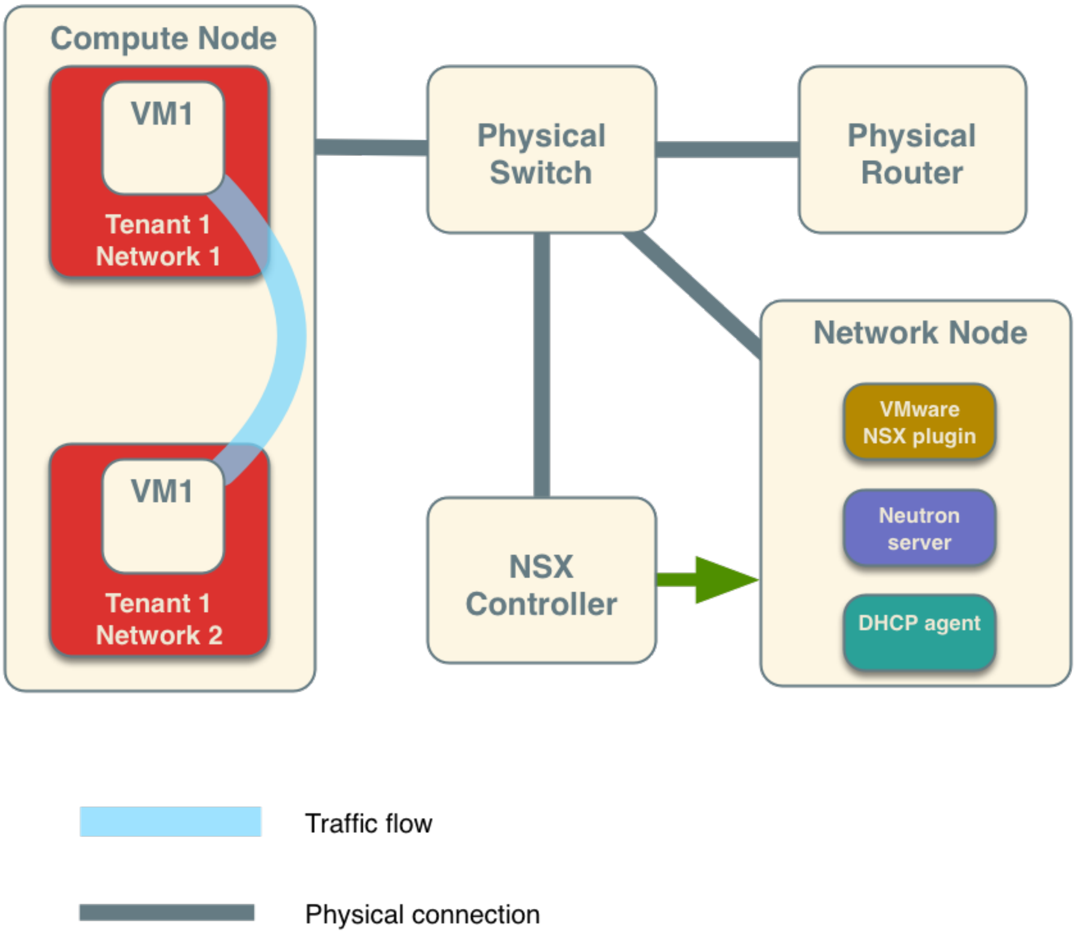
OpenStack Networking uses the NSX plug-in to integrate with an existing VMware vCenter deployment. When installed on the network nodes, the NSX plug-in enables a NSX controller to centrally manage configuration settings and push them to managed network nodes. Network nodes are considered managed when they are added as hypervisors to the NSX controller.

The diagrams below depict some VMware NSX deployment examples. The first diagram illustrates the traffic flow between VMs on separate Compute nodes, and the second diagram between two VMs on a single compute node. Note the placement of the VMware NSX plug-in and the neutron-server service on the network node. The green arrow indicates the management relationship between the NSX controller and the network node.

8.8.3 Plug-in configurations

For configurations options, see Networking configuration options in Configuration Reference. These sections explain how to configure specific plug-ins.


Compute Node 1
VM1
Tenant 1 Network 1
Physical Switch
Physical Router
Network Node
VMware NSX plugin
Neutron server
DHCP agent
Compute Node 2
VM2
Tenant 1 Network 1
NSX Controller
Traffic flow
Physical connection


Compute Node
VM1
Tenant 1
Network 1
VM1
Tenant 1
Network 2
Physical
Switch
Physical
Router
Network Node
VMware
NSX plugin
Neutron
server
DHCP agent
NSX
Controller
Traffic flow
Physical connection

Configure Big Switch (Floodlight REST Proxy) plug-in

1. Edit the `/etc/neutron/neutron.conf` file and add this line:

```
core_plugin = bigswitch
```

2. In the `/etc/neutron/neutron.conf` file, set the `service_plugins` option:

```
service_plugins = neutron.plugins.bigswitch.l3_router_plugin.
↪L3RestProxy
```

3. Edit the `/etc/neutron/plugins/bigswitch/restproxy.ini` file for the plug-in and specify a comma-separated list of controller_ip:port pairs:

```
server = CONTROLLER_IP:PORT
```

For database configuration, see Install Networking Services in the Installation Tutorials and Guides. (The link defaults to the Ubuntu version.)

4. Restart the `neutron-server` to apply the settings:

```
# service neutron-server restart
```

Configure Brocade plug-in

1. Install the Brocade-modified Python netconf client (ncclient) library, which is available at https://github.com/brocade/ncclient:

```
$ git clone https://github.com/brocade/ncclient
```

2. As root, run this command:

```
# cd ncclient;python setup.py install
```

3. Edit the `/etc/neutron/neutron.conf` file and set the following option:

```
core_plugin = brocade
```

4. Edit the `/etc/neutron/plugins/brocade/brocade.ini` file for the Brocade plug-in and specify the admin user name, password, and IP address of the Brocade switch:

```
[SWITCH]
username = ADMIN
password = PASSWORD
address  = SWITCH_MGMT_IP_ADDRESS
ostype   = NOS
```

For database configuration, see Install Networking Services in any of the Installation Tutorials and Guides in the OpenStack Documentation index. (The link defaults to the Ubuntu version.)

5. Restart the `neutron-server` service to apply the settings:

```
# service neutron-server restart
```

Configure NSX-mh plug-in

The instructions in this section refer to the VMware NSX-mh platform, formerly known as Nicira NVP.

1. Install the NSX plug-in:

```
# apt-get install python-vmware-nsx
```

2. Edit the /etc/neutron/neutron.conf file and set this line:

```
core_plugin = vmware
```

   Example neutron.conf file for NSX-mh integration:

```
core_plugin = vmware
rabbit_host = 192.168.203.10
allow_overlapping_ips = True
```

3. To configure the NSX-mh controller cluster for OpenStack Networking, locate the [default] section in the /etc/neutron/plugins/vmware/nsx.ini file and add the following entries:

   - To establish and configure the connection with the controller cluster you must set some parameters, including NSX-mh API endpoints, access credentials, and optionally specify settings for HTTP timeouts, redirects and retries in case of connection failures:

```
nsx_user = ADMIN_USER_NAME
nsx_password = NSX_USER_PASSWORD
http_timeout = HTTP_REQUEST_TIMEOUT # (seconds) default 75 seconds
retries = HTTP_REQUEST_RETRIES # default 2
redirects = HTTP_REQUEST_MAX_REDIRECTS # default 2
nsx_controllers = API_ENDPOINT_LIST # comma-separated list
```

     To ensure correct operations, the nsx_user user must have administrator credentials on the NSX-mh platform.

     A controller API endpoint consists of the IP address and port for the controller; if you omit the port, port 443 is used. If multiple API endpoints are specified, it is up to the user to ensure that all these endpoints belong to the same controller cluster. The OpenStack Networking VMware NSX-mh plug-in does not perform this check, and results might be unpredictable.

     When you specify multiple API endpoints, the plug-in takes care of load balancing requests on the various API endpoints.

   - The UUID of the NSX-mh transport zone that should be used by default when a project creates a network. You can get this value from the Transport Zones page for the NSX-mh manager:

     Alternatively the transport zone identifier can be retrieved by query the NSX-mh API: /ws.v1/transport-zone

```
default_tz_uuid = TRANSPORT_ZONE_UUID
```

   -
```
default_l3_gw_service_uuid = GATEWAY_SERVICE_UUID
```

> **Warning:** Ubuntu packaging currently does not update the neutron init script to point to the NSX-mh configuration file. Instead, you must manually update /etc/default/neutron-server to add this line:
>
> ```
> NEUTRON_PLUGIN_CONFIG = /etc/neutron/plugins/vmware/nsx.ini
> ```

For database configuration, see Install Networking Services in the Installation Tutorials and Guides.

4. Restart neutron-server to apply settings:

```
# service neutron-server restart
```

> **Warning:** The neutron NSX-mh plug-in does not implement initial re-synchronization of Neutron resources. Therefore resources that might already exist in the database when Neutron is switched to the NSX-mh plug-in will not be created on the NSX-mh backend upon restart.

Example nsx.ini file:

```
[DEFAULT]
default_tz_uuid = d3afb164-b263-4aaa-a3e4-48e0e09bb33c
default_l3_gw_service_uuid=5c8622cc-240a-40a1-9693-e6a5fca4e3cf
nsx_user=admin
nsx_password=changeme
nsx_controllers=10.127.0.100,10.127.0.200:8888
```

Note: To debug nsx.ini configuration issues, run this command from the host that runs neutron-server:

```
# neutron-check-nsx-config PATH_TO_NSX.INI
```

This command tests whether neutron-server can log into all of the NSX-mh controllers and the SQL server, and whether all UUID values are correct.

Configure PLUMgrid plug-in

1. Edit the /etc/neutron/neutron.conf file and set this line:

```
core_plugin = plumgrid
```

2. Edit the [PLUMgridDirector] section in the /etc/neutron/plugins/plumgrid/plumgrid.ini file and specify the IP address, port, admin user name, and password of the PLUMgrid Director:

```
[PLUMgridDirector]
director_server = "PLUMgrid-director-ip-address"
director_server_port = "PLUMgrid-director-port"
username = "PLUMgrid-director-admin-username"
password = "PLUMgrid-director-admin-password"
```

For database configuration, see Install Networking Services in the Installation Tutorials and Guides.

3. Restart the `neutron-server` service to apply the settings:

```
# service neutron-server restart
```

8.8.4 Configure neutron agents

Plug-ins typically have requirements for particular software that must be run on each node that handles data packets. This includes any node that runs nova-compute and nodes that run dedicated OpenStack Networking service agents such as `neutron-dhcp-agent`, `neutron-l3-agent` or `neutron-metering-agent`.

A data-forwarding node typically has a network interface with an IP address on the management network and another interface on the data network.

This section shows you how to install and configure a subset of the available plug-ins, which might include the installation of switching software (for example, `Open vSwitch`) and as agents used to communicate with the `neutron-server` process running elsewhere in the data center.

Configure data-forwarding nodes

Node set up: NSX plug-in

If you use the NSX plug-in, you must also install Open vSwitch on each data-forwarding node. However, you do not need to install an additional agent on each node.

> **Warning:** It is critical that you run an Open vSwitch version that is compatible with the current version of the NSX Controller software. Do not use the Open vSwitch version that is installed by default on Ubuntu. Instead, use the Open vSwitch version that is provided on the VMware support portal for your NSX Controller version.

To set up each node for the NSX plug-in

1. Ensure that each data-forwarding node has an IP address on the management network, and an IP address on the data network that is used for tunneling data traffic. For full details on configuring your forwarding node, see the NSX Administration Guide.

2. Use the NSX Administrator Guide to add the node as a Hypervisor by using the NSX Manager GUI. Even if your forwarding node has no VMs and is only used for services agents like `neutron-dhcp-agent` , it should still be added to NSX as a Hypervisor.

3. After following the NSX Administrator Guide, use the page for this Hypervisor in the NSX Manager GUI to confirm that the node is properly connected to the NSX Controller Cluster and that the NSX Controller Cluster can see the `br-int` integration bridge.

Configure DHCP agent

The DHCP service agent is compatible with all existing plug-ins and is required for all deployments where VMs should automatically receive IP addresses through DHCP.

To install and configure the DHCP agent

1. You must configure the host running the neutron-dhcp-agent as a data forwarding node according to the requirements for your plug-in.
2. Install the DHCP agent:

```
# apt-get install neutron-dhcp-agent
```

3. Update any options in the `/etc/neutron/dhcp_agent.ini` file that depend on the plug-in in use. See the sub-sections.

Important: If you reboot a node that runs the DHCP agent, you must run the **neutron-ovs-cleanup** command before the `neutron-dhcp-agent` service starts.

On Red Hat, SUSE, and Ubuntu based systems, the `neutron-ovs-cleanup` service runs the **neutron-ovs-cleanup** command automatically. However, on Debian-based systems, you must manually run this command or write your own system script that runs on boot before the `neutron-dhcp-agent` service starts.

Networking dhcp-agent can use dnsmasq driver which supports stateful and stateless DHCPv6 for subnets created with `--ipv6_address_mode` set to `dhcpv6-stateful` or `dhcpv6-stateless`.

For example:

```
$ openstack subnet create --ip-version 6 --ipv6-ra-mode dhcpv6-stateful \
  --ipv6-address-mode dhcpv6-stateful --network NETWORK --subnet-range \
  CIDR SUBNET_NAME
```

```
$ openstack subnet create --ip-version 6 --ipv6-ra-mode dhcpv6-stateless \
  --ipv6-address-mode dhcpv6-stateless --network NETWORK --subnet-range \
  CIDR SUBNET_NAME
```

If no dnsmasq process for subnets network is launched, Networking will launch a new one on subnets dhcp port in `qdhcp-XXX` namespace. If previous dnsmasq process is already launched, restart dnsmasq with a new configuration.

Networking will update dnsmasq process and restart it when subnet gets updated.

Note: For dhcp-agent to operate in IPv6 mode use at least dnsmasq v2.63.

After a certain, configured timeframe, networks uncouple from DHCP agents when the agents are no longer in use. You can configure the DHCP agent to automatically detach from a network when the agent is out of service, or no longer needed.

This feature applies to all plug-ins that support DHCP scaling. For more information, see the DHCP agent configuration options listed in the OpenStack Configuration Reference.

DHCP agent setup: OVS plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the OVS plug-in:

```
[DEFAULT]
enable_isolated_metadata = True
interface_driver = openvswitch
```

DHCP agent setup: NSX plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the NSX plug-in:

```
[DEFAULT]
enable_metadata_network = True
enable_isolated_metadata = True
interface_driver = openvswitch
```

DHCP agent setup: Linux-bridge plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the Linux-bridge plug-in:

```
[DEFAULT]
enabled_isolated_metadata = True
interface_driver = linuxbridge
```

Configure L3 agent

The OpenStack Networking service has a widely used API extension to allow administrators and projects to create routers to interconnect L2 networks, and floating IPs to make ports on private networks publicly accessible.

Many plug-ins rely on the L3 service agent to implement the L3 functionality. However, the following plug-ins already have built-in L3 capabilities:

- Big Switch/Floodlight plug-in, which supports both the open source Floodlight controller and the proprietary Big Switch controller.

  Note: Only the proprietary BigSwitch controller implements L3 functionality. When using Floodlight as your OpenFlow controller, L3 functionality is not available.

- IBM SDN-VE plug-in
- MidoNet plug-in
- NSX plug-in
- PLUMgrid plug-in

Warning: Do not configure or use `neutron-l3-agent` if you use one of these plug-ins.

To install the L3 agent for all other plug-ins

1. Install the `neutron-l3-agent` binary on the network node:

```
# apt-get install neutron-l3-agent
```

2. To uplink the node that runs `neutron-l3-agent` to the external network, create a bridge named `br-ex` and attach the NIC for the external network to this bridge.

   For example, with Open vSwitch and NIC eth1 connected to the external network, run:

```
# ovs-vsctl add-br br-ex
# ovs-vsctl add-port br-ex eth1
```

   When the `br-ex` port is added to the `eth1` interface, external communication is interrupted. To avoid this, edit the `/etc/network/interfaces` file to contain the following information:

```
## External bridge
auto br-ex
iface br-ex inet static
address 192.27.117.101
netmask 255.255.240.0
gateway 192.27.127.254
dns-nameservers 8.8.8.8

## External network interface
auto eth1
iface eth1 inet manual
up ifconfig $IFACE 0.0.0.0 up
up ip link set $IFACE promisc on
down ip link set $IFACE promisc off
down ifconfig $IFACE down
```

   Note: The external bridge configuration address is the external IP address. This address and gateway should be configured in `/etc/network/interfaces`.

   After editing the configuration, restart `br-ex`:

```
# ifdown br-ex && ifup br-ex
```

   Do not manually configure an IP address on the NIC connected to the external network for the node running `neutron-l3-agent`. Rather, you must have a range of IP addresses from the external network that can be used by OpenStack Networking for routers that uplink to the external network. This range must be large enough to have an IP address for each router in the deployment, as well as each floating IP.

3. The `neutron-l3-agent` uses the Linux IP stack and iptables to perform L3 forwarding and NAT. In order to support multiple routers with potentially overlapping IP addresses, `neutron-l3-agent` defaults to using Linux network namespaces to provide isolated forwarding contexts. As a result, the IP addresses of routers are not visible simply by running the **ip**

addr list or **ifconfig** command on the node. Similarly, you cannot directly **ping** fixed IPs.

To do either of these things, you must run the command within a particular network namespace for the router. The namespace has the name `qrouter-ROUTER_UUID`. These example commands run in the router namespace with UUID 47af3868-0fa8-4447-85f6-1304de32153b:

```
# ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ip addr␣
↪list
```

```
# ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ping␣
↪FIXED_IP
```

Important: If you reboot a node that runs the L3 agent, you must run the **neutron-ovs-cleanup** command before the `neutron-l3-agent` service starts.

On Red Hat, SUSE and Ubuntu based systems, the neutron-ovs-cleanup service runs the **neutron-ovs-cleanup** command automatically. However, on Debian-based systems, you must manually run this command or write your own system script that runs on boot before the neutron-l3-agent service starts.

How routers are assigned to L3 agents By default, a router is assigned to the L3 agent with the least number of routers (LeastRoutersScheduler). This can be changed by altering the `router_scheduler_driver` setting in the configuration file.

Configure metering agent

The Neutron Metering agent resides beside neutron-l3-agent.

To install the metering agent and configure the node

1. Install the agent by running:

   ```
   # apt-get install neutron-metering-agent
   ```

2. If you use one of the following plug-ins, you need to configure the metering agent with these lines as well:

   - An OVS-based plug-in such as OVS, NSX, NEC, BigSwitch/Floodlight:

     ```
     interface_driver = openvswitch
     ```

   - A plug-in that uses LinuxBridge:

     ```
     interface_driver = linuxbridge
     ```

3. To use the reference implementation, you must set:

   ```
   driver = iptables
   ```

4. Set the `service_plugins` option in the `/etc/neutron/neutron.conf` file on the host that runs `neutron-server`:

```
service_plugins = metering
```

If this option is already defined, add `metering` to the list, using a comma as separator. For example:

```
service_plugins = router,metering
```

Configure Hyper-V L2 agent

Before you install the OpenStack Networking Hyper-V L2 agent on a Hyper-V compute node, ensure the compute node has been configured correctly using these instructions.

To install the OpenStack Networking Hyper-V agent and configure the node

1. Download the OpenStack Networking code from the repository:

```
> cd C:\OpenStack\
> git clone https://opendev.org/openstack/neutron
```

2. Install the OpenStack Networking Hyper-V Agent:

```
> cd C:\OpenStack\neutron\
> python setup.py install
```

3. Copy the `policy.json` file:

```
> xcopy C:\OpenStack\neutron\etc\policy.json C:\etc\
```

4. Create the `C:\etc\neutron-hyperv-agent.conf` file and add the proper configuration options and the Hyper-V related options. Here is a sample config file:

```
[DEFAULT]
control_exchange = neutron
policy_file = C:\etc\policy.json
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = IP_ADDRESS
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = <password>
logdir = C:\OpenStack\Log
logfile = neutron-hyperv-agent.log

[AGENT]
polling_interval = 2
physical_network_vswitch_mappings = *:YOUR_BRIDGE_NAME
enable_metrics_collection = true

[SECURITYGROUP]
firewall_driver = hyperv.neutron.security_groups_driver.
↪HyperVSecurityGroupsDriver
enable_security_group = true
```

5. Start the OpenStack Networking Hyper-V agent:

```
> C:\Python27\Scripts\neutron-hyperv-agent.exe --config-file
C:\etc\neutron-hyperv-agent.conf
```

Basic operations on agents

This table shows examples of Networking commands that enable you to complete basic operations on agents.

Operation	Command
List all available agents.	`$ openstack network agent list`
Show information of a given agent.	`$ openstack network agent show AGENT_ID`
Update the admin status and description for a specified agent. The command can be used to enable and disable agents by using `--admin-state-up` parameter set to `False` or `True`.	`$ neutron agent-update --admin-state-up False AGENT_ID`
Delete a given agent. Consider disabling the agent before deletion.	`$ openstack network agent delete AGENT_ID`

Basic operations on Networking agents

See the OpenStack Command-Line Interface Reference for more information on Networking commands.

8.8.5 Configure Identity service for Networking

To configure the Identity service for use with Networking

1. Create the `get_id()` function

   The `get_id()` function stores the ID of created objects, and removes the need to copy and paste object IDs in later steps:

   a. Add the following function to your `.bashrc` file:

   ```
   function get_id () {
   echo `"$@" | awk '/ id / { print $4 }'`
   }
   ```

   b. Source the `.bashrc` file:

   ```
   $ source .bashrc
   ```

2. Create the Networking service entry

   Networking must be available in the Compute service catalog. Create the service:

```
$ NEUTRON_SERVICE_ID=$(get_id openstack service create network \
  --name neutron --description 'OpenStack Networking Service')
```

3. Create the Networking service endpoint entry

The way that you create a Networking endpoint entry depends on whether you are using the SQL or the template catalog driver:

- If you are using the SQL driver, run the following command with the specified region ($REGION), IP address of the Networking server ($IP), and service ID ($NEUTRON_SERVICE_ID, obtained in the previous step).

```
$ openstack endpoint create $NEUTRON_SERVICE_ID --region $REGION \
  --publicurl 'http://$IP:9696/' --adminurl 'http://$IP:9696/' \
  --internalurl 'http://$IP:9696/'
```

  For example:

```
$ openstack endpoint create $NEUTRON_SERVICE_ID --region myregion␣
↪\
  --publicurl "http://10.211.55.17:9696/" \
  --adminurl "http://10.211.55.17:9696/" \
  --internalurl "http://10.211.55.17:9696/"
```

- If you are using the template driver, specify the following parameters in your Compute catalog template file (default_catalog.templates), along with the region ($REGION) and IP address of the Networking server ($IP).

```
catalog.$REGION.network.publicURL = http://$IP:9696
catalog.$REGION.network.adminURL = http://$IP:9696
catalog.$REGION.network.internalURL = http://$IP:9696
catalog.$REGION.network.name = Network Service
```

  For example:

```
catalog.$Region.network.publicURL = http://10.211.55.17:9696
catalog.$Region.network.adminURL = http://10.211.55.17:9696
catalog.$Region.network.internalURL = http://10.211.55.17:9696
catalog.$Region.network.name = Network Service
```

4. Create the Networking service user

   You must provide admin user credentials that Compute and some internal Networking components can use to access the Networking API. Create a special service project and a neutron user within this project, and assign an admin role to this role.

   a. Create the admin role:

```
$ ADMIN_ROLE=$(get_id openstack role create admin)
```

   b. Create the neutron user:

```
$ NEUTRON_USER=$(get_id openstack user create neutron \
  --password "$NEUTRON_PASSWORD" --email demo@example.com \
  --project service)
```

   c. Create the service project:

```
$ SERVICE_TENANT=$(get_id openstack project create service \
  --description "Services project" --domain default)
```

   d. Establish the relationship among the project, user, and role:

```
$ openstack role add $ADMIN_ROLE --user $NEUTRON_USER \
  --project $SERVICE_TENANT
```

For information about how to create service entries and users, see the Ocata Installation Tutorials and Guides for your distribution.

Compute

If you use Networking, do not run the Compute `nova-network` service (like you do in traditional Compute deployments). Instead, Compute delegates most network-related decisions to Networking.

Note: Uninstall `nova-network` and reboot any physical nodes that have been running `nova-network` before using them to run Networking. Inadvertently running the `nova-network` process while using Networking can cause problems, as can stale iptables rules pushed down by previously running `nova-network`.

Compute proxies project-facing API calls to manage security groups and floating IPs to Networking APIs. However, operator-facing tools such as `nova-manage`, are not proxied and should not be used.

Warning: When you configure networking, you must use this guide. Do not rely on Compute networking documentation or past experience with Compute. If a **nova** command or configuration option related to networking is not mentioned in this guide, the command is probably not supported for use with Networking. In particular, you cannot use CLI tools like `nova-manage` and `nova` to manage networks or IP addressing, including both fixed and floating IPs, with Networking.

To ensure that Compute works properly with Networking (rather than the legacy `nova-network` mechanism), you must adjust settings in the `nova.conf` configuration file.

Networking API and credential configuration

Each time you provision or de-provision a VM in Compute, `nova-\*` services communicate with Networking using the standard API. For this to happen, you must configure the following items in the `nova.conf` file (used by each `nova-compute` and `nova-api` instance).

Table 7: **nova.conf API and credential settings prior to Mitaka**

Attribute name	Required
[DEFAULT] use_neutron	Modify from the default to True to indicate that Networking should be used rather than the traditional nova-network networking model.
[neutron] url	Update to the host name/IP and port of the neutron-server instance for this deployment.
[neutron] auth_strategy	Keep the default keystone value for all production deployments.
[neutron] admin_project_name	Update to the name of the service tenant created in the above section on Identity configuration.
[neutron] admin_username	Update to the name of the user created in the above section on Identity configuration.
[neutron] admin_password	Update to the password of the user created in the above section on Identity configuration.
[neutron] admin_auth_url	Update to the Identity server IP and port. This is the Identity (keystone) admin API server IP and port value, and not the Identity service API IP and port.

Table 8: **nova.conf API and credential settings in Newton**

Attribute name	Required
[DEFAULT] use_neutron	Modify from the default to True to indicate that Networking should be used rather than the traditional nova-network networking model.
[neutron] url	Update to the host name/IP and port of the neutron-server instance for this deployment.
[neutron] auth_strategy	Keep the default keystone value for all production deployments.
[neutron] project_name	Update to the name of the service tenant created in the above section on Identity configuration.
[neutron] username	Update to the name of the user created in the above section on Identity configuration.
[neutron] password	Update to the password of the user created in the above section on Identity configuration.
[neutron] auth_url	Update to the Identity server IP and port. This is the Identity (keystone) admin API server IP and port value, and not the Identity service API IP and port.

Configure security groups

The Networking service provides security group functionality using a mechanism that is more flexible and powerful than the security group capabilities built into Compute. Therefore, if you use Networking, you should always disable built-in security groups and proxy all security group calls to the Networking API. If you do not, security policies will conflict by being simultaneously applied by both services.

To proxy security groups to Networking, use the following configuration values in the nova.conf file:

nova.conf security group settings

Item	Configuration
firewall_driver	Update to nova.virt.firewall.NoopFirewallDriver, so that nova-compute does not perform iptables-based filtering itself.

Configure metadata

The Compute service allows VMs to query metadata associated with a VM by making a web request to a special 169.254.169.254 address. Networking supports proxying those requests to nova-api, even when the requests are made from isolated networks, or from multiple networks that use overlapping IP addresses.

To enable proxying the requests, you must update the following fields in `[neutron]` section in the `nova.conf`.

nova.conf metadata settings

Item	Configuration
service_metadata_proxy	Update to true, otherwise nova-api will not properly respond to requests from the neutron-metadata-agent.
metadata_proxy_shared_secret	Update to a string password value. You must also configure the same value in the `metadata_agent.ini` file, to authenticate requests made for metadata. The default value of an empty string in both files will allow metadata to function, but will not be secure if any non-trusted entities have access to the metadata APIs exposed by nova-api.

Note: As a precaution, even when using `metadata_proxy_shared_secret`, we recommend that you do not expose metadata using the same nova-api instances that are used for projects. Instead, you should run a dedicated set of nova-api instances for metadata that are available only on your management network. Whether a given nova-api instance exposes metadata APIs is determined by the value of `enabled_apis` in its `nova.conf`.

Example nova.conf (for nova-compute and nova-api)

Example values for the above settings, assuming a cloud controller node running Compute and Networking with an IP address of 192.168.1.2:

```
[DEFAULT]
use_neutron = True
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
url=http://192.168.1.2:9696
auth_strategy=keystone
admin_tenant_name=service
admin_username=neutron
admin_password=password
admin_auth_url=http://192.168.1.2:5000/v2.0
service_metadata_proxy=true
metadata_proxy_shared_secret=foo
```

8.8.6 Advanced configuration options

This section describes advanced configuration options for various system components. For example, configuration options where the default works but that the user wants to customize options. After installing from packages, `$NEUTRON_CONF_DIR` is `/etc/neutron`.

L3 metering agent

You can run an L3 metering agent that enables layer-3 traffic metering. In general, you should launch the metering agent on all nodes that run the L3 agent:

```
$ neutron-metering-agent --config-file NEUTRON_CONFIG_FILE \
  --config-file L3_METERING_CONFIG_FILE
```

You must configure a driver that matches the plug-in that runs on the service. The driver adds metering to the routing interface.

Option	Value
Open vSwitch	
interface_driver ($NEUTRON_CONF_DIR/metering_agent.ini)	openvswitch
Linux Bridge	
interface_driver ($NEUTRON_CONF_DIR/metering_agent.ini)	linuxbridge

L3 metering driver

You must configure any driver that implements the metering abstraction. Currently the only available implementation uses iptables for metering.

```
driver = iptables
```

L3 metering service driver

To enable L3 metering, you must set the following option in the `neutron.conf` file on the host that runs `neutron-server`:

```
service_plugins = metering
```

8.8.7 Scalable and highly available DHCP agents

This section is fully described at the *High-availability for DHCP* in the Networking Guide.

8.8.8 Use Networking

You can manage OpenStack Networking services by using the service command. For example:

```
# service neutron-server stop
# service neutron-server status
# service neutron-server start
# service neutron-server restart
```

Log files are in the `/var/log/neutron` directory.

Configuration files are in the `/etc/neutron` directory.

Administrators and projects can use OpenStack Networking to build rich network topologies. Administrators can create network connectivity on behalf of projects.

Core Networking API features

After installing and configuring Networking (neutron), projects and administrators can perform create-read-update-delete (CRUD) API networking operations. This is performed using the Networking API directly with either the **neutron** command-line interface (CLI) or the **openstack** CLI. The **neutron** CLI is a wrapper around the Networking API. Every Networking API call has a corresponding **neutron** command.

The **openstack** CLI is a common interface for all OpenStack projects, however, not every API operation has been implemented. For the list of available commands, see Command List.

The **neutron** CLI includes a number of options. For details, see Create and manage networks.

Basic Networking operations

To learn about advanced capabilities available through the **neutron** command-line interface (CLI), read the networking section Create and manage networks in the OpenStack End User Guide.

This table shows example **openstack** commands that enable you to complete basic network operations:

Operation	Command
Creates a network.	`$ openstack network create net1`
Creates a subnet that is associated with net1.	`$ openstack subnet create subnet1 --subnet-range 10.0.0.0/24 --network net1`
Lists ports for a specified project.	`$ openstack port list`
Lists ports for a specified project and displays the `ID`, `Fixed IP Addresses`	`$ openstack port list -c ID -c "Fixed IP Addresses`
Shows information for a specified port.	`$ openstack port show PORT_ID`

Basic Networking operations

Note: The `device_owner` field describes who owns the port. A port whose `device_owner` begins with:

- `network` is created by Networking.
- `compute` is created by Compute.

Administrative operations

The administrator can run any **openstack** command on behalf of projects by specifying an Identity `project` in the command, as follows:

```
$ openstack network create --project PROJECT_ID NETWORK_NAME
```

For example:

```
$ openstack network create --project 5e4bbe24b67a4410bc4d9fae29ec394e net1
```

Note: To view all project IDs in Identity, run the following command as an Identity service admin user:

```
$ openstack project list
```

Advanced Networking operations

This table shows example CLI commands that enable you to complete advanced network operations:

Operation	Command
Creates a network that all projects can use.	`$ openstack network create --share public-net`
Creates a subnet with a specified gateway IP address.	`$ openstack subnet create subnet1 --gateway 10.0.0.254 --network net1`
Creates a subnet that has no gateway IP address.	`$ openstack subnet create subnet1 --no-gateway --network net1`
Creates a subnet with DHCP disabled.	`$ openstack subnet create subnet1 --network net1 --no-dhcp`
Specifies a set of host routes	`$ openstack subnet create subnet1 --network net1 --host-route destination=40.0.1.0/24, gateway=40.0.0.2`
Creates a subnet with a specified set of dns name servers.	`$ openstack subnet create subnet1 --network net1 --dns-nameserver 8.8.4.4`
Displays all ports and IPs allocated on a network.	`$ openstack port list --network NET_ID`

Advanced Networking operations

Note: During port creation and update, specific extra-dhcp-options can be left blank. For example, `router` and `classless-static-route`. This causes dnsmasq to have an empty option in the `opts` file related to the network. For example:

```
tag:tag0,option:classless-static-route,
tag:tag0,option:router,
```

Use Compute with Networking

Basic Compute and Networking operations

This table shows example **openstack** commands that enable you to complete basic VM networking operations:

Action	Command
Checks available networks.	`$ openstack network list`
Boots a VM with a single NIC on a selected Networking network.	`$ openstack server create --image IMAGE --flavor FLAVOR --nic net-id=NET_ID VM_NAME`
Searches for ports with a `device_id` that matches the Compute instance UUID. See :ref: *Create and delete VMs*	`$ openstack port list --server VM_ID`
Searches for ports, but shows only the `mac_address` of the port.	`$ openstack port list -c "MAC Address" --server VM_ID`
Temporarily disables a port from sending traffic.	`$ openstack port set PORT_ID --disable`

Basic Compute and Networking operations

Note: The `device_id` can also be a logical router ID.

Note:

- When you boot a Compute VM, a port on the network that corresponds to the VM NIC is automatically created and associated with the default security group. You can configure *security group rules* to enable users to access the VM.

Advanced VM creation operations

This table shows example **openstack** commands that enable you to complete advanced VM creation operations:

Operation	Command
Boots a VM with multiple NICs.	`$ openstack server create --image IMAGE --flavor FLAVOR --nic net-id=NET_ID VM_NAME net-id=NET2-ID VM_NAME`
Boots a VM with a specific IP address. Note that you cannot use the `--max` or `--min` parameters in this case.	`$ openstack server create --image IMAGE --flavor FLAVOR --nic net-id=NET_ID VM_NAME v4-fixed-ip=IP-ADDR VM_NAME`
Boots a VM that connects to all networks that are accessible to the project who submits the request (without the `--nic` option).	`$ openstack server create --image IMAGE --flavor FLAVOR`

Advanced VM creation operations

Note: Cloud images that distribution vendors offer usually have only one active NIC configured. When you boot with multiple NICs, you must configure additional interfaces on the image or the NICs are not reachable.

The following Debian/Ubuntu-based example shows how to set up the interfaces within the instance in the `/etc/network/interfaces` file. You must apply this configuration to the image.

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

Enable ping and SSH on VMs (security groups)

You must configure security group rules depending on the type of plug-in you are using. If you are using a plug-in that:

- Implements Networking security groups, you can configure security group rules directly by using the **`openstack security group rule create`** command. This example enables `ping` and `ssh` access to your VMs.

  ```
  $ openstack security group rule create --protocol icmp \
    --ingress SECURITY_GROUP
  ```

  ```
  $ openstack security group rule create --protocol tcp \
    --egress --description "Sample Security Group" SECURITY_GROUP
  ```

- Does not implement Networking security groups, you can configure security group rules by using the **`openstack security group rule create`** or **`euca-authorize`** command.

These **openstack** commands enable `ping` and `ssh` access to your VMs.

```
$ openstack security group rule create --protocol icmp default
$ openstack security group rule create --protocol tcp --dst-port␣
↪22:22 default
```

Note: If your plug-in implements Networking security groups, you can also leverage Compute security groups by setting `use_neutron = True` in the `nova.conf` file. After you set this option, all Compute security group commands are proxied to Networking.

8.8.9 Advanced features through API extensions

Several plug-ins implement API extensions that provide capabilities similar to what was available in `nova-network`. These plug-ins are likely to be of interest to the OpenStack community.

Provider networks

Networks can be categorized as either project networks or provider networks. Project networks are created by normal users and details about how they are physically realized are hidden from those users. Provider networks are created with administrative credentials, specifying the details of how the network is physically realized, usually to match some existing network in the data center.

Provider networks enable administrators to create networks that map directly to the physical networks in the data center. This is commonly used to give projects direct access to a public network that can be used to reach the Internet. It might also be used to integrate with VLANs in the network that already have a defined meaning (for example, enable a VM from the marketing department to be placed on the same VLAN as bare-metal marketing hosts in the same data center).

The provider extension allows administrators to explicitly manage the relationship between Networking virtual networks and underlying physical mechanisms such as VLANs and tunnels. When this extension is supported, Networking client users with administrative privileges see additional provider attributes on all virtual networks and are able to specify these attributes in order to create provider networks.

The provider extension is supported by the Open vSwitch and Linux Bridge plug-ins. Configuration of these plug-ins requires familiarity with this extension.

Terminology

A number of terms are used in the provider extension and in the configuration of plug-ins supporting the provider extension:

Table 9: **Provider extension terminology**

Term	Description
virtual network	A Networking L2 network (identified by a UUID and optional name) whose ports can be attached as vNICs to Compute instances and to various Networking agents. The Open vSwitch and Linux Bridge plug-ins each support several different mechanisms to realize virtual networks.
physical network	A network connecting virtualization hosts (such as compute nodes) with each other and with other network resources. Each physical network might support multiple virtual networks. The provider extension and the plug-in configurations identify physical networks using simple string names.
project network	A virtual network that a project or an administrator creates. The physical details of the network are not exposed to the project.
provider network	A virtual network administratively created to map to a specific network in the data center, typically to enable direct access to non-OpenStack resources on that network. Project can be given access to provider networks.
VLAN network	A virtual network implemented as packets on a specific physical network containing IEEE 802.1Q headers with a specific VID field value. VLAN networks sharing the same physical network are isolated from each other at L2 and can even have overlapping IP address spaces. Each distinct physical network supporting VLAN networks is treated as a separate VLAN trunk, with a distinct space of VID values. Valid VID values are 1 through 4094.
flat network	A virtual network implemented as packets on a specific physical network containing no IEEE 802.1Q header. Each physical network can realize at most one flat network.
local network	A virtual network that allows communication within each host, but not across a network. Local networks are intended mainly for single-node test scenarios, but can have other uses.
GRE network	A virtual network implemented as network packets encapsulated using GRE. GRE networks are also referred to as *tunnels*. GRE tunnel packets are routed by the IP routing table for the host, so GRE networks are not associated by Networking with specific physical networks.
Virtual Extensible LAN (VXLAN) network	VXLAN is a proposed encapsulation protocol for running an overlay network on existing Layer 3 infrastructure. An overlay network is a virtual network that is built on top of existing network Layer 2 and Layer 3 technologies to support elastic compute architectures.

The ML2, Open vSwitch, and Linux Bridge plug-ins support VLAN networks, flat networks, and local networks. Only the ML2 and Open vSwitch plug-ins currently support GRE and VXLAN networks, provided that the required features exist in the hosts Linux kernel, Open vSwitch, and iproute2 packages.

Provider attributes

The provider extension extends the Networking network resource with these attributes:

Table 10: **Provider network attributes**

Attribute name	Type	Default Value	Description
provider:network_type	String	N/A	The physical mechanism by which the virtual network is implemented. Possible values are `flat`, `vlan`, `local`, `gre`, and `vxlan`, corresponding to flat networks, VLAN networks, local networks, GRE networks, and VXLAN networks as defined above. All types of provider networks can be created by administrators, while project networks can be implemented as `vlan`, `gre`, `vxlan`, or `local` network types depending on plug-in configuration.
provider:physical_network	String	If a physical network named default has been configured and if provider:network_type is `flat` or `vlan`, then default is used.	The name of the physical network over which the virtual network is implemented for flat and VLAN networks. Not applicable to the `local`, `vxlan` or `gre` network types.
provider:segmentation_id	Integer	N/A	For VLAN networks, the VLAN VID on the physical network that realizes the virtual network. Valid VLAN VIDs are 1 through 4094. For GRE networks, the tunnel ID. Valid tunnel IDs are any 32 bit unsigned integer. Not applicable to the `flat` or `local` network types.

To view or set provider extended attributes, a client must be authorized for the `extension:provider_network:view` and `extension:provider_network:set` actions in the Networking policy configuration. The default Networking configuration authorizes both actions for users with the admin role. An authorized client or an administrative user can view and set the provider extended attributes through Networking API calls. See the section called *Authentication and authorization* for details on policy configuration.

L3 routing and NAT

The Networking API provides abstract L2 network segments that are decoupled from the technology used to implement the L2 network. Networking includes an API extension that provides abstract L3 routers that API users can dynamically provision and configure. These Networking routers can connect multiple L2 Networking networks and can also provide a gateway that connects one or more private L2 networks to a shared external network. For example, a public network for access to the Internet. See the OpenStack Configuration Reference for details on common models of deploying Networking L3 routers.

The L3 router provides basic NAT capabilities on gateway ports that uplink the router to external networks. This router SNATs all traffic by default and supports floating IPs, which creates a static one-to-one mapping from a public IP on the external network to a private IP on one of the other subnets attached to the router. This allows a project to selectively expose VMs on private networks to other hosts on the external network (and often to all hosts on the Internet). You can allocate and map floating IPs from one port to another, as needed.

Basic L3 operations

External networks are visible to all users. However, the default policy settings enable only administrative users to create, update, and delete external networks.

This table shows example **openstack** commands that enable you to complete basic L3 operations:

Table 11: **Basic L3 Operations**

Operation	Command
Creates external networks.	`$ openstack network create public --external` `$ openstack subnet create --network public --subnet-range 172.16.1.0/24 subnetname`
Lists external networks.	`$ openstack network list --external`

Creates an internal-only router that connects to multiple L2 networks privately.	`$ openstack network create net1`
	`$ openstack subnet create --network net1 --subnet-range 10.0.0.0/24 subnetname1`
	`$ openstack network create net2`
	`$ openstack subnet create --network net2 --subnet-range 10.0.1.0/24 subnetname2`
	`$ openstack router create router1`
	`$ openstack router add subnet router1 subnetname1`
	`$ openstack router add subnet router1 subnetname2`
	An internal router port can have only one IPv4 subnet and multiple IPv6 subnets that belong to the same network ID. When you call `router-interface-add` with an IPv6 subnet, this operation adds the interface to an existing internal port with the same network ID. If a port with the same network ID does not exist, a new port is created.

Connects a router to an external network, which enables that router to act as a NAT gateway for external connectivity.	`$ openstack router set --external-gateway EXT_NET_ID router1`
	`$ openstack router set --route destination=172.24.4.0/24,gateway=172.24.4.1 router1`
	The router obtains an interface with the gateway_ip address of the subnet and this interface is attached to a port on the L2 Networking network associated with the subnet. The router also gets a gateway interface to the specified external network. This provides SNAT connectivity to the external network as well as support for floating IPs allocated on that external networks. Commonly an external network maps to a network in the provider.

Lists routers.	`$ openstack router list`
Shows information for a specified router.	`$ openstack router show ROUTER_ID`
Shows all internal interfaces for a router.	`$ openstack port list --router  ROUTER_ID` `$ openstack port list --router  ROUTER_NAME`
Identifies the PORT_ID that represents the VM NIC to which the floating IP should map.	`$ openstack port list -c ID -c "Fixed IP Addresses" --server INSTANCE_ID` This port must be on a Networking subnet that is attached to a router uplinked to the external network used to create the floating IP. Conceptually, this is because the router must be able

Security groups

Security groups and security group rules allow administrators and projects to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a port. A security group is a container for security group rules.

When a port is created in Networking it is associated with a security group. If a security group is not specified the port is associated with a default security group. By default, this group drops all ingress traffic and allows all egress. Rules can be added to this group in order to change the behavior.

To use the Compute security group APIs or use Compute to orchestrate the creation of ports for instances on specific security groups, you must complete additional configuration. You must configure the `/etc/nova/nova.conf` file and set the `use_neutron=True` option on every node that runs nova-compute, nova-conductor and nova-api. After you make this change, restart those nova services to pick up this change. Then, you can use both the Compute and OpenStack Network security group APIs at the same time.

Note:

- To use the Compute security group API with Networking, the Networking plug-in must implement the security group API. The following plug-ins currently implement this: ML2, Open vSwitch, Linux Bridge, NEC, and VMware NSX.
- You must configure the correct firewall driver in the `securitygroup` section of the plug-in/agent configuration file. Some plug-ins and agents, such as Linux Bridge Agent and Open vSwitch Agent, use the no-operation driver as the default, which results in non-working security groups.
- When using the security group API through Compute, security groups are applied to all ports on an instance. The reason for this is that Compute security group APIs are instances based and not port based as Networking.
- When creating or updating a port with a specified security group, the admin tenant can use the security groups of other tenants.

Basic security group operations

This table shows example neutron commands that enable you to complete basic security group operations:

Table 12: **Basic security group operations**

Operation	Command
Creates a security group for our web servers.	`$ openstack security group create webservers ↪\ --description "security group for webservers ↪"`
Lists security groups.	`$ openstack security group list`
Creates a security group rule to allow port 80 ingress.	`$ openstack security group rule create --↪ingress \ --protocol tcp SECURITY_GROUP_UUID`
Lists security group rules.	`$ openstack security group rule list`
Deletes a security group rule.	`$ openstack security group rule delete ↪SECURITY_GROUP_RULE_UUID`
Deletes a security group.	`$ openstack security group delete SECURITY_↪GROUP_UUID`
Creates a port and associates two security groups.	`$ openstack port create port1 --security-↪group SECURITY_GROUP_ID1 \ --security-group SECURITY_GROUP_ID2 --↪network NETWORK_ID`
Removes security groups from a port.	`$ openstack port set --no-security-group ↪PORT_ID`

Plug-in specific extensions

Each vendor can choose to implement additional API extensions to the core API. This section describes the extensions for each plug-in.

VMware NSX extensions

These sections explain NSX plug-in extensions.

VMware NSX QoS extension

The VMware NSX QoS extension rate-limits network ports to guarantee a specific amount of bandwidth for each port. This extension, by default, is only accessible by a project with an admin role but is configurable through the `policy.json` file. To use this extension, create a queue and specify the min/max bandwidth rates (kbps) and optionally set the QoS Marking and DSCP value (if your network fabric uses these values to make forwarding decisions). Once created, you can associate a queue with a network. Then, when ports are created on that network they are automatically created and associated with the specific queue size that was associated with the network. Because one size queue for a every port on a network might not be optimal, a scaling factor from the nova flavor `rxtx_factor` is passed in from Compute when creating the port to scale the queue.

Lastly, if you want to set a specific baseline QoS policy for the amount of bandwidth a single port can use (unless a network queue is specified with the network a port is created on) a default queue can be created in Networking which then causes ports created to be associated with a queue of that size times the rxtx scaling factor. Note that after a network or default queue is specified, queues are added to ports that are subsequently created but are not added to existing ports.

Basic VMware NSX QoS operations

This table shows example neutron commands that enable you to complete basic queue operations:

Table 13: **Basic VMware NSX QoS operations**

Operation	Command
Creates QoS queue (admin-only).	`$ neutron queue-create --min 10 --max 1000 myqueue`
Associates a queue with a network.	`$ neutron net-create network --queue_id QUEUE_ID`
Creates a default system queue.	`$ neutron queue-create --default True --min 10 --max 2000 default`
Lists QoS queues.	`$ neutron queue-list`
Deletes a QoS queue.	`$ neutron queue-delete QUEUE_ID_OR_NAME`

VMware NSX provider networks extension

Provider networks can be implemented in different ways by the underlying NSX platform.

The *FLAT* and *VLAN* network types use bridged transport connectors. These network types enable the attachment of large number of ports. To handle the increased scale, the NSX plug-in can back a single OpenStack Network with a chain of NSX logical switches. You can specify the maximum number of ports on each logical switch in this chain on the `max_lp_per_bridged_ls` parameter, which has a default value of 5,000.

The recommended value for this parameter varies with the NSX version running in the back-end, as shown in the following table.

Recommended values for max_lp_per_bridged_ls

NSX version	Recommended Value
2.x	64
3.0.x	5,000
3.1.x	5,000
3.2.x	10,000

In addition to these network types, the NSX plug-in also supports a special *l3_ext* network type, which maps external networks to specific NSX gateway services as discussed in the next section.

VMware NSX L3 extension

NSX exposes its L3 capabilities through gateway services which are usually configured out of band from OpenStack. To use NSX with L3 capabilities, first create an L3 gateway service in the NSX Manager. Next, in `/etc/neutron/plugins/vmware/nsx.ini` set `default_l3_gw_service_uuid` to this value. By default, routers are mapped to this gateway service.

VMware NSX L3 extension operations

Create external network and map it to a specific NSX gateway service:

```
$ openstack network create public --external --provider-network-type l3_
↪ext \
--provider-physical-network L3_GATEWAY_SERVICE_UUID
```

Terminate traffic on a specific VLAN from a NSX gateway service:

```
$ openstack network create public --external --provider-network-type l3_
↪ext \
--provider-physical-network L3_GATEWAY_SERVICE_UUID --provider-segment␣
↪VLAN_ID
```

Operational status synchronization in the VMware NSX plug-in

Starting with the Havana release, the VMware NSX plug-in provides an asynchronous mechanism for retrieving the operational status for neutron resources from the NSX back-end; this applies to *network*, *port*, and *router* resources.

The back-end is polled periodically and the status for every resource is retrieved; then the status in the Networking database is updated only for the resources for which a status change occurred. As operational status is now retrieved asynchronously, performance for `GET` operations is consistently improved.

Data to retrieve from the back-end are divided in chunks in order to avoid expensive API requests; this is achieved leveraging NSX APIs response paging capabilities. The minimum chunk size can be specified using a configuration option; the actual chunk size is then determined dynamically according to: total number of resources to retrieve, interval between two synchronization task runs, minimum delay between two subsequent requests to the NSX back-end.

The operational status synchronization can be tuned or disabled using the configuration options reported in this table; it is however worth noting that the default values work fine in most cases.

Table 14: **Configuration options for tuning operational status synchronization in the NSX plug-in**

Option name	Group	Default value	Type and constraints	Notes
state_sync_interval	nsx_sync	10 seconds	Integer; no constraint.	Interval in seconds between two run of the synchronization task. If the synchronization task takes more than state_sync_interval seconds to execute, a new instance of the task is started as soon as the other is completed. Setting the value for this option to 0 will disable the synchronization task.
max_random_sync_delay	nsx_sync	0 seconds	Integer. Must not exceed min_sync_req_delay	When different from zero, a random delay between 0 and max_random_sync_delay will be added before processing the next chunk.
min_sync_req_delay	nsx_sync	1 second	Integer. Must not exceed state_sync_interval.	The value of this option can be tuned according to the observed load on the NSX controllers. Lower values will result in faster synchronization, but might increase the load on the controller cluster.
min_chunk_size	nsx_sync	500 resources	Integer; no constraint.	Minimum number of resources to retrieve from the back-end for each synchronization chunk. The expected number of synchronization chunks is given by the ratio between state_sync_interval and min_sync_req_delay. This size of a chunk might increase if the total number of resources is such that more than min_chunk_size resources must be fetched in one chunk with the current number of chunks.
always_read_status	nsx_sync	False	Boolean; no constraint.	When this option is enabled, the operational status will always be retrieved from the NSX back-end ad every GET request. In this case it is advisable to disable the synchronization task.

When running multiple OpenStack Networking server instances, the status synchronization task should not run on every node; doing so sends unnecessary traffic to the NSX back-end and performs unnecessary DB operations. Set the `state_sync_interval` configuration option to a non-zero value exclusively on a node designated for back-end status synchronization.

The `fields=status` parameter in Networking API requests always triggers an explicit query to the NSX back end, even when you enable asynchronous state synchronization. For example, `GET /v2.0/networks/NET_ID?fields=status&fields=name`.

Big Switch plug-in extensions

This section explains the Big Switch neutron plug-in-specific extension.

Big Switch router rules

Big Switch allows router rules to be added to each project router. These rules can be used to enforce routing policies such as denying traffic between subnets or traffic to external networks. By enforcing these at the router level, network segmentation policies can be enforced across many VMs that have differing security groups.

Router rule attributes

Each project router has a set of router rules associated with it. Each router rule has the attributes in this table. Router rules and their attributes can be set using the **`neutron router-update`** command, through the horizon interface or the Networking API.

Table 15: **Big Switch Router rule attributes**

Attribute name	Required	Input type	Description
source	Yes	A valid CIDR or one of the keywords any or external	The network that a packets source IP must match for the rule to be applied.
destination	Yes	A valid CIDR or one of the keywords any or external	The network that a packets destination IP must match for the rule to be applied.
action	Yes	permit or deny	Determines whether or not the matched packets will allowed to cross the router.
nexthop	No	A plus-separated (+) list of next-hop IP addresses. For example, `1.1.1.1+1.1.1.2`.	Overrides the default virtual router used to handle traffic for packets that match the rule.

Order of rule processing

The order of router rules has no effect. Overlapping rules are evaluated using longest prefix matching on the source and destination fields. The source field is matched first so it always takes higher precedence over the destination field. In other words, longest prefix matching is used on the destination field only if there are multiple matching rules with the same source.

Big Switch router rules operations

Router rules are configured with a router update operation in OpenStack Networking. The update overrides any previous rules so all rules must be provided at the same time.

Update a router with rules to permit traffic by default but block traffic from external networks to the 10.10.10.0/24 subnet:

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true \
  source=any,destination=any,action=permit \
  source=external,destination=10.10.10.0/24,action=deny
```

Specify alternate next-hop addresses for a specific subnet:

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true  \
  source=any,destination=any,action=permit \
  source=10.10.10.0/24,destination=any,action=permit,nexthops=10.10.10.
↪254+10.10.10.253
```

Block traffic between two subnets while allowing everything else:

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true \
  source=any,destination=any,action=permit \
  source=10.10.10.0/24,destination=10.20.20.20/24,action=deny
```

L3 metering

The L3 metering API extension enables administrators to configure IP ranges and assign a specified label to them to be able to measure traffic that goes through a virtual router.

The L3 metering extension is decoupled from the technology that implements the measurement. Two abstractions have been added: One is the metering label that can contain metering rules. Because a metering label is associated with a project, all virtual routers in this project are associated with this label.

Basic L3 metering operations

Only administrators can manage the L3 metering labels and rules.

This table shows example **neutron** commands that enable you to complete basic L3 metering operations:

Table 16: **Basic L3 operations**

<table>
<tr><th>Operation</th><th>Command</th></tr>
<tr><td>Creates a metering label.</td><td><pre>$ openstack network meter label create LABEL1 \
  --description "DESCRIPTION_LABEL1"</pre></td></tr>
<tr><td>Lists metering labels.</td><td><pre>$ openstack network meter label list</pre></td></tr>
<tr><td>Shows information for a specified label.</td><td><pre>$ openstack network meter label show LABEL_UUID
$ openstack network meter label show LABEL1</pre></td></tr>
<tr><td>Deletes a metering label.</td><td><pre>$ openstack network meter label delete LABEL_UUID
$ openstack network meter label delete LABEL1</pre></td></tr>
<tr><td>Creates a metering rule.</td><td><pre>$ openstack network meter label rule create LABEL_UUID \
  --remote-ip-prefix CIDR \
  --direction DIRECTION --exclude</pre>For example:<pre>$ openstack network meter label rule create label1 \
  --remote-ip-prefix 10.0.0.0/24 --direction ingress
$ openstack network meter label rule create label1 \
  --remote-ip-prefix 20.0.0.0/24 --exclude</pre></td></tr>
<tr><td>Lists metering all label rules.</td><td><pre>$ openstack network meter label rule list</pre></td></tr>
<tr><td>Shows information for a specified label rule.</td><td><pre>$ openstack network meter label rule show RULE_UUID</pre></td></tr>
<tr><td>Deletes a metering label rule.</td><td><pre>$ openstack network meter label rule delete RULE_UUID</pre></td></tr>
<tr><td>Lists the value of created metering label rules.</td><td><pre>$ ceilometer sample-list -m SNMP_MEASUREMENT</pre>For example:<pre>$ ceilometer sample-list -m hardware.network.bandwidth.bytes
$ ceilometer sample-list -m hardware.network.incoming.bytes
$ ceilometer sample-list -m hardware.network.outgoing.bytes
$ ceilometer sample-list -m hardware.network.outgoing.errors</pre></td></tr>
</table>

8.8.10 Advanced operational features

Logging settings

Networking components use Python logging module to do logging. Logging configuration can be provided in `neutron.conf` or as command-line options. Command options override ones in `neutron.conf`.

To configure logging for Networking components, use one of these methods:

- Provide logging settings in a logging configuration file.

  See Python logging how-to to learn more about logging.

- Provide logging setting in `neutron.conf`.

```
[DEFAULT]
# Default log level is WARNING
# Show debugging output in logs (sets DEBUG log level output)
# debug = False

# log_date_format = %Y-%m-%d %H:%M:%S

# use_syslog = False
# syslog_log_facility = LOG_USER

# if use_syslog is False, we can set log_file and log_dir.
# if use_syslog is False and we do not set log_file,
# the log will be printed to stdout.
# log_file =
# log_dir =
```

Notifications

Notifications can be sent when Networking resources such as network, subnet and port are created, updated or deleted.

Notification options

To support DHCP agent, `rpc_notifier` driver must be set. To set up the notification, edit notification options in `neutron.conf`:

```
# Driver or drivers to handle sending notifications. (multi
# valued)
# notification_driver=messagingv2

# AMQP topic used for OpenStack notifications. (list value)
# Deprecated group/name - [rpc_notifier2]/topics
notification_topics = notifications
```

Setting cases

Logging and RPC

These options configure the Networking server to send notifications through logging and RPC. The logging options are described in OpenStack Configuration Reference . RPC notifications go to `notifications.info` queue bound to a topic exchange defined by `control_exchange` in `neutron.conf`.

Notification System Options

A notification can be sent when a network, subnet, or port is created, updated or deleted. The notification system options are:

- **notification_driver** Defines the driver or drivers to handle the sending of a notification. The six available options are:
  - **messaging** Send notifications using the 1.0 message format.
  - **messagingv2** Send notifications using the 2.0 message format (with a message envelope).
  - **routing** Configurable routing notifier (by priority or event_type).
  - **log** Publish notifications using Python logging infrastructure.
  - **test** Store notifications in memory for test verification.
  - **noop** Disable sending notifications entirely.
- **default_notification_level** Is used to form topic names or to set a logging level.
- **default_publisher_id** Is a part of the notification payload.
- **notification_topics** AMQP topic used for OpenStack notifications. They can be comma-separated values. The actual topic names will be the values of `default_notification_level`.
- **control_exchange** This is an option defined in oslo.messaging. It is the default exchange under which topics are scoped. May be overridden by an exchange name specified in the `transport_url` option. It is a string value.

Below is a sample `neutron.conf` configuration file:

```
notification_driver = messagingv2

default_notification_level = INFO

host = myhost.com
default_publisher_id = $host

notification_topics = notifications

control_exchange = openstack
```

8.8.11 Authentication and authorization

Networking uses the Identity service as the default authentication service. When the Identity service is enabled, users who submit requests to the Networking service must provide an authentication token in `X-Auth-Token` request header. Users obtain this token by authenticating with the Identity service endpoint. For more information about authentication with the Identity service, see OpenStack Identity service API v3 Reference. When the Identity service is enabled, it is not mandatory to specify the project ID for resources in create requests because the project ID is derived from the authentication token.

The default authorization settings only allow administrative users to create resources on behalf of a different project. Networking uses information received from Identity to authorize user requests. Networking handles two kind of authorization policies:

- **Operation-based** policies specify access criteria for specific operations, possibly with fine-grained control over specific attributes.
- **Resource-based** policies specify whether access to specific resource is granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in Networking might vary from deployment to deployment.

The policy engine reads entries from the `policy.json` file. The actual location of this file might vary from distribution to distribution. Entries can be updated while the system is running, and no service restart is required. Every time the policy file is updated, the policies are automatically reloaded. Currently the only way of updating such policies is to edit the policy file. In this section, the terms *policy* and *rule* refer to objects that are specified in the same way in the policy file. There are no syntax differences between a rule and a policy. A policy is something that is matched directly from the Networking policy engine. A rule is an element in a policy, which is evaluated. For instance in `"create_subnet": "rule:admin_or_network_owner"`, *create_subnet* is a policy, and *admin_or_network_owner* is a rule.

Policies are triggered by the Networking policy engine whenever one of them matches a Networking API operation or a specific attribute being used in a given operation. For instance the `create_subnet` policy is triggered every time a `POST /v2.0/subnets` request is sent to the Networking server; on the other hand `create_network:shared` is triggered every time the *shared* attribute is explicitly specified (and set to a value different from its default) in a `POST /v2.0/networks` request. It is also worth mentioning that policies can also be related to specific API extensions; for instance `extension:provider_network:set` is triggered if the attributes defined by the Provider Network extensions are specified in an API request.

An authorization policy can be composed by one or more rules. If more rules are specified then the evaluation policy succeeds if any of the rules evaluates successfully; if an API operation matches multiple policies, then all the policies must evaluate successfully. Also, authorization rules are recursive. Once a rule is matched, the rule(s) can be resolved to another rule, until a terminal rule is reached.

The Networking policy engine currently defines the following kinds of terminal rules:

- **Role-based rules** evaluate successfully if the user who submits the request has the specified role. For instance `"role:admin"` is successful if the user who submits the request is an administrator.
- **Field-based rules** evaluate successfully if a field of the resource specified in the current request matches a specific value. For instance `"field:networks:shared=True"` is successful if the `shared` attribute of the `network` resource is set to true.
- **Generic rules** compare an attribute in the resource with an attribute extracted from the users

security credentials and evaluates successfully if the comparison is successful. For instance `"tenant_id:%(tenant_id)s"` is successful if the project identifier in the resource is equal to the project identifier of the user submitting the request.

This extract is from the default `policy.json` file:

- A rule that evaluates successfully if the current user is an administrator or the owner of the resource specified in the request (project identifier is equal).

```
{
 "admin_or_owner": "role:admin",
 "tenant_id:%(tenant_id)s",
 "admin_or_network_owner": "role:admin",
 "tenant_id:%(network_tenant_id)s",
 "admin_only": "role:admin",
 "regular_user": "",
 "shared":"field:networks:shared=True",
 "default":
```

- The default policy that is always evaluated if an API operation does not match any of the policies in `policy.json`.

```
"rule:admin_or_owner",
"create_subnet": "rule:admin_or_network_owner",
"get_subnet": "rule:admin_or_owner",
"rule:shared",
"update_subnet": "rule:admin_or_network_owner",
"delete_subnet": "rule:admin_or_network_owner",
"create_network": "",
"get_network": "rule:admin_or_owner",
```

- This policy evaluates successfully if either *admin_or_owner*, or *shared* evaluates successfully.

```
"rule:shared",
"create_network:shared": "rule:admin_only"
```

- This policy restricts the ability to manipulate the *shared* attribute for a network to administrators only.

```
 ,
 "update_network": "rule:admin_or_owner",
"delete_network": "rule:admin_or_owner",
"create_port": "",
"create_port:mac_address": "rule:admin_or_network_owner",
"create_port:fixed_ips":
```

- This policy restricts the ability to manipulate the *mac_address* attribute for a port only to administrators and the owner of the network where the port is attached.

```
  "rule:admin_or_network_owner",
 "get_port": "rule:admin_or_owner",
 "update_port": "rule:admin_or_owner",
  "delete_port": "rule:admin_or_owner"
}
```

In some cases, some operations are restricted to administrators only. This example shows you how to modify a policy file to permit project to define networks, see their resources, and permit administrative users to perform all other operations:

```
{
        "admin_or_owner": "role:admin", "tenant_id:%(tenant_id)s",
        "admin_only": "role:admin", "regular_user": "",
        "default": "rule:admin_only",
        "create_subnet": "rule:admin_only",
        "get_subnet": "rule:admin_or_owner",
        "update_subnet": "rule:admin_only",
        "delete_subnet": "rule:admin_only",
        "create_network": "",
        "get_network": "rule:admin_or_owner",
        "create_network:shared": "rule:admin_only",
        "update_network": "rule:admin_or_owner",
        "delete_network": "rule:admin_or_owner",
        "create_port": "rule:admin_only",
        "get_port": "rule:admin_or_owner",
        "update_port": "rule:admin_only",
        "delete_port": "rule:admin_only"
}
```

CHAPTER

NINE

CONFIGURATION GUIDE

9.1 Configuration Reference

This section provides a list of all configuration options for various neutron services. These are auto-generated from neutron code when this documentation is built.

Configuration filenames used below are filenames usually used, but there is no restriction on configuration filename in neutron and you can use arbitrary file names.

9.1.1 neutron.conf

DEFAULT

state_path

> **Type** string
>
> **Default** `/var/lib/neutron`

Where to store Neutron state files. This directory must be writable by the agent.

bind_host

> **Type** host address
>
> **Default** `0.0.0.0`

The host IP to bind to.

bind_port

> **Type** port number
>
> **Default** `9696`
>
> **Minimum Value** 0
>
> **Maximum Value** 65535

The port to bind to

api_extensions_path

> **Type** string
>
> **Default** `''`

The path for API extensions. Note that this can be a colon-separated list of paths. For example: api_extensions_path = extensions:/path/to/more/exts:/even/more/exts. The __path__ of neutron.extensions is appended to this, so if your extensions are in there you dont need to specify them here.

auth_strategy

Type string

Default `keystone`

The type of authentication to use

core_plugin

Type string

Default `<None>`

The core plugin Neutron will use

service_plugins

Type list

Default `[]`

The service plugins Neutron will use

base_mac

Type string

Default `fa:16:3e:00:00:00`

The base MAC address Neutron will use for VIFs. The first 3 octets will remain unchanged. If the 4th octet is not 00, it will also be used. The others will be randomly generated.

allow_bulk

Type boolean

Default `True`

Allow the usage of the bulk API

pagination_max_limit

Type string

Default `-1`

The maximum number of items returned in a single response, value was infinite or negative integer means no limit

default_availability_zones

Type list

Default `[]`

Default value of availability zone hints. The availability zone aware schedulers use this when the resources availability_zone_hints is empty. Multiple availability zones can be specified by a comma separated string. This value can be empty. In this case, even if availability_zone_hints for a resource is empty, availability zone is considered for high availability while scheduling the resource.

max_dns_nameservers

Type integer

Default 5

Maximum number of DNS nameservers per subnet

max_subnet_host_routes

Type integer

Default 20

Maximum number of host routes per subnet

ipv6_pd_enabled

Type boolean

Default False

Enables IPv6 Prefix Delegation for automatic subnet CIDR allocation. Set to True to enable IPv6 Prefix Delegation for subnet allocation in a PD-capable environment. Users making subnet creation requests for IPv6 subnets without providing a CIDR or subnetpool ID will be given a CIDR via the Prefix Delegation mechanism. Note that enabling PD will override the behavior of the default IPv6 subnetpool.

dhcp_lease_duration

Type integer

Default 86400

DHCP lease duration (in seconds). Use -1 to tell dnsmasq to use infinite lease times.

dns_domain

Type string

Default openstacklocal

Domain to use for building the hostnames

external_dns_driver

Type string

Default <None>

Driver for external DNS integration.

dhcp_agent_notification

Type boolean

Default True

Allow sending resource operation notification to DHCP agent

allow_overlapping_ips

Type boolean

Default `False`

Allow overlapping IP support in Neutron. Attention: the following parameter MUST be set to False if Neutron is being used in conjunction with Nova security groups.

host

Type host address

Default `example.domain`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Hostname to be used by the Neutron server, agents and services running on this machine. All the agents and services running on this machine must use the same host value.

network_link_prefix

Type string

Default `<None>`

This string is prepended to the normal URL that is returned in links to the OpenStack Network API. If it is empty (the default), the URLs are returned unchanged.

notify_nova_on_port_status_changes

Type boolean

Default `True`

Send notification to nova when port status changes

notify_nova_on_port_data_changes

Type boolean

Default `True`

Send notification to nova when port data (fixed_ips/floatingip) changes so nova can update its cache.

send_events_interval

Type integer

Default `2`

Number of seconds between sending events to nova if there are any events to send.

setproctitle

Type string

Default `on`

Set process name to match child worker role. Available options are: off - retains the previous behavior; on - renames processes to neutron-server: role (original string); brief - renames the same as on, but without the original string, such as neutron-server: role.

ipam_driver

Type string

Default `internal`

Neutron IPAM (IP address management) driver to use. By default, the reference implementation of the Neutron IPAM driver is used.

vlan_transparent

Type boolean

Default False

If True, then allow plugins that support it to create VLAN transparent networks.

filter_validation

Type boolean

Default True

If True, then allow plugins to decide whether to perform validations on filter parameters. Filter validation is enabled if this config is turned on and it is supported by all plugins

global_physnet_mtu

Type integer

Default 1500

MTU of the underlying physical network. Neutron uses this value to calculate MTU for all virtual network components. For flat and VLAN networks, neutron uses this value without modification. For overlay networks such as VXLAN, neutron automatically subtracts the overlay protocol overhead from this value. Defaults to 1500, the standard value for Ethernet.

Table 1: Deprecated Variations

Group	Name
ml2	segment_mtu

http_retries

Type integer

Default 3

Minimum Value 0

Number of times client connections (nova, ironic) should be retried on a failed HTTP call. 0 (zero) means connection is attempted only once (not retried). Setting to any positive integer means that on failure the connection is retried that many times. For example, setting to 3 means total attempts to connect will be 4.

backlog

Type integer

Default 4096

Number of backlog requests to configure the socket with

retry_until_window

Type integer

Default 30

Number of seconds to keep retrying to listen

use_ssl

Type boolean

Default False

Enable SSL on the API server

periodic_interval

Type integer

Default 40

Seconds between running periodic tasks.

api_workers

Type integer

Default <None>

Number of separate API worker processes for service. If not specified, the default is equal to the number of CPUs available for best performance, capped by potential RAM usage.

rpc_workers

Type integer

Default <None>

Number of RPC worker processes for service. If not specified, the default is equal to half the number of API workers.

rpc_state_report_workers

Type integer

Default 1

Number of RPC worker processes dedicated to state reports queue.

periodic_fuzzy_delay

Type integer

Default 5

Range of seconds to randomly delay when starting the periodic task scheduler to reduce stampeding. (Disable by setting to 0)

rpc_response_max_timeout

Type integer

Default 600

Maximum seconds to wait for a response from an RPC call.

interface_driver

Type string

Default <None>

The driver used to manage the virtual interface.

metadata_proxy_socket

Type string

Default `$state_path/metadata_proxy`

Location for Metadata Proxy UNIX domain socket.

metadata_proxy_user

Type string

Default `''`

User (uid or name) running metadata proxy after its initialization (if empty: agent effective user).

metadata_proxy_group

Type string

Default `''`

Group (gid or name) running metadata proxy after its initialization (if empty: agent effective group).

agent_down_time

Type integer

Default `75`

Seconds to regard the agent is down; should be at least twice report_interval, to be sure the agent is down for good.

dhcp_load_type

Type string

Default `networks`

Valid Values networks, subnets, ports

Representing the resource type whose load is being reported by the agent. This can be networks, subnets or ports. When specified (Default is networks), the server will extract particular load sent as part of its agent configuration object from the agent report state, which is the number of resources being consumed, at every report_interval.dhcp_load_type can be used in combination with network_scheduler_driver = neutron.scheduler.dhcp_agent_scheduler.WeightScheduler When the network_scheduler_driver is WeightScheduler, dhcp_load_type can be configured to represent the choice for the resource being balanced. Example: dhcp_load_type=networks

enable_new_agents

Type boolean

Default `True`

Agent starts with admin_state_up=False when enable_new_agents=False. In the case, users resources will not be scheduled automatically to the agent until admin changes admin_state_up to True.

max_routes

Type integer

Default `30`

Maximum number of routes per router

enable_snat_by_default

Type boolean

Default `True`

Define the default value of enable_snat if not provided in external_gateway_info.

network_scheduler_driver

Type string

Default `neutron.scheduler.dhcp_agent_scheduler.WeightScheduler`

Driver to use for scheduling network to DHCP agent

network_auto_schedule

Type boolean

Default `True`

Allow auto scheduling networks to DHCP agent.

allow_automatic_dhcp_failover

Type boolean

Default `True`

Automatically remove networks from offline DHCP agents.

dhcp_agents_per_network

Type integer

Default `1`

Minimum Value 1

Number of DHCP agents scheduled to host a tenant network. If this number is greater than 1, the scheduler automatically assigns multiple DHCP agents for a given tenant network, providing high availability for the DHCP service. However this does not provide high availability for the IPv6 metadata service in isolated networks.

enable_services_on_agents_with_admin_state_down

Type boolean

Default `False`

Enable services on an agent with admin_state_up False. If this option is False, when admin_state_up of an agent is turned False, services on it will be disabled. Agents with admin_state_up False are not selected for automatic scheduling regardless of this option. But manual scheduling to such agents is available if this option is True.

dvr_base_mac

Type string

Default `fa:16:3f:00:00:00`

The base mac address used for unique DVR instances by Neutron. The first 3 octets will remain unchanged. If the 4th octet is not 00, it will also be used. The others will be randomly generated. The dvr_base_mac *must* be different from base_mac to avoid mixing them up with MACs allocated for tenant ports. A 4 octet example would be dvr_base_mac = fa:16:3f:4f:00:00. The default is 3 octet

router_distributed

Type boolean

Default `False`

System-wide flag to determine the type of router that tenants can create. Only admin can override.

enable_dvr

Type boolean

Default `True`

Determine if setup is configured for DVR. If False, DVR API extension will be disabled.

host_dvr_for_dhcp

Type boolean

Default `True`

Flag to determine if hosting a DVR local router to the DHCP agent is desired. If False, any L3 function supported by the DHCP agent instance will not be possible, for instance: DNS.

router_scheduler_driver

Type string

Default `neutron.scheduler.l3_agent_scheduler.LeastRoutersScheduler`

Driver to use for scheduling router to a default L3 agent

router_auto_schedule

Type boolean

Default `True`

Allow auto scheduling of routers to L3 agent.

allow_automatic_l3agent_failover

Type boolean

Default `False`

Automatically reschedule routers from offline L3 agents to online L3 agents.

l3_ha

Type boolean

Default `False`

Enable HA mode for virtual routers.

max_l3_agents_per_router

Type integer

Default `3`

Maximum number of L3 agents which a HA router will be scheduled on. If it is set to 0 then the router will be scheduled on every agent.

l3_ha_net_cidr

Type string

Default `169.254.192.0/18`

Subnet used for the l3 HA admin network.

l3_ha_network_type

Type string

Default `''`

The network type to use when creating the HA network for an HA router. By default or if empty, the first tenant_network_types is used. This is helpful when the VRRP traffic should use a specific network which is not the default one.

l3_ha_network_physical_name

Type string

Default `''`

The physical network name with which the HA network can be created.

max_allowed_address_pair

Type integer

Default `10`

Maximum number of allowed address pairs

allowed_conntrack_helpers

Type list

Default `[{'tftp': 'udp'}, {'ftp': 'tcp'}, {'sip': 'tcp'}, {'sip': 'udp'}]`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Defines the allowed conntrack helpers, and conntack helper module protocol constraints.

rpc_conn_pool_size

Type integer

Default `30`

Minimum Value 1

Size of RPC connection pool.

Table 2: Deprecated Variations

Group	Name
DEFAULT	rpc_conn_pool_size

conn_pool_min_size

Type integer

Default 2

The pool size limit for connections expiration policy

conn_pool_ttl

Type integer

Default 1200

The time-to-live in sec of idle connections in the pool

executor_thread_pool_size

Type integer

Default 64

Size of executor thread pool when executor is threading or eventlet.

Table 3: Deprecated Variations

Group	Name
DEFAULT	rpc_thread_pool_size

rpc_response_timeout

Type integer

Default 60

Seconds to wait for a response from a call.

transport_url

Type string

Default rabbit://

The network address and optional user credentials for connecting to the messaging backend, in URL format. The expected format is:

driver://[user:pass@]host:port[,[userN:passN@]hostN:portN]/virtual_host?query

Example: rabbit://rabbitmq:password@127.0.0.1:5672//

For full details on the fields in the URL see the documentation of oslo_messaging.TransportURL at https://docs.openstack.org/oslo.messaging/latest/reference/transport.html

control_exchange

Type string

Default neutron

The default exchange under which topics are scoped. May be overridden by an exchange name specified in the transport_url option.

rpc_ping_enabled

Type boolean

Default `False`

Add an endpoint to answer to ping calls. Endpoint is named oslo_rpc_server_ping

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 4: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 5: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 6: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO',`

```
'oslo.messaging=INFO', 'oslo_messaging=INFO',
'iso8601=WARN', 'requests.packages.urllib3.
connectionpool=WARN', 'urllib3.connectionpool=WARN',
'websocket=WARN', 'requests.packages.
urllib3.util.retry=WARN', 'urllib3.util.
retry=WARN', 'keystonemiddleware=WARN', 'routes.
middleware=WARN', 'stevedore=WARN', 'taskflow=WARN',
'keystoneauth=WARN', 'oslo.cache=INFO',
'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

api_paste_config

Type string

Default `api-paste.ini`

File name for the paste.deploy config for api service

wsgi_log_format

Type string

Default `%(client_ip)s "%(request_line)s" status: %(status_code)s len: %(body_length)s time: %(wall_seconds).7f`

A python format string that is used as the template to generate log lines. The following values can beformatted into it: client_ip, date_time, request_line, status_code, body_length, wall_seconds.

tcp_keepidle

Type integer

Default `600`

Sets the value of TCP_KEEPIDLE in seconds for each server socket. Not supported on OS X.

wsgi_default_pool_size

Type integer

Default `100`

Size of the pool of greenthreads used by wsgi

max_header_line

Type integer

Default `16384`

Maximum line size of message headers to be accepted. max_header_line may need to be increased when using large tokens (typically those generated when keystone is configured to use PKI tokens with big service catalogs).

wsgi_keep_alive

Type boolean

Default `True`

If False, closes the client socket connection explicitly.

client_socket_timeout

Type integer

Default `900`

Timeout for client connections socket operations. If an incoming connection is idle for this number of seconds it will be closed. A value of 0 means wait forever.

wsgi_server_debug

Type boolean

Default `False`

True if the server should send exception tracebacks to the clients on 500 errors. If False, the server will respond with empty bodies.

agent

root_helper

Type string

Default `sudo`

Root helper application. Use sudo neutron-rootwrap /etc/neutron/rootwrap.conf to use the real root filter facility. Change to sudo to skip the filtering and just run the command directly.

use_helper_for_ns_read

Type boolean

Default `True`

Use the root helper when listing the namespaces on a system. This may not be required depending on the security configuration. If the root helper is not required, set this to False for a performance improvement.

root_helper_daemon

Type string

Default `<None>`

Root helper daemon application to use when possible.

Use sudo neutron-rootwrap-daemon /etc/neutron/rootwrap.conf to run rootwrap in daemon mode which has been reported to improve performance at scale. For more information on running rootwrap in daemon mode, see:

https://docs.openstack.org/oslo.rootwrap/latest/user/usage.html#daemon-mode

For the agent which needs to execute commands in Dom0 in the hypervisor of XenServer, this option should be set to xenapi_root_helper, so that it will keep a XenAPI session to pass commands to Dom0.

report_interval

Type floating point

Default `30`

Seconds between nodes reporting state to server; should be less than agent_down_time, best if it is half or less than agent_down_time.

log_agent_heartbeats

Type boolean

Default False

Log agent heartbeats

comment_iptables_rules

Type boolean

Default True

Add comments to iptables rules. Set to false to disallow the addition of comments to generated iptables rules that describe each rules purpose. System must support the iptables comments module for addition of comments.

debug_iptables_rules

Type boolean

Default False

Duplicate every iptables difference calculation to ensure the format being generated matches the format of iptables-save. This option should not be turned on for production systems because it imposes a performance penalty.

use_random_fully

Type boolean

Default True

Use random-fully in SNAT masquerade rules.

check_child_processes_action

Type string

Default respawn

Valid Values respawn, exit

Action to be executed when a child process dies

check_child_processes_interval

Type integer

Default 60

Interval between checks of child process liveness (seconds), use 0 to disable

kill_scripts_path

Type string

Default /etc/neutron/kill_scripts/

Location of scripts used to kill external processes. Names of scripts here must follow the pattern: <process-name>-kill where <process-name> is name of the process which should be killed using this script. For example, kill script for dnsmasq process should be named dnsmasq-kill. If path is set to None, then default kill command will be used to stop processes.

availability_zone

Type string

Default nova

Availability zone of this node

cors

allowed_origin

Type list

Default `<None>`

Indicate whether this resource may be shared with the domain received in the requests origin header. Format: <protocol>://<host>[:<port>], no trailing slash. Example: https://horizon.example.com

allow_credentials

Type boolean

Default `True`

Indicate that the actual request can include user credentials

expose_headers

Type list

Default `['X-Auth-Token', 'X-Subject-Token', 'X-Service-Token', 'X-OpenStack-Request-ID', 'OpenStack-Volume-microversion']`

Indicate which headers are safe to expose to the API. Defaults to HTTP Simple Headers.

max_age

Type integer

Default `3600`

Maximum cache age of CORS preflight requests.

allow_methods

Type list

Default `['GET', 'PUT', 'POST', 'DELETE', 'PATCH']`

Indicate which methods can be used during the actual request.

allow_headers

Type list

Default `['X-Auth-Token', 'X-Identity-Status', 'X-Roles', 'X-Service-Catalog', 'X-User-Id', 'X-Tenant-Id', 'X-OpenStack-Request-ID']`

Indicate which header field names may be used during the actual request.

database

engine

Type string

Default ''

Database engine for which script will be generated when using offline migration.

sqlite_synchronous

Type boolean

Default True

If True, SQLite uses synchronous mode.

Table 7: Deprecated Variations

Group	Name
DEFAULT	sqlite_synchronous

backend

Type string

Default sqlalchemy

The back end to use for the database.

Table 8: Deprecated Variations

Group	Name
DEFAULT	db_backend

connection

Type string

Default <None>

The SQLAlchemy connection string to use to connect to the database.

Table 9: Deprecated Variations

Group	Name
DEFAULT	sql_connection
DATABASE	sql_connection
sql	connection

slave_connection

Type string

Default <None>

The SQLAlchemy connection string to use to connect to the slave database.

mysql_sql_mode

Type string

Default `TRADITIONAL`

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

`mysql_enable_ndb`

Type boolean

Default `False`

If True, transparently enables support for handling MySQL Cluster (NDB).

`connection_recycle_time`

Type integer

Default `3600`

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

Table 10: Deprecated Variations

Group	Name
DATABASE	idle_timeout
database	idle_timeout
DEFAULT	sql_idle_timeout
DATABASE	sql_idle_timeout
sql	idle_timeout

`max_pool_size`

Type integer

Default `5`

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

Table 11: Deprecated Variations

Group	Name
DEFAULT	sql_max_pool_size
DATABASE	sql_max_pool_size

`max_retries`

Type integer

Default `10`

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

Table 12: Deprecated Variations

Group	Name
DEFAULT	sql_max_retries
DATABASE	sql_max_retries

retry_interval

Type integer

Default 10

Interval between retries of opening a SQL connection.

Table 13: Deprecated Variations

Group	Name
DEFAULT	sql_retry_interval
DATABASE	reconnect_interval

max_overflow

Type integer

Default 50

If set, use this value for max_overflow with SQLAlchemy.

Table 14: Deprecated Variations

Group	Name
DEFAULT	sql_max_overflow
DATABASE	sqlalchemy_max_overflow

connection_debug

Type integer

Default 0

Minimum Value 0

Maximum Value 100

Verbosity of SQL debugging information: 0=None, 100=Everything.

Table 15: Deprecated Variations

Group	Name
DEFAULT	sql_connection_debug

connection_trace

Type boolean

Default False

Add Python stack traces to SQL as comment strings.

Table 16: Deprecated Variations

Group	Name
DEFAULT	sql_connection_trace

pool_timeout

Type integer

Default `<None>`

If set, use this value for pool_timeout with SQLAlchemy.

Table 17: Deprecated Variations

Group	Name
DATABASE	sqlalchemy_pool_timeout

use_db_reconnect

Type boolean

Default `False`

Enable the experimental use of database reconnect on connection lost.

db_retry_interval

Type integer

Default `1`

Seconds between retries of a database transaction.

db_inc_retry_interval

Type boolean

Default `True`

If True, increases the interval between retries of a database operation up to db_max_retry_interval.

db_max_retry_interval

Type integer

Default `10`

If db_inc_retry_interval is set, the maximum seconds between retries of a database operation.

db_max_retries

Type integer

Default `20`

Maximum retries in case of connection error or deadlock error before error is raised. Set to -1 to specify an infinite retry count.

connection_parameters

Type string

Default ''

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&

ironic

auth_url

Type unknown type

Default <None>

Authentication URL

auth_type

Type unknown type

Default <None>

Authentication type to load

Table 18: Deprecated Variations

Group	Name
ironic	auth_plugin

cafile

Type string

Default <None>

PEM encoded Certificate Authority to use when verifying HTTPs connections.

certfile

Type string

Default <None>

PEM encoded client certificate cert file

collect_timing

Type boolean

Default False

Collect per-API call timing information.

default_domain_id

Type unknown type

Default <None>

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

default_domain_name

Type unknown type

Default `<None>`

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

domain_id

Type unknown type

Default `<None>`

Domain ID to scope to

domain_name

Type unknown type

Default `<None>`

Domain name to scope to

insecure

Type boolean

Default `False`

Verify HTTPS connections.

keyfile

Type string

Default `<None>`

PEM encoded client certificate key file

password

Type unknown type

Default `<None>`

Users password

project_domain_id

Type unknown type

Default `<None>`

Domain ID containing project

project_domain_name

Type unknown type

Default `<None>`

Domain name containing project

project_id

Type unknown type

Default `<None>`

Project ID to scope to

Table 19: Deprecated Variations

Group	Name
ironic	tenant-id
ironic	tenant_id

project_name

Type unknown type

Default `<None>`

Project name to scope to

Table 20: Deprecated Variations

Group	Name
ironic	tenant-name
ironic	tenant_name

split_loggers

Type boolean

Default `False`

Log requests to multiple loggers.

system_scope

Type unknown type

Default `<None>`

Scope for system operations

tenant_id

Type unknown type

Default `<None>`

Tenant ID

tenant_name

Type unknown type

Default `<None>`

Tenant Name

timeout

Type integer

Default `<None>`

Timeout value for http requests

trust_id

Type unknown type

Default `<None>`

Trust ID

user_domain_id

Type unknown type

Default `<None>`

Users domain id

user_domain_name

Type unknown type

Default `<None>`

Users domain name

user_id

Type unknown type

Default `<None>`

User id

username

Type unknown type

Default `<None>`

Username

Table 21: Deprecated Variations

Group	Name
ironic	user-name
ironic	user_name

enable_notifications

Type boolean

Default `False`

Send notification events to ironic. (For example on relevant port status changes.)

keystone_authtoken

www_authenticate_uri

Type string

Default `<None>`

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 22: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

auth_uri

Type string

Default `<None>`

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

> **Warning:** This option is deprecated for removal since Queens. Its value may be silently ignored in the future.
>
> **Reason** The auth_uri option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

auth_version

Type string

Default `<None>`

API version of the Identity API endpoint.

interface

Type string

Default `internal`

Interface to use for the Identity API endpoint. Valid values are public, internal (default) or admin.

delay_auth_decision

Type boolean

Default `False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

http_connect_timeout

Type integer

Default `<None>`

Request timeout value for communicating with Identity API server.

http_request_max_retries

Type integer

Default `3`

How many times are we trying to reconnect when communicating with Identity API Server.

cache

Type string

Default `<None>`

Request environment key where the Swift cache object is stored. When auth_token middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

certfile

Type string

Default `<None>`

Required if identity server requires client certificate

keyfile

Type string

Default `<None>`

Required if identity server requires client certificate

cafile

Type string

Default `<None>`

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

insecure

Type boolean

Default `False`

Verify HTTPS connections.

region_name

Type string

Default `<None>`

The region in which the identity server can be found.

memcached_servers

Type list

Default `<None>`

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 23: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

token_cache_time

Type integer

Default `300`

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

memcache_security_strategy

Type string

Default `None`

Valid Values None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, auth_token will raise an exception on initialization.

memcache_secret_key

Type string

Default `<None>`

(Optional, mandatory if memcache_security_strategy is defined) This string is used for key derivation.

memcache_pool_dead_retry

Type integer

Default `300`

(Optional) Number of seconds memcached server is considered dead before it is tried again.

memcache_pool_maxsize

Type integer

Default `10`

(Optional) Maximum total number of open connections to every memcached server.

memcache_pool_socket_timeout

Type integer

Default `3`

(Optional) Socket timeout in seconds for communicating with a memcached server.

memcache_pool_unused_timeout

Type integer

Default `60`

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

memcache_pool_conn_get_timeout

Type integer

Default `10`

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

memcache_use_advanced_pool

Type boolean

Default `False`

(Optional) Use the advanced (eventlet safe) memcached client pool. The advanced pool will only work under python 2.x.

include_service_catalog

Type boolean

Default `True`

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

enforce_token_bind

Type string

Default `permissive`

Used to control the use and type of token binding. Can be set to: disabled to not check token binding. permissive (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. strict like permissive but if the bind type is unknown the token will be rejected. required any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

service_token_roles

Type list

Default `['service']`

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

service_token_roles_required

Type boolean

Default `False`

For backwards compatibility reasons we must let valid service tokens pass that dont pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

service_type

Type string

Default `<None>`

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

auth_type

Type unknown type

Default `<None>`

Authentication type to load

Table 24: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

auth_section

Type unknown type

Default `<None>`

Config Section from which to load plugin specific options

nova

region_name

Type string

Default `<None>`

Name of nova region to use. Useful if keystone manages more than one region.

endpoint_type

Type string

Default `public`

Valid Values public, admin, internal

Type of the nova endpoint to use. This endpoint will be looked up in the keystone catalog and should be one of public, internal or admin.

live_migration_events

Type boolean

Default `False`

When this option is enabled, during the live migration, the OVS agent will only send the vif-plugged-event when the destination host interface is bound. This option also disables any other agent (like DHCP) to send to Nova this event when the port is provisioned.This option can be enabled if Nova patch https://review.opendev.org/c/openstack/nova/+/767368 is in place.This option is temporary and will be removed in Y and the behavior will be True.

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** In Y the Nova patch https://review.opendev.org/c/openstack/nova/+/767368 will be in the code even when running a Nova server in X.

auth_url

Type unknown type

Default `<None>`

Authentication URL

auth_type

Type unknown type

Default `<None>`

Authentication type to load

Table 25: Deprecated Variations

Group	Name
nova	auth_plugin

cafile

Type string

Default `<None>`

PEM encoded Certificate Authority to use when verifying HTTPs connections.

certfile

Type string

Default `<None>`

PEM encoded client certificate cert file

collect_timing

Type boolean

Default `False`

Collect per-API call timing information.

default_domain_id

Type unknown type

Default <None>

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

default_domain_name

Type unknown type

Default <None>

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

domain_id

Type unknown type

Default <None>

Domain ID to scope to

domain_name

Type unknown type

Default <None>

Domain name to scope to

insecure

Type boolean

Default False

Verify HTTPS connections.

keyfile

Type string

Default <None>

PEM encoded client certificate key file

password

Type unknown type

Default <None>

Users password

project_domain_id

Type unknown type

Default <None>

Domain ID containing project

project_domain_name

Type unknown type

Default <None>

Domain name containing project

project_id

Type unknown type

Default <None>

Project ID to scope to

Table 26: Deprecated Variations

Group	Name
nova	tenant-id
nova	tenant_id

project_name

Type unknown type

Default <None>

Project name to scope to

Table 27: Deprecated Variations

Group	Name
nova	tenant-name
nova	tenant_name

split_loggers

Type boolean

Default False

Log requests to multiple loggers.

system_scope

Type unknown type

Default <None>

Scope for system operations

tenant_id

Type unknown type

Default <None>

Tenant ID

tenant_name

Type unknown type

Default <None>

Tenant Name

timeout

Type integer

Default `<None>`

Timeout value for http requests

trust_id

Type unknown type

Default `<None>`

Trust ID

user_domain_id

Type unknown type

Default `<None>`

Users domain id

user_domain_name

Type unknown type

Default `<None>`

Users domain name

user_id

Type unknown type

Default `<None>`

User id

username

Type unknown type

Default `<None>`

Username

Table 28: Deprecated Variations

Group	Name
nova	user-name
nova	user_name

oslo_concurrency

disable_process_locking

Type boolean

Default `False`

Enables or disables inter-process locks.

Table 29: Deprecated Variations

Group	Name
DEFAULT	disable_process_locking

lock_path

Type string

Default `<None>`

Directory to use for lock files. For security, the specified directory should only be writable by the user running the processes that need locking. Defaults to environment variable OSLO_LOCK_PATH. If external locks are used, a lock path must be set.

Table 30: Deprecated Variations

Group	Name
DEFAULT	lock_path

oslo_messaging_amqp

container_name

Type string

Default `<None>`

Name for the AMQP container. must be globally unique. Defaults to a generated UUID

Table 31: Deprecated Variations

Group	Name
amqp1	container_name

idle_timeout

Type integer

Default `0`

Timeout for inactive connections (in seconds)

Table 32: Deprecated Variations

Group	Name
amqp1	idle_timeout

trace

Type boolean

Default `False`

Debug: dump AMQP frames to stdout

Table 33: Deprecated Variations

Group	Name
amqp1	trace

ssl

Type boolean

Default `False`

Attempt to connect via SSL. If no other ssl-related parameters are given, it will use the systems CA-bundle to verify the servers certificate.

ssl_ca_file

Type string

Default `''`

CA certificate PEM file used to verify the servers certificate

Table 34: Deprecated Variations

Group	Name
amqp1	ssl_ca_file

ssl_cert_file

Type string

Default `''`

Self-identifying certificate PEM file for client authentication

Table 35: Deprecated Variations

Group	Name
amqp1	ssl_cert_file

ssl_key_file

Type string

Default `''`

Private key PEM file used to sign ssl_cert_file certificate (optional)

Table 36: Deprecated Variations

Group	Name
amqp1	ssl_key_file

ssl_key_password

Type string

Default `<None>`

Password for decrypting ssl_key_file (if encrypted)

Table 37: Deprecated Variations

Group	Name
amqp1	ssl_key_password

ssl_verify_vhost

Type boolean

Default `False`

By default SSL checks that the name in the servers certificate matches the hostname in the transport_url. In some configurations it may be preferable to use the virtual hostname instead, for example if the server uses the Server Name Indication TLS extension (rfc6066) to provide a certificate per virtual host. Set ssl_verify_vhost to True if the servers SSL certificate uses the virtual host name instead of the DNS name.

sasl_mechanisms

Type string

Default `''`

Space separated list of acceptable SASL mechanisms

Table 38: Deprecated Variations

Group	Name
amqp1	sasl_mechanisms

sasl_config_dir

Type string

Default `''`

Path to directory that contains the SASL configuration

Table 39: Deprecated Variations

Group	Name
amqp1	sasl_config_dir

sasl_config_name

Type string

Default `''`

Name of configuration file (without .conf suffix)

Table 40: Deprecated Variations

Group	Name
amqp1	sasl_config_name

sasl_default_realm

Type string

Default ''

SASL realm to use if no realm present in username

connection_retry_interval

Type integer

Default 1

Minimum Value 1

Seconds to pause before attempting to re-connect.

connection_retry_backoff

Type integer

Default 2

Minimum Value 0

Increase the connection_retry_interval by this many seconds after each unsuccessful failover attempt.

connection_retry_interval_max

Type integer

Default 30

Minimum Value 1

Maximum limit for connection_retry_interval + connection_retry_backoff

link_retry_delay

Type integer

Default 10

Minimum Value 1

Time to pause between re-connecting an AMQP 1.0 link that failed due to a recoverable error.

default_reply_retry

Type integer

Default 0

Minimum Value -1

The maximum number of attempts to re-send a reply message which failed due to a recoverable error.

default_reply_timeout

Type integer

Default 30

Minimum Value 5

The deadline for an rpc reply message delivery.

default_send_timeout

Type integer

Default `30`

Minimum Value 5

The deadline for an rpc cast or call message delivery. Only used when caller does not provide a timeout expiry.

default_notify_timeout

Type integer

Default `30`

Minimum Value 5

The deadline for a sent notification message delivery. Only used when caller does not provide a timeout expiry.

default_sender_link_timeout

Type integer

Default `600`

Minimum Value 1

The duration to schedule a purge of idle sender links. Detach link after expiry.

addressing_mode

Type string

Default `dynamic`

Indicates the addressing mode used by the driver. Permitted values: legacy - use legacy non-routable addressing routable - use routable addresses dynamic - use legacy addresses if the message bus does not support routing otherwise use routable addressing

pseudo_vhost

Type boolean

Default `True`

Enable virtual host support for those message buses that do not natively support virtual hosting (such as qpidd). When set to true the virtual host name will be added to all message bus addresses, effectively creating a private subnet per virtual host. Set to False if the message bus supports virtual hosting using the hostname field in the AMQP 1.0 Open performative as the name of the virtual host.

server_request_prefix

Type string

Default `exclusive`

address prefix used when sending to a specific server

Table 41: Deprecated Variations

Group	Name
amqp1	server_request_prefix

broadcast_prefix

Type string

Default broadcast

address prefix used when broadcasting to all servers

Table 42: Deprecated Variations

Group	Name
amqp1	broadcast_prefix

group_request_prefix

Type string

Default unicast

address prefix when sending to any server in group

Table 43: Deprecated Variations

Group	Name
amqp1	group_request_prefix

rpc_address_prefix

Type string

Default openstack.org/om/rpc

Address prefix for all generated RPC addresses

notify_address_prefix

Type string

Default openstack.org/om/notify

Address prefix for all generated Notification addresses

multicast_address

Type string

Default multicast

Appended to the address prefix when sending a fanout message. Used by the message bus to identify fanout messages.

unicast_address

Type string

Default unicast

Appended to the address prefix when sending to a particular RPC/Notification server. Used by the message bus to identify messages sent to a single destination.

anycast_address

Type string

Default `anycast`

Appended to the address prefix when sending to a group of consumers. Used by the message bus to identify messages that should be delivered in a round-robin fashion across consumers.

default_notification_exchange

Type string

Default `<None>`

Exchange name used in notification addresses. Exchange name resolution precedence: Target.exchange if set else default_notification_exchange if set else control_exchange if set else notify

default_rpc_exchange

Type string

Default `<None>`

Exchange name used in RPC addresses. Exchange name resolution precedence: Target.exchange if set else default_rpc_exchange if set else control_exchange if set else rpc

reply_link_credit

Type integer

Default `200`

Minimum Value 1

Window size for incoming RPC Reply messages.

rpc_server_credit

Type integer

Default `100`

Minimum Value 1

Window size for incoming RPC Request messages

notify_server_credit

Type integer

Default `100`

Minimum Value 1

Window size for incoming Notification messages

pre_settled

Type multi-valued

Default `rpc-cast`

Default `rpc-reply`

Send messages of this type pre-settled. Pre-settled messages will not receive acknowledgement from the peer. Note well: pre-settled messages may be silently discarded if the delivery fails. Permitted values: rpc-call - send RPC Calls pre-settled rpc-reply- send RPC Replies pre-settled rpc-cast - Send RPC Casts pre-settled notify - Send Notifications pre-settled

oslo_messaging_kafka

kafka_max_fetch_bytes

Type integer

Default `1048576`

Max fetch bytes of Kafka consumer

kafka_consumer_timeout

Type floating point

Default `1.0`

Default timeout(s) for Kafka consumers

pool_size

Type integer

Default `10`

Pool Size for Kafka Consumers

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

conn_pool_min_size

Type integer

Default `2`

The pool size limit for connections expiration policy

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

conn_pool_ttl

Type integer

Default `1200`

The time-to-live in sec of idle connections in the pool

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

consumer_group

Type string

Default `oslo_messaging_consumer`

Group id for Kafka consumer. Consumers in one group will coordinate message consumption

producer_batch_timeout

Type floating point

Default `0.0`

Upper bound on the delay for KafkaProducer batching in seconds

producer_batch_size

Type integer

Default `16384`

Size of batch for the producer async send

compression_codec

Type string

Default `none`

Valid Values none, gzip, snappy, lz4, zstd

The compression codec for all data generated by the producer. If not set, compression will not be used. Note that the allowed values of this depend on the kafka version

enable_auto_commit

Type boolean

Default `False`

Enable asynchronous consumer commits

max_poll_records

Type integer

Default `500`

The maximum number of records returned in a poll call

security_protocol

Type string

Default `PLAINTEXT`

Valid Values PLAINTEXT, SASL_PLAINTEXT, SSL, SASL_SSL

Protocol used to communicate with brokers

sasl_mechanism

Type string

Default PLAIN

Mechanism when security protocol is SASL

ssl_cafile

Type string

Default ''

CA certificate PEM file used to verify the server certificate

ssl_client_cert_file

Type string

Default ''

Client certificate PEM file used for authentication.

ssl_client_key_file

Type string

Default ''

Client key PEM file used for authentication.

ssl_client_key_password

Type string

Default ''

Client key password file used for authentication.

oslo_messaging_notifications

driver

Type multi-valued

Default ''

The Drivers(s) to handle sending notifications. Possible values are messaging, messagingv2, routing, log, test, noop

Table 44: Deprecated Variations

Group	Name
DEFAULT	notification_driver

transport_url

Type string

Default `<None>`

A URL representing the messaging driver to use for notifications. If not set, we fall back to the same configuration used for RPC.

Table 45: Deprecated Variations

Group	Name
DEFAULT	notification_transport_url

topics

Type list

Default `['notifications']`

AMQP topic used for OpenStack notifications.

Table 46: Deprecated Variations

Group	Name
rpc_notifier2	topics
DEFAULT	notification_topics

retry

Type integer

Default `-1`

The maximum number of attempts to re-send a notification message which failed to be delivered due to a recoverable error. 0 - No retry, -1 - indefinite

oslo_messaging_rabbit

amqp_durable_queues

Type boolean

Default `False`

Use durable queues in AMQP.

amqp_auto_delete

Type boolean

Default `False`

Auto-delete queues in AMQP.

Table 47: Deprecated Variations

Group	Name
DEFAULT	amqp_auto_delete

ssl

Type boolean

Default `False`

Connect over SSL.

Table 48: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_use_ssl

ssl_version

Type string

Default `''`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

Table 49: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_version

ssl_key_file

Type string

Default `''`

SSL key file (valid only if SSL enabled).

Table 50: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_keyfile

ssl_cert_file

Type string

Default `''`

SSL cert file (valid only if SSL enabled).

Table 51: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_certfile

ssl_ca_file

Type string

Default `''`

SSL certification authority file (valid only if SSL enabled).

Table 52: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_ca_certs

heartbeat_in_pthread

Type boolean

Default False

EXPERIMENTAL: Run the health check heartbeat thread through a native python thread. By default if this option isnt provided the health check heartbeat will inherit the execution model from the parent process. By example if the parent process have monkey patched the stdlib by using eventlet/greenlet then the heartbeat will be run through a green thread.

kombu_reconnect_delay

Type floating point

Default 1.0

How long to wait before reconnecting in response to an AMQP consumer cancel notification.

Table 53: Deprecated Variations

Group	Name
DEFAULT	kombu_reconnect_delay

kombu_compression

Type string

Default <None>

EXPERIMENTAL: Possible values are: gzip, bz2. If not set compression will not be used. This option may not be available in future versions.

kombu_missing_consumer_retry_timeout

Type integer

Default 60

How long to wait a missing client before abandoning to send it its replies. This value should not be longer than rpc_response_timeout.

Table 54: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_reconnect_timeout

kombu_failover_strategy

Type string

Default round-robin

Valid Values round-robin, shuffle

Determines how the next RabbitMQ node is chosen in case the one we are currently connected to becomes unavailable. Takes effect only if more than one RabbitMQ node is provided in config.

rabbit_login_method

Type string

Default `AMQPLAIN`

Valid Values PLAIN, AMQPLAIN, RABBIT-CR-DEMO

The RabbitMQ login method.

Table 55: Deprecated Variations

Group	Name
DEFAULT	rabbit_login_method

rabbit_retry_interval

Type integer

Default `1`

How frequently to retry connecting with RabbitMQ.

rabbit_retry_backoff

Type integer

Default `2`

How long to backoff for between retries when connecting to RabbitMQ.

Table 56: Deprecated Variations

Group	Name
DEFAULT	rabbit_retry_backoff

rabbit_interval_max

Type integer

Default `30`

Maximum interval of RabbitMQ connection retries. Default is 30 seconds.

rabbit_ha_queues

Type boolean

Default `False`

Try to use HA queues in RabbitMQ (x-ha-policy: all). If you change this option, you must wipe the RabbitMQ database. In RabbitMQ 3.0, queue mirroring is no longer controlled by the x-ha-policy argument when declaring a queue. If you just want to make sure that all queues (except those with auto-generated names) are mirrored across all nodes, run: rabbitmqctl set_policy HA ^(?!amq.).* {ha-mode: all}

Table 57: Deprecated Variations

Group	Name
DEFAULT	rabbit_ha_queues

rabbit_transient_queues_ttl

Type integer

Default 1800

Minimum Value 1

Positive integer representing duration in seconds for queue TTL (x-expires). Queues which are unused for the duration of the TTL are automatically deleted. The parameter affects only reply and fanout queues.

rabbit_qos_prefetch_count

Type integer

Default 0

Specifies the number of messages to prefetch. Setting to zero allows unlimited messages.

heartbeat_timeout_threshold

Type integer

Default 60

Number of seconds after which the Rabbit broker is considered down if heartbeats keep-alive fails (0 disables heartbeat).

heartbeat_rate

Type integer

Default 2

How often times during the heartbeat_timeout_threshold we check the heartbeat.

direct_mandatory_flag

Type boolean

Default True

(DEPRECATED) Enable/Disable the RabbitMQ mandatory flag for direct send. The direct send is used as reply, so the MessageUndeliverable exception is raised in case the client queue does not exist.MessageUndeliverable exception will be used to loop for a timeout to lets a chance to sender to recover.This flag is deprecated and it will not be possible to deactivate this functionality anymore

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

Reason Mandatory flag no longer deactivable.

enable_cancel_on_failover

Type boolean

Default `False`

Enable x-cancel-on-ha-failover flag so that rabbitmq server will cancel and notify consumerswhen queue is down

oslo_middleware

enable_proxy_headers_parsing

Type boolean

Default `False`

Whether the application is behind a proxy or not. This determines if the middleware should parse the headers or not.

oslo_policy

enforce_scope

Type boolean

Default `False`

This option controls whether or not to enforce scope when evaluating policies. If `True`, the scope of the token used in the request is compared to the `scope_types` of the policy being enforced. If the scopes do not match, an `InvalidScope` exception will be raised. If `False`, a message will be logged informing operators that policies are being invoked with mismatching scope.

enforce_new_defaults

Type boolean

Default `False`

This option controls whether or not to use old deprecated defaults when evaluating policies. If `True`, the old deprecated defaults are not going to be evaluated. This means if any existing token is allowed for old defaults but is disallowed for new defaults, it will be disallowed. It is encouraged to enable this flag along with the `enforce_scope` flag so that you can get the benefits of new defaults and `scope_type` together

policy_file

Type string

Default `policy.json`

The relative or absolute path of a file that maps roles to permissions for a given service. Relative paths must be specified in relation to the configuration file setting this option.

Table 58: Deprecated Variations

Group	Name
DEFAULT	policy_file

policy_default_rule

Type string

Default `default`

Default rule. Enforced when a requested rule is not found.

Table 59: Deprecated Variations

Group	Name
DEFAULT	policy_default_rule

policy_dirs

Type multi-valued

Default `policy.d`

Directories where policy configuration files are stored. They can be relative to any directory in the search path defined by the config_dir option, or absolute paths. The file defined by policy_file must exist for these directories to be searched. Missing or empty directories are ignored.

Table 60: Deprecated Variations

Group	Name
DEFAULT	policy_dirs

remote_content_type

Type string

Default `application/x-www-form-urlencoded`

Valid Values application/x-www-form-urlencoded, application/json

Content Type to send and receive data for REST based policy check

remote_ssl_verify_server_crt

Type boolean

Default `False`

server identity verification for REST based policy check

remote_ssl_ca_crt_file

Type string

Default `<None>`

Absolute path to ca cert file for REST based policy check

remote_ssl_client_crt_file

Type string

Default `<None>`

Absolute path to client cert for REST based policy check

remote_ssl_client_key_file

Type string

Default `<None>`

Absolute path client key file REST based policy check

privsep

Configuration options for the oslo.privsep daemon. Note that this group name can be changed by the consuming service. Check the services docs to see if this is the case.

user

Type string

Default `<None>`

User that the privsep daemon should run as.

group

Type string

Default `<None>`

Group that the privsep daemon should run as.

capabilities

Type unknown type

Default `[]`

List of Linux capabilities retained by the privsep daemon.

thread_pool_size

Type integer

Default `multiprocessing.cpu_count()`

Minimum Value 1

This option has a sample default set, which means that its actual default value may vary from the one documented above.

The number of threads available for privsep to concurrently run processes. Defaults to the number of CPU cores in the system.

helper_command

Type string

Default `<None>`

Command to invoke to start the privsep daemon if not using the fork method. If not specified, a default is generated using sudo privsep-helper and arguments designed to recreate the current configuration. This command must accept suitable privsep_context and privsep_sock_path arguments.

quotas

default_quota

Type integer

Default `-1`

Default number of resource allowed per tenant. A negative value means unlimited.

quota_network

Type integer

Default `100`

Number of networks allowed per tenant. A negative value means unlimited.

quota_subnet

Type integer

Default `100`

Number of subnets allowed per tenant, A negative value means unlimited.

quota_port

Type integer

Default `500`

Number of ports allowed per tenant. A negative value means unlimited.

quota_driver

Type string

Default `neutron.db.quota.driver.DbQuotaDriver`

Default driver to use for quota checks.

track_quota_usage

Type boolean

Default `True`

Keep in track in the database of current resource quota usage. Plugins which do not leverage the neutron database should set this flag to False.

quota_router

Type integer

Default `10`

Number of routers allowed per tenant. A negative value means unlimited.

quota_floatingip

Type integer

Default `50`

Number of floating IPs allowed per tenant. A negative value means unlimited.

quota_security_group

Type integer

Default 10

Number of security groups allowed per tenant. A negative value means unlimited.

quota_security_group_rule

Type integer

Default 100

Number of security rules allowed per tenant. A negative value means unlimited.

ssl

ca_file

Type string

Default <None>

CA certificate file to use to verify connecting clients.

Table 61: Deprecated Variations

Group	Name
DEFAULT	ssl_ca_file

cert_file

Type string

Default <None>

Certificate file to use when starting the server securely.

Table 62: Deprecated Variations

Group	Name
DEFAULT	ssl_cert_file

key_file

Type string

Default <None>

Private key file to use when starting the server securely.

Table 63: Deprecated Variations

Group	Name
DEFAULT	ssl_key_file

version

Type string

Default `<None>`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

ciphers

Type string

Default `<None>`

Sets the list of available ciphers. value should be a string in the OpenSSL cipher list format.

9.1.2 ml2_conf.ini

DEFAULT

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 64: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 65: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 66: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO',`

```
'oslo.messaging=INFO', 'oslo_messaging=INFO',
'iso8601=WARN', 'requests.packages.urllib3.
connectionpool=WARN', 'urllib3.connectionpool=WARN',
'websocket=WARN', 'requests.packages.
urllib3.util.retry=WARN', 'urllib3.util.
retry=WARN', 'keystonemiddleware=WARN', 'routes.
middleware=WARN', 'stevedore=WARN', 'taskflow=WARN',
'keystoneauth=WARN', 'oslo.cache=INFO',
'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

ml2

type_drivers

Type list

Default `['local', 'flat', 'vlan', 'gre', 'vxlan', 'geneve']`

List of network type driver entrypoints to be loaded from the neutron.ml2.type_drivers namespace.

tenant_network_types

Type list

Default `['local']`

Ordered list of network_types to allocate as tenant networks. The default value local is useful for single-box testing but provides no connectivity between hosts.

mechanism_drivers

Type list

Default `[]`

An ordered list of networking mechanism driver entrypoints to be loaded from the neutron.ml2.mechanism_drivers namespace.

extension_drivers

Type list

Default `[]`

An ordered list of extension driver entrypoints to be loaded from the neutron.ml2.extension_drivers namespace. For example: extension_drivers = port_security,qos

path_mtu

Type integer

Default `0`

Maximum size of an IP packet (MTU) that can traverse the underlying physical network infrastructure without fragmentation when using an overlay/tunnel protocol. This option allows specifying a physical network MTU value that differs from the default global_physnet_mtu value.

physical_network_mtus

Type list

Default `[]`

A list of mappings of physical networks to MTU values. The format of the mapping is <physnet>:<mtu val>. This mapping allows specifying a physical network MTU value that differs from the default global_physnet_mtu value.

external_network_type

Type string

Default `<None>`

Default network type for external networks when no provider attributes are specified. By default it is None, which means that if provider attributes are not specified while creating external networks then they will have the same type as tenant networks. Allowed values for external_network_type config option depend on the network type values configured in type_drivers config option.

overlay_ip_version

Type integer

Default `4`

IP version of all overlay (tunnel) network endpoints. Use a value of 4 for IPv4 or 6 for IPv6.

ml2_type_flat

flat_networks

Type list

Default `*`

List of physical_network names with which flat networks can be created. Use default * to allow flat networks with arbitrary physical_network names. Use an empty list to disable flat networks.

ml2_type_geneve

vni_ranges

Type list

Default `[]`

Comma-separated list of <vni_min>:<vni_max> tuples enumerating ranges of Geneve VNI IDs that are available for tenant network allocation

max_header_size

Type integer

Default `30`

Geneve encapsulation header size is dynamic, this value is used to calculate the maximum MTU for the driver. The default size for this field is 30, which is the size of the Geneve header without any additional option headers.

ml2_type_gre

tunnel_id_ranges

Type list

Default `[]`

Comma-separated list of <tun_min>:<tun_max> tuples enumerating ranges of GRE tunnel IDs that are available for tenant network allocation

ml2_type_vlan

network_vlan_ranges

Type list

Default `[]`

List of <physical_network>:<vlan_min>:<vlan_max> or <physical_network> specifying physical_network names usable for VLAN provider and tenant networks, as well as ranges of VLAN tags on each available for allocation to tenant networks.

ml2_type_vxlan

vni_ranges

Type list

Default `[]`

Comma-separated list of <vni_min>:<vni_max> tuples enumerating ranges of VXLAN VNI IDs that are available for tenant network allocation

vxlan_group

Type string

Default `<None>`

Multicast group for VXLAN. When configured, will enable sending all broadcast traffic to this multicast group. When left unconfigured, will disable multicast VXLAN mode.

ovs_driver

vnic_type_prohibit_list

Type list

Default `[]`

Comma-separated list of VNIC types for which support is administratively prohibited by the mechanism driver. Please note that the supported vnic_types depend on your network interface card, on the kernel version of your operating system, and on other factors, like OVS version. In case of ovs mechanism driver the valid vnic types are normal and direct. Note that direct is supported only from kernel 4.8, and from ovs 2.8.0. Bind DIRECT (SR-IOV) port allows to offload the OVS flows using tc to the SR-IOV NIC. This allows to support hardware offload via tc and that allows us to manage the VF by OpenFlow control plane using representor net-device.

Table 67: Deprecated Variations

Group	Name
ovs_driver	vnic_type_blacklist

securitygroup

firewall_driver

Type string

Default `<None>`

Driver for security groups firewall in the L2 agent

enable_security_group

Type boolean

Default `True`

Controls whether the neutron security group API is enabled in the server. It should be false when using no security groups or using the nova security group API.

enable_ipset

Type boolean

Default `True`

Use ipset to speed-up the iptables based security groups. Enabling ipset support requires that ipset is installed on L2 agent node.

permitted_ethertypes

Type list

Default `[]`

Comma-separated list of ethertypes to be permitted, in hexadecimal (starting with 0x). For example, 0x4008 to permit InfiniBand.

sriov_driver

vnic_type_prohibit_list

Type list

Default `[]`

Comma-separated list of VNIC types for which support is administratively prohibited by the mechanism driver. Please note that the supported vnic_types depend on your network interface card, on the kernel version of your operating system, and on other factors. In case of sriov mechanism driver the valid VNIC types are direct, macvtap and direct-physical.

Table 68: Deprecated Variations

Group	Name
sriov_driver	vnic_type_blacklist

9.1.3 linuxbridge_agent.ini

DEFAULT

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 69: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 70: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 71: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO',`

```
'oslo.messaging=INFO', 'oslo_messaging=INFO',
'iso8601=WARN', 'requests.packages.urllib3.
connectionpool=WARN', 'urllib3.connectionpool=WARN',
'websocket=WARN', 'requests.packages.
urllib3.util.retry=WARN', 'urllib3.util.
retry=WARN', 'keystonemiddleware=WARN', 'routes.
middleware=WARN', 'stevedore=WARN', 'taskflow=WARN',
'keystoneauth=WARN', 'oslo.cache=INFO',
'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance:  %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance:  %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

`polling_interval`

Type integer

Default `2`

The number of seconds the agent will wait between polling for local device changes.

`quitting_rpc_timeout`

Type integer

Default `10`

Set new timeout in seconds for new rpc calls after agent receives SIGTERM. If value is set to 0, rpc timeout wont be changed

`dscp`

Type integer

Default `<None>`

Minimum Value 0

Maximum Value 63

The DSCP value to use for outer headers during tunnel encapsulation.

`dscp_inherit`

Type boolean

Default `False`

If set to True, the DSCP value of tunnel interfaces is overwritten and set to inherit. The DSCP value of the inner header is then copied to the outer header.

`extensions`

Type list

Default `[]`

Extensions list to use

linux_bridge

physical_interface_mappings

Type list

Default `[]`

Comma-separated list of <physical_network>:<physical_interface> tuples mapping physical network names to the agents node-specific physical network interfaces to be used for flat and VLAN networks. All physical networks listed in network_vlan_ranges on the server should have mappings to appropriate interfaces on each agent.

bridge_mappings

Type list

Default `[]`

List of <physical_network>:<physical_bridge>

network_log

rate_limit

Type integer

Default `100`

Minimum Value 100

Maximum packets logging per second.

burst_limit

Type integer

Default `25`

Minimum Value 25

Maximum number of packets per rate_limit.

local_output_log_base

Type string

Default `<None>`

Output logfile path on agent side, default syslog file.

securitygroup

firewall_driver

Type string

Default <None>

Driver for security groups firewall in the L2 agent

enable_security_group

Type boolean

Default True

Controls whether the neutron security group API is enabled in the server. It should be false when using no security groups or using the nova security group API.

enable_ipset

Type boolean

Default True

Use ipset to speed-up the iptables based security groups. Enabling ipset support requires that ipset is installed on L2 agent node.

permitted_ethertypes

Type list

Default []

Comma-separated list of ethertypes to be permitted, in hexadecimal (starting with 0x). For example, 0x4008 to permit InfiniBand.

vxlan

enable_vxlan

Type boolean

Default True

Enable VXLAN on the agent. Can be enabled when agent is managed by ml2 plugin using linuxbridge mechanism driver

ttl

Type integer

Default <None>

TTL for vxlan interface protocol packets.

tos

Type integer

Default <None>

TOS for vxlan interface protocol packets. This option is deprecated in favor of the dscp option in the AGENT section and will be removed in a future release. To convert the TOS value to DSCP, divide by 4.

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.

vxlan_group

Type string

Default `224.0.0.1`

Multicast group(s) for vxlan interface. A range of group addresses may be specified by using CIDR notation. Specifying a range allows different VNIs to use different group addresses, reducing or eliminating spurious broadcast traffic to the tunnel endpoints. To reserve a unique group for each possible (24-bit) VNI, use a /8 such as 239.0.0.0/8. This setting must be the same on all the agents.

local_ip

Type ip address

Default `<None>`

IP address of local overlay (tunnel) network endpoint. Use either an IPv4 or IPv6 address that resides on one of the host network interfaces. The IP version of this value must match the value of the overlay_ip_version option in the ML2 plug-in configuration file on the neutron server node(s).

udp_srcport_min

Type port number

Default `0`

Minimum Value 0

Maximum Value 65535

The minimum of the UDP source port range used for VXLAN communication.

udp_srcport_max

Type port number

Default `0`

Minimum Value 0

Maximum Value 65535

The maximum of the UDP source port range used for VXLAN communication.

udp_dstport

Type port number

Default `<None>`

Minimum Value 0

Maximum Value 65535

The UDP port used for VXLAN communication. By default, the Linux kernel doesnt use the IANA assigned standard value, so if you want to use it, this option must be set to 4789. It is not set by default because of backward compatibiltiy.

l2_population

Type boolean

Default `False`

Extension to use alongside ml2 plugins l2population mechanism driver. It enables the plugin to populate VXLAN forwarding table.

arp_responder

Type boolean

Default `False`

Enable local ARP responder which provides local responses instead of performing ARP broadcast into the overlay. Enabling local ARP responder is not fully compatible with the allowed-address-pairs extension.

multicast_ranges

Type list

Default `[]`

Optional comma-separated list of <multicast address>:<vni_min>:<vni_max> triples describing how to assign a multicast address to VXLAN according to its VNI ID.

9.1.4 macvtap_agent.ini

DEFAULT

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 72: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 73: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 74: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default 0

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default 0

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default CRITICAL

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default False

Enables or disables fatal status of deprecations.

agent

polling_interval

Type integer

Default 2

The number of seconds the agent will wait between polling for local device changes.

quitting_rpc_timeout

Type integer

Default 10

Set new timeout in seconds for new rpc calls after agent receives SIGTERM. If value is set to 0, rpc timeout wont be changed

dscp

Type integer

Default <None>

Minimum Value 0

Maximum Value 63

The DSCP value to use for outer headers during tunnel encapsulation.

dscp_inherit

Type boolean

Default `False`

If set to True, the DSCP value of tunnel interfaces is overwritten and set to inherit. The DSCP value of the inner header is then copied to the outer header.

macvtap

physical_interface_mappings

Type list

Default `[]`

Comma-separated list of <physical_network>:<physical_interface> tuples mapping physical network names to the agents node-specific physical network interfaces to be used for flat and VLAN networks. All physical networks listed in network_vlan_ranges on the server should have mappings to appropriate interfaces on each agent.

securitygroup

firewall_driver

Type string

Default `<None>`

Driver for security groups firewall in the L2 agent

enable_security_group

Type boolean

Default `True`

Controls whether the neutron security group API is enabled in the server. It should be false when using no security groups or using the nova security group API.

enable_ipset

Type boolean

Default `True`

Use ipset to speed-up the iptables based security groups. Enabling ipset support requires that ipset is installed on L2 agent node.

permitted_ethertypes

Type list

Default `[]`

Comma-separated list of ethertypes to be permitted, in hexadecimal (starting with 0x). For example, 0x4008 to permit InfiniBand.

9.1.5 openvswitch_agent.ini

DEFAULT

rpc_response_max_timeout

Type integer

Default 600

Maximum seconds to wait for a response from an RPC call.

debug

Type boolean

Default False

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default <None>

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 75: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default %Y-%m-%d %H:%M:%S

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default <None>

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 76: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 77: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default False

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default False

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default 1

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default days

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default 30

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default 200

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default none

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO',`

'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default False

Enables or disables publication of error events.

instance_format

Type string

Default "[instance: %(uuid)s] "

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default "[instance: %(uuid)s] "

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default 0

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default 0

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default CRITICAL

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

minimize_polling

Type boolean

Default `True`

Minimize polling by monitoring ovsdb for interface changes.

ovsdb_monitor_respawn_interval

Type integer

Default `30`

The number of seconds to wait before respawning the ovsdb monitor after losing communication with it.

tunnel_types

Type list

Default `[]`

Network types supported by the agent (gre, vxlan and/or geneve).

vxlan_udp_port

Type port number

Default `4789`

Minimum Value 0

Maximum Value 65535

The UDP port to use for VXLAN tunnels.

veth_mtu

Type integer

Default `9000`

MTU size of veth interfaces

l2_population

Type boolean

Default `False`

Use ML2 l2population mechanism driver to learn remote MAC and IPs and improve tunnel scalability.

arp_responder

Type boolean

Default `False`

Enable local ARP responder if it is supported. Requires OVS 2.1 and ML2 l2population driver. Allows the switch (when supporting an overlay) to respond to an ARP request locally without performing a costly ARP broadcast into the overlay. NOTE: If enable_distributed_routing is set to True then arp_responder will automatically be set to True in the agent, regardless of the setting in the config file.

dont_fragment

Type boolean

Default `True`

Set or un-set the dont fragment (DF) bit on outgoing IP packet carrying GRE/VXLAN tunnel.

enable_distributed_routing

Type boolean

Default `False`

Make the l2 agent run in DVR mode.

drop_flows_on_start

Type boolean

Default `False`

Reset flow table on start. Setting this to True will cause brief traffic interruption.

tunnel_csum

Type boolean

Default `False`

Set or un-set the tunnel header checksum on outgoing IP packet carrying GRE/VXLAN tunnel.

baremetal_smartnic

Type boolean

Default `False`

Enable the agent to process Smart NIC ports.

explicitly_egress_direct

Type boolean

Default `False`

When set to True, the accepted egress unicast traffic will not use action NORMAL. The accepted egress packets will be taken care of in the final egress tables direct output flows for unicast traffic.

extensions

Type list

Default `[]`

Extensions list to use

network_log

rate_limit

Type integer

Default 100

Minimum Value 100

Maximum packets logging per second.

burst_limit

Type integer

Default 25

Minimum Value 25

Maximum number of packets per rate_limit.

local_output_log_base

Type string

Default <None>

Output logfile path on agent side, default syslog file.

ovs

integration_bridge

Type string

Default br-int

Integration bridge to use. Do not change this parameter unless you have a good reason to. This is the name of the OVS integration bridge. There is one per hypervisor. The integration bridge acts as a virtual patch bay. All VM VIFs are attached to this bridge and then patched according to their network connectivity.

Table 78: Deprecated Variations

Group	Name
ovs	ovs_integration_bridge

tunnel_bridge

Type string

Default br-tun

Tunnel bridge to use.

int_peer_patch_port

Type string

Default patch-tun

Peer patch port in integration bridge for tunnel bridge.

tun_peer_patch_port

Type string

Default patch-int

Peer patch port in tunnel bridge for integration bridge.

local_ip

Type ip address

Default <None>

IP address of local overlay (tunnel) network endpoint. Use either an IPv4 or IPv6 address that resides on one of the host network interfaces. The IP version of this value must match the value of the overlay_ip_version option in the ML2 plug-in configuration file on the neutron server node(s).

bridge_mappings

Type list

Default []

Comma-separated list of <physical_network>:<bridge> tuples mapping physical network names to the agents node-specific Open vSwitch bridge names to be used for flat and VLAN networks. The length of bridge names should be no more than 11. Each bridge must exist, and should have a physical network interface configured as a port. All physical networks configured on the server should have mappings to appropriate bridges on each agent. Note: If you remove a bridge from this mapping, make sure to disconnect it from the integration bridge as it wont be managed by the agent anymore.

resource_provider_bandwidths

Type list

Default []

Comma-separated list of <bridge>:<egress_bw>:<ingress_bw> tuples, showing the available bandwidth for the given bridge in the given direction. The direction is meant from VM perspective. Bandwidth is measured in kilobits per second (kbps). The bridge must appear in bridge_mappings as the value. But not all bridges in bridge_mappings must be listed here. For a bridge not listed here we neither create a resource provider in placement nor report inventories against. An omitted direction means we do not report an inventory for the corresponding class.

resource_provider_hypervisors

Type dict

Default {}

Mapping of bridges to hypervisors: <bridge>:<hypervisor>, hypervisor name is used to locate the parent of the resource provider tree. Only needs to be set in the rare case when the hypervisor name is different from the resource_provider_default_hypervisor config option value as known by the nova-compute managing that hypervisor.

resource_provider_default_hypervisor

Type string

Default <None>

The default hypervisor name used to locate the parent of the resource provider. If this option is not set, canonical name is used

resource_provider_inventory_defaults

Type dict

Default `{'allocation_ratio': 1.0, 'min_unit': 1, 'step_size': 1, 'reserved': 0}`

Key:value pairs to specify defaults used while reporting resource provider inventories. Possible keys with their types: allocation_ratio:float, max_unit:int, min_unit:int, reserved:int, step_size:int, See also: https://docs.openstack.org/api-ref/placement/#update-resource-provider-inventories

use_veth_interconnection

Type boolean

Default `False`

Use veths instead of patch ports to interconnect the integration bridge to physical networks. Support kernel without Open vSwitch patch port support so long as it is set to True.

> **Warning:** This option is deprecated for removal since Victoria. Its value may be silently ignored in the future.
>
> **Reason** Patch ports should be used to provide bridges interconnection.

datapath_type

Type string

Default `system`

Valid Values system, netdev

OVS datapath to use. system is the default value and corresponds to the kernel datapath. To enable the userspace datapath set this value to netdev.

vhostuser_socket_dir

Type string

Default `/var/run/openvswitch`

OVS vhost-user socket directory.

of_listen_address

Type ip address

Default `127.0.0.1`

Address to listen on for OpenFlow connections.

of_listen_port

Type port number

Default `6633`

Minimum Value 0

Maximum Value 65535

Port to listen on for OpenFlow connections.

of_connect_timeout

Type integer

Default `300`

Timeout in seconds to wait for the local switch connecting the controller.

of_request_timeout

Type integer

Default `300`

Timeout in seconds to wait for a single OpenFlow request.

of_inactivity_probe

Type integer

Default `10`

The inactivity_probe interval in seconds for the local switch connection to the controller. A value of 0 disables inactivity probes.

ovsdb_connection

Type string

Default `tcp:127.0.0.1:6640`

The connection string for the OVSDB backend. Will be used for all ovsdb commands and by ovsdb-client when monitoring

ssl_key_file

Type string

Default `<None>`

The SSL private key file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_cert_file

Type string

Default `<None>`

The SSL certificate file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_ca_cert_file

Type string

Default `<None>`

The Certificate Authority (CA) certificate to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ovsdb_debug

Type boolean

Default `False`

Enable OVSDB debug logs

securitygroup

`firewall_driver`

Type string

Default `<None>`

Driver for security groups firewall in the L2 agent

`enable_security_group`

Type boolean

Default `True`

Controls whether the neutron security group API is enabled in the server. It should be false when using no security groups or using the nova security group API.

`enable_ipset`

Type boolean

Default `True`

Use ipset to speed-up the iptables based security groups. Enabling ipset support requires that ipset is installed on L2 agent node.

`permitted_ethertypes`

Type list

Default `[]`

Comma-separated list of ethertypes to be permitted, in hexadecimal (starting with 0x). For example, 0x4008 to permit InfiniBand.

xenapi

`connection_url`

Type string

Default `<None>`

URL for connection to XenServer/Xen Cloud Platform.

`connection_username`

Type string

Default `<None>`

Username for connection to XenServer/Xen Cloud Platform.

`connection_password`

Type string

Default `<None>`

Password for connection to XenServer/Xen Cloud Platform.

9.1.6 sriov_agent.ini

DEFAULT

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 79: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 80: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 81: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO',`

```
'oslo.messaging=INFO', 'oslo_messaging=INFO',
'iso8601=WARN', 'requests.packages.urllib3.
connectionpool=WARN', 'urllib3.connectionpool=WARN',
'websocket=WARN', 'requests.packages.
urllib3.util.retry=WARN', 'urllib3.util.
retry=WARN', 'keystonemiddleware=WARN', 'routes.
middleware=WARN', 'stevedore=WARN', 'taskflow=WARN',
'keystoneauth=WARN', 'oslo.cache=INFO',
'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance:  %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance:  %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

`extensions`

Type list

Default `[]`

Extensions list to use

sriov_nic

`physical_device_mappings`

Type list

Default `[]`

Comma-separated list of <physical_network>:<network_device> tuples mapping physical network names to the agents node-specific physical network device interfaces of SR-IOV physical function to be used for VLAN networks. All physical networks listed in network_vlan_ranges on the server should have mappings to appropriate interfaces on each agent.

`exclude_devices`

Type list

Default `[]`

Comma-separated list of <network_device>:<vfs_to_exclude> tuples, mapping network_device to the agents node-specific list of virtual functions that should not be used for virtual networking. vfs_to_exclude is a semicolon-separated list of virtual functions to exclude from network_device. The network_device in the mapping should appear in the physical_device_mappings list.

`resource_provider_bandwidths`

Type list

Default `[]`

Comma-separated list of <network_device>:<egress_bw>:<ingress_bw> tuples, showing the available bandwidth for the given device in the given direction. The direction is meant from VM perspective. Bandwidth is measured in kilobits per second (kbps). The device must appear in physical_device_mappings as the value. But not all devices in physical_device_mappings must be listed here. For a device not listed here we neither create a resource provider in placement nor report inventories against. An omitted direction means we do not report an inventory for the corresponding class.

`resource_provider_hypervisors`

Type dict

Default `{}`

Mapping of network devices to hypervisors: <network_device>:<hypervisor>, hypervisor name is used to locate the parent of the resource provider tree. Only needs to be set in the rare case when the hypervisor name is different from the resource_provider_default_hypervisor config option value as known by the nova-compute managing that hypervisor.

resource_provider_default_hypervisor

Type string

Default `<None>`

The default hypervisor name used to locate the parent of the resource provider. If this option is not set, canonical name is used

resource_provider_inventory_defaults

Type dict

Default `{'allocation_ratio': 1.0, 'min_unit': 1, 'step_size': 1, 'reserved': 0}`

Key:value pairs to specify defaults used while reporting resource provider inventories. Possible keys with their types: allocation_ratio:float, max_unit:int, min_unit:int, reserved:int, step_size:int, See also: https://docs.openstack.org/api-ref/placement/#update-resource-provider-inventories

9.1.7 ovn.ini

DEFAULT

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 82: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default %Y-%m-%d %H:%M:%S

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default <None>

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 83: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default <None>

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 84: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default False

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default False

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default False

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

ovn

ovn_nb_connection

Type string

Default `tcp:127.0.0.1:6641`

The connection string for the OVN_Northbound OVSDB. Use tcp:IP:PORT for TCP connection. Use ssl:IP:PORT for SSL connection. The ovn_nb_private_key, ovn_nb_certificate and ovn_nb_ca_cert are mandatory. Use unix:FILE for unix domain socket connection.

ovn_nb_private_key

Type string

Default `''`

The PEM file with private key for SSL connection to OVN-NB-DB

ovn_nb_certificate

Type string

Default `''`

The PEM file with certificate that certifies the private key specified in ovn_nb_private_key

ovn_nb_ca_cert

Type string

Default `''`

The PEM file with CA certificate that OVN should use to verify certificates presented to it by SSL peers

ovn_sb_connection

Type string

Default `tcp:127.0.0.1:6642`

The connection string for the OVN_Southbound OVSDB. Use tcp:IP:PORT for TCP connection. Use ssl:IP:PORT for SSL connection. The ovn_sb_private_key, ovn_sb_certificate and ovn_sb_ca_cert are mandatory. Use unix:FILE for unix domain socket connection.

ovn_sb_private_key

Type string

Default `''`

The PEM file with private key for SSL connection to OVN-SB-DB

ovn_sb_certificate

Type string

Default `''`

The PEM file with certificate that certifies the private key specified in ovn_sb_private_key

ovn_sb_ca_cert

Type string

Default `''`

The PEM file with CA certificate that OVN should use to verify certificates presented to it by SSL peers

ovsdb_connection_timeout

Type integer

Default `180`

Timeout in seconds for the OVSDB connection transaction

ovsdb_retry_max_interval

Type integer

Default `180`

Max interval in seconds between each retry to get the OVN NB and SB IDLs

ovsdb_probe_interval

Type integer

Default `60000`

Minimum Value 0

The probe interval in for the OVSDB session in milliseconds. If this is zero, it disables the connection keepalive feature. If non-zero the value will be forced to at least 1000 milliseconds. Defaults to 60 seconds.

neutron_sync_mode

Type string

Default `log`

Valid Values off, log, repair

The synchronization mode of OVN_Northbound OVSDB with Neutron DB. off - synchronization is off log - during neutron-server startup, check to see if OVN is in sync with the Neutron database. Log warnings for any inconsistencies found so that an admin can investigate repair - during neutron-server startup, automatically create resources found in Neutron but not in OVN. Also remove resources from OVN that are no longer in Neutron.

ovn_l3_mode

Type boolean

Default `True`

Whether to use OVN native L3 support. Do not change the value for existing deployments that contain routers.

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** This option is no longer used. Native L3 support in OVN is always used.

ovn_l3_scheduler

Type string

Default `leastloaded`

Valid Values leastloaded, chance

The OVN L3 Scheduler type used to schedule router gateway ports on hypervisors/chassis. leastloaded - chassis with fewest gateway ports selected chance - chassis randomly selected

enable_distributed_floating_ip

Type boolean

Default `False`

Enable distributed floating IP support. If True, the NAT action for floating IPs will be done locally and not in the centralized gateway. This saves the path to the external network. This requires the user to configure the physical network map (i.e. ovn-bridge-mappings) on each compute node.

vif_type

Type string

Default `ovs`

Valid Values ovs, vhostuser

Type of VIF to be used for ports valid values are (ovs, vhostuser) default ovs

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** The port VIF type is now determined based on the OVN chassis information when the port is bound to a host.

vhost_sock_dir

Type string

Default `/var/run/openvswitch`

The directory in which vhost virtio socket is created by all the vswitch daemons

dhcp_default_lease_time

Type integer

Default `43200`

Default least time (in seconds) to use with OVNs native DHCP service.

ovsdb_log_level

Type string

Default `INFO`

Valid Values CRITICAL, ERROR, WARNING, INFO, DEBUG

The log level used for OVSDB

ovn_metadata_enabled

Type boolean

Default `False`

Whether to use metadata service.

dns_servers

Type list

Default `[]`

Comma-separated list of the DNS servers which will be used as forwarders if a subnets dns_nameservers field is empty. If both subnets dns_nameservers and this option is empty, then the DNS resolvers on the host running the neutron server will be used.

ovn_dhcp4_global_options

Type dict

Default `{}`

Dictionary of global DHCPv4 options which will be automatically set on each subnet upon creation and on all existing subnets when Neutron starts. An empty value for a DHCP option will cause that option to be unset globally. EXAMPLES: - ntp_server:1.2.3.4,wpad:1.2.3.5 - Set ntp_server and wpad - ntp_server:,wpad:1.2.3.5 - Unset ntp_server and set wpad See the ovn-nb(5) man page for available options.

ovn_dhcp6_global_options

Type dict

Default `{}`

Dictionary of global DHCPv6 options which will be automatically set on each subnet upon creation and on all existing subnets when Neutron starts. An empty value for a DHCP option will cause that option to be unset globally. EXAMPLES: - ntp_server:1.2.3.4,wpad:1.2.3.5 - Set ntp_server and wpad - ntp_server:,wpad:1.2.3.5 - Unset ntp_server and set wpad See the ovn-nb(5) man page for available options.

ovn_emit_need_to_frag

Type boolean

Default `False`

Configure OVN to emit need to frag packets in case of MTU mismatch. Before enabling this configuration make sure that its supported by the host kernel (version >= 5.2) or by checking the output of the following command: ovs-appctl -t ovs-vswitchd dpif/show-dp-features br-int | grep Check pkt length action.

ovs

ovsdb_timeout

Type integer

Default `10`

Timeout in seconds for ovsdb commands. If the timeout expires, ovsdb commands will fail with ALARMCLOCK error.

bridge_mac_table_size

Type integer

Default `50000`

The maximum number of MAC addresses to learn on a bridge managed by the Neutron OVS agent. Values outside a reasonable range (10 to 1,000,000) might be overridden by Open vSwitch according to the documentation.

igmp_snooping_enable

Type boolean

Default `False`

Enable IGMP snooping for integration bridge. If this option is set to True, support for Internet Group Management Protocol (IGMP) is enabled in integration bridge. Setting this option to True will also enable Open vSwitch mcast-snooping-disable-flood-unregistered flag. This option will disable flooding of unregistered multicast packets to all ports. The switch will send unregistered multicast packets only to ports connected to multicast routers.

9.1.8 dhcp_agent.ini

DEFAULT

ovs_integration_bridge

Type string

Default `br-int`

Name of Open vSwitch bridge to use

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** This variable is a duplicate of OVS.integration_bridge. To be removed in W.

ovs_use_veth

Type boolean

Default `False`

Uses veth for an OVS interface or not. Support kernels with limited namespace support (e.g. RHEL 6.5) and rate limiting on routers gateway port so long as ovs_use_veth is set to True.

interface_driver

Type string

Default `<None>`

The driver used to manage the virtual interface.

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

resync_interval

Type integer

Default `5`

The DHCP agent will resync its state with Neutron to recover from any transient notification or RPC errors. The interval is maximum number of seconds between attempts. The resync can be done more often based on the events triggered.

resync_throttle

Type integer

Default `1`

Throttle the number of resync state events between the local DHCP state and Neutron to only once per resync_throttle seconds. The value of throttle introduces a minimum interval between resync state events. Otherwise the resync may end up in a busy-loop. The value must be less than resync_interval.

dhcp_driver

Type string

Default `neutron.agent.linux.dhcp.Dnsmasq`

The driver used to manage the DHCP server.

enable_isolated_metadata

Type boolean

Default `False`

The DHCP server can assist with providing metadata support on isolated networks. Setting this value to True will cause the DHCP server to append specific host routes to the DHCP request. The metadata service will only be activated when the subnet does not contain any router port. The guest instance must be configured to request host routes via DHCP (Option 121). This option doesnt have any effect when force_metadata is set to True.

force_metadata

Type boolean

Default `False`

In some cases the Neutron router is not present to provide the metadata IP but the DHCP server can be used to provide this info. Setting this value will force the DHCP server to append specific host routes to the DHCP request. If this option is set, then the metadata service will be activated for all the networks.

enable_metadata_network

Type boolean

Default `False`

Allows for serving metadata requests coming from a dedicated metadata access network whose CIDR is 169.254.169.254/16 (or larger prefix), and is connected to a Neutron router from which the VMs send metadata:1 request. In this case DHCP Option 121 will not be injected in VMs, as they will be able to reach 169.254.169.254 through a router. This option requires enable_isolated_metadata = True.

num_sync_threads

Type integer

Default `4`

Number of threads to use during sync process. Should not exceed connection pool size configured on server.

bulk_reload_interval

Type integer

Default `0`

Minimum Value 0

Time to sleep between reloading the DHCP allocations. This will only be invoked if the value is not 0. If a network has N updates in X seconds then we will reload once with the port changes in the X seconds and not N times.

dhcp_confs

Type string

Default `$state_path/dhcp`

Location to store DHCP server config files.

dnsmasq_config_file

Type string

Default `''`

Override the default dnsmasq settings with this file.

dnsmasq_dns_servers

Type list

Default `[]`

Comma-separated list of the DNS servers which will be used as forwarders.

dnsmasq_base_log_dir

Type string

Default `<None>`

Base log dir for dnsmasq logging. The log contains DHCP and DNS log information and is useful for debugging issues with either DHCP or DNS. If this section is null, disable dnsmasq log.

dnsmasq_local_resolv

Type boolean

Default `False`

Enables the dnsmasq service to provide name resolution for instances via DNS resolvers on the host running the DHCP agent. Effectively removes the no-resolv option from the dnsmasq process arguments. Adding custom DNS resolvers to the dnsmasq_dns_servers option disables this feature.

dnsmasq_lease_max

Type integer

Default `16777216`

Limit number of leases to prevent a denial-of-service.

dhcp_broadcast_reply

Type boolean

Default `False`

Use broadcast in DHCP replies.

dhcp_renewal_time

Type integer

Default `0`

DHCP renewal time T1 (in seconds). If set to 0, it will default to half of the lease time.

dhcp_rebinding_time

Type integer

Default `0`

DHCP rebinding time T2 (in seconds). If set to 0, it will default to 7/8 of the lease time.

dnsmasq_enable_addr6_list

Type boolean

Default `False`

Enable dhcp-host entry with list of addresses when port has multiple IPv6 addresses in the same subnet.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 85: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 86: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 87: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

availability_zone

Type string

Default `nova`

Availability zone of this node

report_interval

Type floating point

Default `30`

Seconds between nodes reporting state to server; should be less than agent_down_time, best if it is half or less than agent_down_time.

log_agent_heartbeats

Type boolean

Default `False`

Log agent heartbeats

ovs

ovsdb_connection

Type string

Default `tcp:127.0.0.1:6640`

The connection string for the OVSDB backend. Will be used for all ovsdb commands and by ovsdb-client when monitoring

ssl_key_file

Type string

Default `<None>`

The SSL private key file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_cert_file

Type string

Default `<None>`

The SSL certificate file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_ca_cert_file

Type string

Default `<None>`

The Certificate Authority (CA) certificate to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ovsdb_debug

Type boolean

Default `False`

Enable OVSDB debug logs

ovsdb_timeout

Type integer

Default `10`

Timeout in seconds for ovsdb commands. If the timeout expires, ovsdb commands will fail with ALARMCLOCK error.

bridge_mac_table_size

Type integer

Default `50000`

The maximum number of MAC addresses to learn on a bridge managed by the Neutron OVS agent. Values outside a reasonable range (10 to 1,000,000) might be overridden by Open vSwitch according to the documentation.

igmp_snooping_enable

Type boolean

Default `False`

Enable IGMP snooping for integration bridge. If this option is set to True, support for Internet Group Management Protocol (IGMP) is enabled in integration bridge. Setting this option to True will also enable Open vSwitch mcast-snooping-disable-flood-unregistered flag. This option will disable flooding of unregistered multicast packets to all ports. The switch will send unregistered multicast packets only to ports connected to multicast routers.

9.1.9 l3_agent.ini

DEFAULT

ovs_integration_bridge

Type string

Default `br-int`

Name of Open vSwitch bridge to use

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** This variable is a duplicate of OVS.integration_bridge. To be removed in W.

ovs_use_veth

Type boolean

Default `False`

Uses veth for an OVS interface or not. Support kernels with limited namespace support (e.g. RHEL 6.5) and rate limiting on routers gateway port so long as ovs_use_veth is set to True.

interface_driver

Type string

Default `<None>`

The driver used to manage the virtual interface.

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

agent_mode

Type string

Default `legacy`

Valid Values dvr, dvr_snat, legacy, dvr_no_external

The working mode for the agent. Allowed modes are: legacy - this preserves the existing behavior where the L3 agent is deployed on a centralized networking node to provide L3 services like DNAT, and SNAT. Use this mode if you do not want to adopt DVR. dvr - this mode enables DVR functionality and must be used for an L3 agent that runs on a compute host. dvr_snat - this enables centralized SNAT support in conjunction with DVR. This mode must be used for an L3 agent running on a centralized node (or in single-host deployments, e.g. devstack). dvr_no_external - this mode enables only East/West DVR routing functionality for a L3 agent that runs on a compute host, the North/South functionality such as DNAT and SNAT will be provided by the centralized network node that is running in dvr_snat mode. This mode should be used when there is no external network connectivity on the compute host.

metadata_port

Type port number

Default `9697`

Minimum Value 0

Maximum Value 65535

TCP Port used by Neutron metadata namespace proxy.

handle_internal_only_routers

Type boolean

Default `True`

Indicates that this L3 agent should also handle routers that do not have an external network gateway configured. This option should be True only for a single agent in a Neutron deployment, and may be False for all agents if all routers must have an external network gateway.

ipv6_gateway

Type string

Default `''`

With IPv6, the network used for the external gateway does not need to have an associated subnet, since the automatically assigned link-local address (LLA) can be used. However, an IPv6 gateway address is needed for use as the next-hop for the default route. If no IPv6 gateway address is configured here, (and only then) the neutron router will be configured to get its default route from router advertisements (RAs) from the upstream router; in which case the upstream router must also be configured to send these RAs. The ipv6_gateway, when configured, should be the LLA of the interface on the upstream router. If a next-hop using a global unique address (GUA) is desired, it needs to be done via a subnet allocated to the network and not through this parameter.

prefix_delegation_driver

Type string

Default `dibbler`

Driver used for ipv6 prefix delegation. This needs to be an entry point defined in the neutron.agent.linux.pd_drivers namespace. See setup.cfg for entry points included with the neutron source.

enable_metadata_proxy

Type boolean

Default `True`

Allow running metadata proxy.

metadata_access_mark

Type string

Default `0x1`

Iptables mangle mark used to mark metadata valid requests. This mark will be masked with 0xffff so that only the lower 16 bits will be used.

external_ingress_mark

Type string

Default `0x2`

Iptables mangle mark used to mark ingress from external network. This mark will be masked with 0xffff so that only the lower 16 bits will be used.

radvd_user

Type string

Default `''`

The username passed to radvd, used to drop root privileges and change user ID to username and group ID to the primary group of username. If no user specified (by default), the user executing the L3 agent will be passed. If root specified, because radvd is spawned as root, no username parameter will be passed.

cleanup_on_shutdown

Type boolean

Default `False`

Delete all routers on L3 agent shutdown. For L3 HA routers it includes a shutdown of keepalived and the state change monitor. NOTE: Setting to True could affect the data plane when stopping or restarting the L3 agent.

keepalived_use_no_track

Type boolean

Default `True`

If keepalived without support for no_track option is used, this should be set to False. Support for this option was introduced in keepalived 2.x

periodic_interval

Type integer

Default `40`

Seconds between running periodic tasks.

api_workers

Type integer

Default `<None>`

Number of separate API worker processes for service. If not specified, the default is equal to the number of CPUs available for best performance, capped by potential RAM usage.

rpc_workers

Type integer

Default `<None>`

Number of RPC worker processes for service. If not specified, the default is equal to half the number of API workers.

rpc_state_report_workers

Type integer

Default `1`

Number of RPC worker processes dedicated to state reports queue.

periodic_fuzzy_delay

Type integer

Default `5`

Range of seconds to randomly delay when starting the periodic task scheduler to reduce stampeding. (Disable by setting to 0)

ha_confs_path

Type string

Default `$state_path/ha_confs`

Location to store keepalived config files

ha_vrrp_auth_type

Type string

Default `PASS`

Valid Values AH, PASS

VRRP authentication type

ha_vrrp_auth_password

Type string

Default `<None>`

VRRP authentication password

ha_vrrp_advert_int

Type integer

Default `2`

The advertisement interval in seconds

ha_keepalived_state_change_server_threads

Type integer

Default `(1 + <num_of_cpus>) / 2`

Minimum Value 1

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Number of concurrent threads for keepalived server connection requests. More threads create a higher CPU load on the agent node.

ha_vrrp_health_check_interval

Type integer

Default `0`

The VRRP health check interval in seconds. Values > 0 enable VRRP health checks. Setting it to 0 disables VRRP health checks. Recommended value is 5. This will cause pings to be sent to the gateway IP address(es) - requires ICMP_ECHO_REQUEST to be enabled on the gateway(s). If a gateway fails, all routers will be reported as primary, and a primary election will be repeated in a round-robin fashion, until one of the routers restores the gateway connection.

pd_confs

Type string

Default `$state_path/pd`

Location to store IPv6 PD files.

vendor_pen

Type string

Default `8888`

A decimal value as Vendors Registered Private Enterprise Number as required by RFC3315 DUID-EN.

ra_confs

Type string

Default `$state_path/ra`

Location to store IPv6 RA config files

min_rtr_adv_interval

Type integer

Default `30`

MinRtrAdvInterval setting for radvd.conf

max_rtr_adv_interval

Type integer

Default `100`

MaxRtrAdvInterval setting for radvd.conf

agent

availability_zone

Type string

Default `nova`

Availability zone of this node

report_interval

Type floating point

Default `30`

Seconds between nodes reporting state to server; should be less than agent_down_time, best if it is half or less than agent_down_time.

log_agent_heartbeats

Type boolean

Default `False`

Log agent heartbeats

extensions

Type list

Default `[]`

Extensions list to use

network_log

rate_limit

Type integer

Default 100

Minimum Value 100

Maximum packets logging per second.

burst_limit

Type integer

Default 25

Minimum Value 25

Maximum number of packets per rate_limit.

local_output_log_base

Type string

Default <None>

Output logfile path on agent side, default syslog file.

ovs

ovsdb_connection

Type string

Default tcp:127.0.0.1:6640

The connection string for the OVSDB backend. Will be used for all ovsdb commands and by ovsdb-client when monitoring

ssl_key_file

Type string

Default <None>

The SSL private key file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_cert_file

Type string

Default <None>

The SSL certificate file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_ca_cert_file

Type string

Default <None>

The Certificate Authority (CA) certificate to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ovsdb_debug

Type boolean

Default `False`

Enable OVSDB debug logs

ovsdb_timeout

Type integer

Default `10`

Timeout in seconds for ovsdb commands. If the timeout expires, ovsdb commands will fail with ALARMCLOCK error.

bridge_mac_table_size

Type integer

Default `50000`

The maximum number of MAC addresses to learn on a bridge managed by the Neutron OVS agent. Values outside a reasonable range (10 to 1,000,000) might be overridden by Open vSwitch according to the documentation.

igmp_snooping_enable

Type boolean

Default `False`

Enable IGMP snooping for integration bridge. If this option is set to True, support for Internet Group Management Protocol (IGMP) is enabled in integration bridge. Setting this option to True will also enable Open vSwitch mcast-snooping-disable-flood-unregistered flag. This option will disable flooding of unregistered multicast packets to all ports. The switch will send unregistered multicast packets only to ports connected to multicast routers.

9.1.10 metadata_agent.ini

DEFAULT

metadata_proxy_socket

Type string

Default `$state_path/metadata_proxy`

Location for Metadata Proxy UNIX domain socket.

metadata_proxy_user

Type string

Default `''`

User (uid or name) running metadata proxy after its initialization (if empty: agent effective user).

metadata_proxy_group

Type string

Default `''`

Group (gid or name) running metadata proxy after its initialization (if empty: agent effective group).

auth_ca_cert

Type string

Default `<None>`

Certificate Authority public key (CA cert) file for ssl

nova_metadata_host

Type host address

Default `127.0.0.1`

IP address or DNS name of Nova metadata server.

nova_metadata_port

Type port number

Default `8775`

Minimum Value 0

Maximum Value 65535

TCP Port used by Nova metadata server.

metadata_proxy_shared_secret

Type string

Default `''`

When proxying metadata requests, Neutron signs the Instance-ID header with a shared secret to prevent spoofing. You may select any string for a secret, but it must match here and in the configuration used by the Nova Metadata Server. NOTE: Nova uses the same config key, but in [neutron] section.

nova_metadata_protocol

Type string

Default `http`

Valid Values http, https

Protocol to access nova metadata, http or https

nova_metadata_insecure

Type boolean

Default `False`

Allow to perform insecure SSL (https) requests to nova metadata

nova_client_cert

Type string

Default `''`

Client certificate for nova metadata api server.

nova_client_priv_key

Type string

Default `''`

Private key of client certificate.

metadata_proxy_socket_mode

Type string

Default `deduce`

Valid Values deduce, user, group, all

Metadata Proxy UNIX domain socket mode, 4 values allowed: deduce: deduce mode from metadata_proxy_user/group values, user: set metadata proxy socket mode to 0o644, to use when metadata_proxy_user is agent effective user or root, group: set metadata proxy socket mode to 0o664, to use when metadata_proxy_group is agent effective group or root, all: set metadata proxy socket mode to 0o666, to use otherwise.

metadata_workers

Type integer

Default <num_of_cpus> / 2

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Number of separate worker processes for metadata server (defaults to 2 when used with ML2/OVN and half of the number of CPUs with other backend drivers)

metadata_backlog

Type integer

Default `4096`

Number of backlog requests to configure the metadata server socket with

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 88: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 89: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 90: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default days

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default 30

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default 200

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default none

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default %(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default %(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

report_interval

Type floating point

Default `30`

Seconds between nodes reporting state to server; should be less than agent_down_time, best if it is half or less than agent_down_time.

log_agent_heartbeats

Type boolean

Default `False`

Log agent heartbeats

cache

config_prefix

Type string

Default `cache.oslo`

Prefix for building the configuration dictionary for the cache region. This should not need to be changed unless there is another dogpile.cache region with the same configuration name.

expiration_time

Type integer

Default `600`

Default TTL, in seconds, for any cached item in the dogpile.cache region. This applies to any cached method that doesnt have an explicit cache expiration time defined for it.

backend

Type string

Default `dogpile.cache.null`

Valid Values oslo_cache.memcache_pool, oslo_cache.dict, oslo_cache.mongo, oslo_cache.etcd3gw, dogpile.cache.memcached, dogpile.cache.pylibmc, dogpile.cache.bmemcached, dogpile.cache.dbm, dogpile.cache.redis, dogpile.cache.memory, dogpile.cache.memory_pickle, dogpile.cache.null

Cache backend module. For eventlet-based or environments with hundreds of threaded servers, Memcache with pooling (oslo_cache.memcache_pool) is recommended. For environments with less than 100 threaded servers, Memcached (dogpile.cache.memcached) or Redis (dogpile.cache.redis) is recommended. Test environments with a single instance of the server can use the dogpile.cache.memory backend.

backend_argument

Type multi-valued

Default `''`

Arguments supplied to the backend module. Specify this option once per argument to be passed to the dogpile.cache backend. Example format: <argname>:<value>.

proxies

Type list

Default `[]`

Proxy classes to import that will affect the way the dogpile.cache backend functions. See the dogpile.cache documentation on changing-backend-behavior.

enabled

Type boolean

Default `False`

Global toggle for caching.

debug_cache_backend

Type boolean

Default `False`

Extra debugging from the cache backend (cache keys, get/set/delete/etc calls). This is only really useful if you need to see the specific cache-backend get/set/delete calls with the keys/values. Typically this should be left set to false.

memcache_servers

Type list

Default `['localhost:11211']`

Memcache servers in the format of host:port. (dogpile.cache.memcached and oslo_cache.memcache_pool backends only). If a given host refer to an IPv6 or a given domain refer to IPv6 then you should prefix the given address with the address family (`inet6`) (e.g `inet6[::1]:11211`, `inet6:[fd12:3456:789a:1::1]:11211`, `inet6:[controller-0.internalapi]:11211`). If the address family is not given then default address family used will be `inet` which correspond to IPv4

memcache_dead_retry

Type integer

Default `300`

Number of seconds memcached server is considered dead before it is tried again. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

memcache_socket_timeout

Type floating point

Default `1.0`

Timeout in seconds for every call to a server. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

memcache_pool_maxsize

Type integer

Default `10`

Max total number of open connections to every memcached server. (oslo_cache.memcache_pool backend only).

memcache_pool_unused_timeout

Type integer

Default `60`

Number of seconds a connection to memcached is held unused in the pool before it is closed. (oslo_cache.memcache_pool backend only).

memcache_pool_connection_get_timeout

Type integer

Default `10`

Number of seconds that an operation will wait to get a memcache client connection.

tls_enabled

Type boolean

Default `False`

Global toggle for TLS usage when comunicating with the caching servers.

tls_cafile

Type string

Default `<None>`

Path to a file of concatenated CA certificates in PEM format necessary to establish the caching servers authenticity. If tls_enabled is False, this option is ignored.

tls_certfile

Type string

Default `<None>`

Path to a single file in PEM format containing the clients certificate as well as any number of CA certificates needed to establish the certificates authenticity. This file is only required when client side authentication is necessary. If tls_enabled is False, this option is ignored.

tls_keyfile

Type string

Default `<None>`

Path to a single file containing the clients private key in. Otherwhise the private key will be taken from the file specified in tls_certfile. If tls_enabled is False, this option is ignored.

tls_allowed_ciphers

Type string

Default `<None>`

Set the available ciphers for sockets created with the TLS context. It should be a string in the OpenSSL cipher list format. If not specified, all OpenSSL enabled ciphers will be available.

enable_socket_keepalive

Type boolean

Default `False`

Global toggle for the socket keepalive of dogpiles pymemcache backend

socket_keepalive_idle

Type integer

Default `1`

Minimum Value 0

The time (in seconds) the connection needs to remain idle before TCP starts sending keepalive probes. Should be a positive integer most greater than zero.

socket_keepalive_interval

Type integer

Default `1`

Minimum Value 0

The time (in seconds) between individual keepalive probes. Should be a positive integer greater than zero.

socket_keepalive_count

Type integer

Default `1`

Minimum Value 0

The maximum number of keepalive probes TCP should send before dropping the connection. Should be a positive integer greater than zero.

enable_retry_client

Type boolean

Default `False`

Enable retry client mechanisms to handle failure. Those mechanisms can be used to wrap all kind of pymemcache clients. The wrapper allows you to define how many attempts to make and how long to wait between attemots.

retry_attempts

Type integer

Default `2`

Minimum Value 1

Number of times to attempt an action before failing.

retry_delay

Type floating point

Default `0`

Number of seconds to sleep between each attempt.

hashclient_retry_attempts

Type integer

Default `2`

Minimum Value 1

Amount of times a client should be tried before it is marked dead and removed from the pool in the HashClients internal mechanisms.

hashclient_retry_delay

Type floating point

Default `1`

Time in seconds that should pass between retry attempts in the HashClients internal mechanisms.

dead_timeout

Type floating point

Default 60

Time in seconds before attempting to add a node back in the pool in the HashClients internal mechanisms.

9.1.11 Neutron Metering system

The Neutron metering service enables operators to account the traffic in/out of the OpenStack environment. The concept is quite simple, operators can create metering labels, and decide if the labels are applied to all projects (tenants) or if they are applied to a specific one. Then, the operator needs to create traffic rules in the metering labels. The traffic rules are used to match traffic in/out of the OpenStack environment, and the accounting of packets and bytes is sent to the notification queue for further processing by Ceilometer (or some other system that is consuming that queue). The message sent in the queue is of type `event`. Therefore, it requires an event processing configuration to be added/enabled in Ceilometer.

The metering agent has the following configurations:

- `driver`: the driver used to implement the metering rules. The default is `neutron.services.metering.drivers.noop`, which means, we do not execute anything in the networking host. The only driver implemented so far is `neutron.services.metering.drivers.iptables.iptables_driver.IptablesMeteringDriver`. Therefore, only `iptables` is supported so far;
- `measure_interval`: the interval in seconds used to gather the bytes and packets information from the network plane. The default value is `30` seconds;
- `report_interval`: the interval in secodns used to generated the report (message) of the data that is gathered. The default value is `300` seconds.
- `granular_traffic_data`: Defines if the metering agent driver should present traffic data in a granular fashion, instead of grouping all of the traffic data for all projects and routers where the labels were assigned to. The default value is `False` for backward compatibility.

Non-granular traffic messages

The non-granular (`granular_traffic_data = False`) traffic messages (here also called as legacy) have the following format; bear in mind that if labels are shared, then the counters are for all routers of all projects where the labels were applied.

```
{
 "pkts": "<the number of packets that matched the rules of the␣
↪labels>",
 "bytes": "<the number of bytes that matched the rules of the␣
↪labels>",
 "time": "<seconds between the first data collection and the last␣
↪one>",
 "first_update": "timeutils.utcnow_ts() of the first collection",
 "last_update": "timeutils.utcnow_ts() of the last collection",
 "host": "<neutron metering agent host name>",
 "label_id": "<the label id>",
```

(continues on next page)

(continued from previous page)

```
"tenant_id": "<the tenant id>"
}
```

The first_update and last_update timestamps represent the moment when the first and last data collection happened within the report interval. On the other hand, the time represents the difference between those two timestamp.

The tenant_id is only consistent when labels are not shared. Otherwise, they will contain the project id of the last router of the last project processed when the agent is started up. In other words, it is better not use it when dealing with shared labels.

All of the messages generated in this configuration mode are sent to the message bus as l3.meter events.

Granular traffic messages

The granular (granular_traffic_data = True) traffic messages allow operators to obtain granular information for shared metering labels. Therefore, a single label, when configured as shared=True and applied in all projects/routers of the environment, it will generate data in a granular fashion.

It (the metering agent) will account the traffic counter data in the following granularities.

- label all of the traffic counter for a given label. One must bear in mind that a label can be assigned to multiple routers. Therefore, this granularity represents all aggregation for all data for all routers of all projects where the label has been applied.
- router all of the traffic counter for all labels that are assigned to the router.
- project all of the traffic counters for all labels of all routers that a project has.
- router-label all of the traffic counters for a router and the given label.
- project-label all of the traffic counters for all routers of a project that have a given label.

Each granularity presented here is sent to the message bus with different events types that vary according to the granularity. The mapping between granularity and event type is presented as follows.

- label event type l3.meter.label.
- router event type l3.meter.router.
- project event type l3.meter.project..
- router-label event type l3.meter.label_router.
- project-label event type l3.meter.label_project.

Furthermore, we have metadata that is appended to the messages depending on the granularity. As follows we present the mapping between the granularities and the metadata that will be available.

- label, router-label, and project-label granularities have the metadata label_id, label_name, label_shared, project_id (if shared, this value will come with all for the label granularity), and router_id (only for router-label granularity).
- The router granularity has the router_id and project_id metadata.
- The project granularity only has the project_id metadata.

The message will also contain some attributes that can be found in the legacy mode such as `bytes`, `pkts`, `time`, `first_update`, `last_update`, and `host`. As follows we present an example of JSON message with all of the possible attributes.


```
{
"resource_id": "router-f0f745d9a59c47fdbbdd187d718f9e41-label-
↪00c714f1-49c8-462c-8f5d-f05f21e035c7",
 "project_id": "f0f745d9a59c47fdbbdd187d718f9e41",
 "first_update": 1591058790,
 "bytes": 0,
 "label_id": "00c714f1-49c8-462c-8f5d-f05f21e035c7",
 "label_name": "test1",
 "last_update": 1591059037,
 "host": "<hostname>",
 "time": 247,
 "pkts": 0,
 "label_shared": true
 }
```


The `resource_id` is a unique identified for the resource being monitored. Here we consider a resource to be any of the granularities that we handle.

Sample of metering_agent.ini

As follows we present all of the possible configuration one can use in the metering agent init file.

DEFAULT

ovs_integration_bridge

Type string

Default `br-int`

Name of Open vSwitch bridge to use

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** This variable is a duplicate of OVS.integration_bridge. To be removed in W.

ovs_use_veth

Type boolean

Default `False`

Uses veth for an OVS interface or not. Support kernels with limited namespace support (e.g. RHEL 6.5) and rate limiting on routers gateway port so long as ovs_use_veth is set to True.

interface_driver

Type string

Default `<None>`

The driver used to manage the virtual interface.

rpc_response_max_timeout

Type integer

Default `600`

Maximum seconds to wait for a response from an RPC call.

driver

Type string

Default `neutron.services.metering.drivers.noop.noop_driver.NoopMeteringDriver`

Metering driver

measure_interval

Type integer

Default `30`

Interval between two metering measures

report_interval

Type integer

Default `300`

Interval between two metering reports

granular_traffic_data

Type boolean

Default `False`

Defines if the metering agent driver should present traffic data in a granular fashion, instead of grouping all of the traffic data for all projects and routers where the labels were assigned to. The default value is *False* for backward compatibility.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 91: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 92: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 93: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

use_eventlog

Type boolean

Default `False`

Log output to Windows Event Log.

log_rotate_interval

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

log_rotate_interval_type

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type integer

Default `30`

Maximum number of rotated log files.

max_logfile_size_mb

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

log_rotation_type

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

logging_context_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

logging_default_format_string

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

agent

report_interval

Type floating point

Default `30`

Seconds between nodes reporting state to server; should be less than agent_down_time, best if it is half or less than agent_down_time.

log_agent_heartbeats

Type boolean

Default `False`

Log agent heartbeats

ovs

ovsdb_connection

Type string

Default `tcp:127.0.0.1:6640`

The connection string for the OVSDB backend. Will be used for all ovsdb commands and by ovsdb-client when monitoring

ssl_key_file

Type string

Default `<None>`

The SSL private key file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_cert_file

Type string

Default `<None>`

The SSL certificate file to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ssl_ca_cert_file

Type string

Default `<None>`

The Certificate Authority (CA) certificate to use when interacting with OVSDB. Required when using an ssl: prefixed ovsdb_connection

ovsdb_debug

Type boolean

Default `False`

Enable OVSDB debug logs

ovsdb_timeout

Type integer

Default `10`

Timeout in seconds for ovsdb commands. If the timeout expires, ovsdb commands will fail with ALARMCLOCK error.

bridge_mac_table_size

Type integer

Default `50000`

The maximum number of MAC addresses to learn on a bridge managed by the Neutron OVS agent. Values outside a reasonable range (10 to 1,000,000) might be overridden by Open vSwitch according to the documentation.

igmp_snooping_enable

Type boolean

Default `False`

Enable IGMP snooping for integration bridge. If this option is set to True, support for Internet Group Management Protocol (IGMP) is enabled in integration bridge. Setting this option to True will also enable Open vSwitch mcast-snooping-disable-flood-unregistered flag. This option will disable flooding of unregistered multicast packets to all ports. The switch will send unregistered multicast packets only to ports connected to multicast routers.

9.2 Policy Reference

Neutron, like most OpenStack projects, uses a policy language to restrict permissions on REST API actions.

The following is an overview of all available policies in neutron.

9.2.1 neutron

context_is_admin

Default `role:admin`

Rule for cloud admin access

owner

Default `tenant_id:%(tenant_id)s`

Rule for resource owner access

admin_or_owner

Default `rule:context_is_admin or rule:owner`

Rule for admin or owner access

context_is_advsvc

Default `role:advsvc`

Rule for advsvc role access

admin_or_network_owner

Default `rule:context_is_admin or tenant_id:%(network:tenant_id)s`

Rule for admin or network owner access

admin_owner_or_network_owner

Default `rule:owner or rule:admin_or_network_owner`

Rule for resource owner, admin or network owner access

admin_only

Default `rule:context_is_admin`

Rule for admin-only access

regular_user

Default <empty string>

Rule for regular user access

shared

Default `field:networks:shared=True`

Rule of shared network

default

Default `rule:admin_or_owner`

Default access rule

admin_or_ext_parent_owner

Default `rule:context_is_admin or tenant_id:%(ext_parent:tenant_id)s`

Rule for common parent owner check

shared_address_scopes

Default `field:address_scopes:shared=True`

Definition of a shared address scope

create_address_scope

Default `rule:regular_user`

Operations

- **POST** `/address-scopes`

Create an address scope

create_address_scope:shared

Default `rule:admin_only`

Operations

- **POST** `/address-scopes`

Create a shared address scope

get_address_scope

Default `rule:admin_or_owner or rule:shared_address_scopes`

Operations

- **GET** `/address-scopes`
- **GET** `/address-scopes/{id}`

Get an address scope

update_address_scope

Default `rule:admin_or_owner`

Operations

- **PUT** `/address-scopes/{id}`

Update an address scope

`update_address_scope:shared`

Default `rule:admin_only`

Operations

- **PUT** `/address-scopes/{id}`

Update `shared` attribute of an address scope

`delete_address_scope`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/address-scopes/{id}`

Delete an address scope

`get_agent`

Default `rule:admin_only`

Operations

- **GET** `/agents`
- **GET** `/agents/{id}`

Get an agent

`update_agent`

Default `rule:admin_only`

Operations

- **PUT** `/agents/{id}`

Update an agent

`delete_agent`

Default `rule:admin_only`

Operations

- **DELETE** `/agents/{id}`

Delete an agent

`create_dhcp-network`

Default `rule:admin_only`

Operations

- **POST** `/agents/{agent_id}/dhcp-networks`

Add a network to a DHCP agent

get_dhcp-networks

Default rule:admin_only

Operations

- **GET** /agents/{agent_id}/dhcp-networks

List networks on a DHCP agent

delete_dhcp-network

Default rule:admin_only

Operations

- **DELETE** /agents/{agent_id}/dhcp-networks/{network_id}

Remove a network from a DHCP agent

create_l3-router

Default rule:admin_only

Operations

- **POST** /agents/{agent_id}/l3-routers

Add a router to an L3 agent

get_l3-routers

Default rule:admin_only

Operations

- **GET** /agents/{agent_id}/l3-routers

List routers on an L3 agent

delete_l3-router

Default rule:admin_only

Operations

- **DELETE** /agents/{agent_id}/l3-routers/{router_id}

Remove a router from an L3 agent

get_dhcp-agents

Default rule:admin_only

Operations

- **GET** /networks/{network_id}/dhcp-agents

List DHCP agents hosting a network

get_l3-agents

Default rule:admin_only

Operations

- **GET** /routers/{router_id}/l3-agents

List L3 agents hosting a router

get_auto_allocated_topology

Default `rule:admin_or_owner`

Operations

- **GET** `/auto-allocated-topology/{project_id}`

Get a projects auto-allocated topology

delete_auto_allocated_topology

Default `rule:admin_or_owner`

Operations

- **DELETE** `/auto-allocated-topology/{project_id}`

Delete a projects auto-allocated topology

get_availability_zone

Default `rule:regular_user`

Operations

- **GET** `/availability_zones`

List availability zones

create_flavor

Default `rule:admin_only`

Operations

- **POST** `/flavors`

Create a flavor

get_flavor

Default `rule:regular_user`

Operations

- **GET** `/flavors`
- **GET** `/flavors/{id}`

Get a flavor

update_flavor

Default `rule:admin_only`

Operations

- **PUT** `/flavors/{id}`

Update a flavor

delete_flavor

Default `rule:admin_only`

Operations

- **DELETE** `/flavors/{id}`

Delete a flavor

`create_service_profile`

Default `rule:admin_only`

Operations

- **POST** `/service_profiles`

Create a service profile

`get_service_profile`

Default `rule:admin_only`

Operations

- **GET** `/service_profiles`
- **GET** `/service_profiles/{id}`

Get a service profile

`update_service_profile`

Default `rule:admin_only`

Operations

- **PUT** `/service_profiles/{id}`

Update a service profile

`delete_service_profile`

Default `rule:admin_only`

Operations

- **DELETE** `/service_profiles/{id}`

Delete a service profile

`get_flavor_service_profile`

Default `rule:regular_user`

Get a flavor associated with a given service profiles. There is no corresponding GET operations in API currently. This rule is currently referred only in the DELETE of flavor_service_profile.

`create_flavor_service_profile`

Default `rule:admin_only`

Operations

- **POST** `/flavors/{flavor_id}/service_profiles`

Associate a flavor with a service profile

`delete_flavor_service_profile`

Default `rule:admin_only`

Operations

- **DELETE** `/flavors/{flavor_id}/service_profiles/{profile_id}`

Disassociate a flavor with a service profile

`create_floatingip`

Default `rule:regular_user`

Operations

- **POST** `/floatingips`

Create a floating IP

`create_floatingip:floating_ip_address`

Default `rule:admin_only`

Operations

- **POST** `/floatingips`

Create a floating IP with a specific IP address

`get_floatingip`

Default `rule:admin_or_owner`

Operations

- **GET** `/floatingips`
- **GET** `/floatingips/{id}`

Get a floating IP

`update_floatingip`

Default `rule:admin_or_owner`

Operations

- **PUT** `/floatingips/{id}`

Update a floating IP

`delete_floatingip`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/floatingips/{id}`

Delete a floating IP

`get_floatingip_pool`

Default `rule:regular_user`

Operations

- **GET** `/floatingip_pools`

Get floating IP pools

create_floatingip_port_forwarding

Default rule:admin_or_ext_parent_owner

Operations

- **POST** /floatingips/{floatingip_id}/port_forwardings

Create a floating IP port forwarding

get_floatingip_port_forwarding

Default rule:admin_or_ext_parent_owner

Operations

- **GET** /floatingips/{floatingip_id}/port_forwardings
- **GET** /floatingips/{floatingip_id}/port_forwardings/{port_forwarding_id}

Get a floating IP port forwarding

update_floatingip_port_forwarding

Default rule:admin_or_ext_parent_owner

Operations

- **PUT** /floatingips/{floatingip_id}/port_forwardings/{port_forwarding_id}

Update a floating IP port forwarding

delete_floatingip_port_forwarding

Default rule:admin_or_ext_parent_owner

Operations

- **DELETE** /floatingips/{floatingip_id}/port_forwardings/{port_forwarding_id}

Delete a floating IP port forwarding

create_router_conntrack_helper

Default rule:admin_or_ext_parent_owner

Operations

- **POST** /routers/{router_id}/conntrack_helpers

Create a router conntrack helper

get_router_conntrack_helper

Default rule:admin_or_ext_parent_owner

Operations

- **GET** /routers/{router_id}/conntrack_helpers
- **GET** /routers/{router_id}/conntrack_helpers/{conntrack_helper_id}

Get a router conntrack helper

update_router_conntrack_helper

Default rule:admin_or_ext_parent_owner

Operations

- **PUT** /routers/{router_id}/conntrack_helpers/{conntrack_helper_id}

Update a router conntrack helper

delete_router_conntrack_helper

Default rule:admin_or_ext_parent_owner

Operations

- **DELETE** /routers/{router_id}/conntrack_helpers/{conntrack_helper_id}

Delete a router conntrack helper

get_loggable_resource

Default rule:admin_only

Operations

- **GET** /log/loggable-resources

Get loggable resources

create_log

Default rule:admin_only

Operations

- **POST** /log/logs

Create a network log

get_log

Default rule:admin_only

Operations

- **GET** /log/logs
- **GET** /log/logs/{id}

Get a network log

update_log

Default rule:admin_only

Operations

- **PUT** /log/logs/{id}

Update a network log

delete_log

Default rule:admin_only

Operations

- **DELETE** `/log/logs/{id}`

Delete a network log

`create_metering_label`

Default `rule:admin_only`

Operations

- **POST** `/metering/metering-labels`

Create a metering label

`get_metering_label`

Default `rule:admin_only`

Operations

- **GET** `/metering/metering-labels`
- **GET** `/metering/metering-labels/{id}`

Get a metering label

`delete_metering_label`

Default `rule:admin_only`

Operations

- **DELETE** `/metering/metering-labels/{id}`

Delete a metering label

`create_metering_label_rule`

Default `rule:admin_only`

Operations

- **POST** `/metering/metering-label-rules`

Create a metering label rule

`get_metering_label_rule`

Default `rule:admin_only`

Operations

- **GET** `/metering/metering-label-rules`
- **GET** `/metering/metering-label-rules/{id}`

Get a metering label rule

`delete_metering_label_rule`

Default `rule:admin_only`

Operations

- **DELETE** `/metering/metering-label-rules/{id}`

Delete a metering label rule

external

Default `field:networks:router:external=True`

Definition of an external network

create_network

Default `rule:regular_user`

Operations

- **POST** `/networks`

Create a network

create_network:shared

Default `rule:admin_only`

Operations

- **POST** `/networks`

Create a shared network

create_network:router:external

Default `rule:admin_only`

Operations

- **POST** `/networks`

Create an external network

create_network:is_default

Default `rule:admin_only`

Operations

- **POST** `/networks`

Specify `is_default` attribute when creating a network

create_network:port_security_enabled

Default `rule:regular_user`

Operations

- **POST** `/networks`

Specify `port_security_enabled` attribute when creating a network

create_network:segments

Default `rule:admin_only`

Operations

- **POST** `/networks`

Specify `segments` attribute when creating a network

create_network:provider:network_type

Default `rule:admin_only`

Operations

- **POST** `/networks`

Specify `provider:network_type` when creating a network

create_network:provider:physical_network

Default `rule:admin_only`

Operations

- **POST** `/networks`

Specify `provider:physical_network` when creating a network

create_network:provider:segmentation_id

Default `rule:admin_only`

Operations

- **POST** `/networks`

Specify `provider:segmentation_id` when creating a network

get_network

Default `rule:admin_or_owner or rule:shared or rule:external or rule:context_is_advsvc`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get a network

get_network:router:external

Default `rule:regular_user`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get `router:external` attribute of a network

get_network:segments

Default `rule:admin_only`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get `segments` attribute of a network

get_network:provider:network_type

Default `rule:admin_only`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get `provider:network_type` attribute of a network

`get_network:provider:physical_network`

Default `rule:admin_only`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get `provider:physical_network` attribute of a network

`get_network:provider:segmentation_id`

Default `rule:admin_only`

Operations

- **GET** `/networks`
- **GET** `/networks/{id}`

Get `provider:segmentation_id` attribute of a network

`update_network`

Default `rule:admin_or_owner`

Operations

- **PUT** `/networks/{id}`

Update a network

`update_network:segments`

Default `rule:admin_only`

Operations

- **PUT** `/networks/{id}`

Update `segments` attribute of a network

`update_network:shared`

Default `rule:admin_only`

Operations

- **PUT** `/networks/{id}`

Update `shared` attribute of a network

`update_network:provider:network_type`

Default `rule:admin_only`

Operations

- **PUT** /networks/{id}

Update provider:network_type attribute of a network

update_network:provider:physical_network

Default rule:admin_only

Operations

- **PUT** /networks/{id}

Update provider:physical_network attribute of a network

update_network:provider:segmentation_id

Default rule:admin_only

Operations

- **PUT** /networks/{id}

Update provider:segmentation_id attribute of a network

update_network:router:external

Default rule:admin_only

Operations

- **PUT** /networks/{id}

Update router:external attribute of a network

update_network:is_default

Default rule:admin_only

Operations

- **PUT** /networks/{id}

Update is_default attribute of a network

update_network:port_security_enabled

Default rule:admin_or_owner

Operations

- **PUT** /networks/{id}

Update port_security_enabled attribute of a network

delete_network

Default rule:admin_or_owner

Operations

- **DELETE** /networks/{id}

Delete a network

get_network_ip_availability

Default `rule:admin_only`

Operations

- **GET** `/network-ip-availabilities`
- **GET** `/network-ip-availabilities/{network_id}`

Get network IP availability

create_network_segment_range

Default `rule:admin_only`

Operations

- **POST** `/network_segment_ranges`

Create a network segment range

get_network_segment_range

Default `rule:admin_only`

Operations

- **GET** `/network_segment_ranges`
- **GET** `/network_segment_ranges/{id}`

Get a network segment range

update_network_segment_range

Default `rule:admin_only`

Operations

- **PUT** `/network_segment_ranges/{id}`

Update a network segment range

delete_network_segment_range

Default `rule:admin_only`

Operations

- **DELETE** `/network_segment_ranges/{id}`

Delete a network segment range

network_device

Default `field:port:device_owner=~^network:`

Definition of port with network device_owner

admin_or_data_plane_int

Default `rule:context_is_admin or role:data_plane_integrator`

Rule for data plane integration

create_port

Default `rule:regular_user`

Operations

- **POST** `/ports`

Create a port

`create_port:device_owner`

Default `not rule:network_device or rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify `device_owner` attribute when creting a port

`create_port:mac_address`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify `mac_address` attribute when creating a port

`create_port:fixed_ips`

Default `rule:context_is_advsvc or rule:admin_or_network_owner or rule:shared`

Operations

- **POST** `/ports`

Specify `fixed_ips` information when creating a port

`create_port:fixed_ips:ip_address`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify IP address in `fixed_ips` when creating a port

`create_port:fixed_ips:subnet_id`

Default `rule:context_is_advsvc or rule:admin_or_network_owner or rule:shared`

Operations

- **POST** `/ports`

Specify subnet ID in `fixed_ips` when creating a port

`create_port:port_security_enabled`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify `port_security_enabled` attribute when creating a port

create_port:binding:host_id

Default `rule:admin_only`

Operations

- **POST** `/ports`

Specify `binding:host_id` attribute when creating a port

create_port:binding:profile

Default `rule:admin_only`

Operations

- **POST** `/ports`

Specify `binding:profile` attribute when creating a port

create_port:binding:vnic_type

Default `rule:regular_user`

Operations

- **POST** `/ports`

Specify `binding:vnic_type` attribute when creating a port

create_port:allowed_address_pairs

Default `rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify `allowed_address_pairs` attribute when creating a port

create_port:allowed_address_pairs:mac_address

Default `rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify ``mac_address` of `allowed_address_pairs`` attribute when creating a port

create_port:allowed_address_pairs:ip_address

Default `rule:admin_or_network_owner`

Operations

- **POST** `/ports`

Specify `ip_address` of `allowed_address_pairs` attribute when creating a port

get_port

Default `rule:context_is_advsvc or rule:admin_owner_or_network_owner`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get a port

`get_port:binding:vif_type`

Default `rule:admin_only`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get `binding:vif_type` attribute of a port

`get_port:binding:vif_details`

Default `rule:admin_only`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get `binding:vif_details` attribute of a port

`get_port:binding:host_id`

Default `rule:admin_only`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get `binding:host_id` attribute of a port

`get_port:binding:profile`

Default `rule:admin_only`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get `binding:profile` attribute of a port

`get_port:resource_request`

Default `rule:admin_only`

Operations

- **GET** `/ports`
- **GET** `/ports/{id}`

Get `resource_request` attribute of a port

`update_port`

Default `rule:admin_or_owner or rule:context_is_advsvc`

Operations

- **PUT** `/ports/{id}`

Update a port

`update_port:device_owner`

Default `not rule:network_device or rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Update `device_owner` attribute of a port

`update_port:mac_address`

Default `rule:admin_only or rule:context_is_advsvc`

Operations

- **PUT** `/ports/{id}`

Update `mac_address` attribute of a port

`update_port:fixed_ips`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Specify `fixed_ips` information when updating a port

`update_port:fixed_ips:ip_address`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Specify IP address in `fixed_ips` information when updating a port

`update_port:fixed_ips:subnet_id`

Default `rule:context_is_advsvc or rule:admin_or_network_owner or rule:shared`

Operations

- **PUT** `/ports/{id}`

Specify subnet ID in `fixed_ips` information when updating a port

`update_port:port_security_enabled`

Default `rule:context_is_advsvc or rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Update `port_security_enabled` attribute of a port

`update_port:binding:host_id`

Default `rule:admin_only`

Operations

- **PUT** `/ports/{id}`

Update `binding:host_id` attribute of a port

`update_port:binding:profile`

Default `rule:admin_only`

Operations

- **PUT** `/ports/{id}`

Update `binding:profile` attribute of a port

`update_port:binding:vnic_type`

Default `rule:admin_or_owner or rule:context_is_advsvc`

Operations

- **PUT** `/ports/{id}`

Update `binding:vnic_type` attribute of a port

`update_port:allowed_address_pairs`

Default `rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Update `allowed_address_pairs` attribute of a port

`update_port:allowed_address_pairs:mac_address`

Default `rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Update `mac_address` of `allowed_address_pairs` attribute of a port

`update_port:allowed_address_pairs:ip_address`

Default `rule:admin_or_network_owner`

Operations

- **PUT** `/ports/{id}`

Update `ip_address` of `allowed_address_pairs` attribute of a port

`update_port:data_plane_status`

Default `rule:admin_or_data_plane_int`

Operations

- **PUT** `/ports/{id}`

Update `data_plane_status` attribute of a port

delete_port

Default `rule:context_is_advsvc or rule:admin_owner_or_network_owner`

Operations

- **DELETE** `/ports/{id}`

Delete a port

get_policy

Default `rule:regular_user`

Operations

- **GET** `/qos/policies`
- **GET** `/qos/policies/{id}`

Get QoS policies

create_policy

Default `rule:admin_only`

Operations

- **POST** `/qos/policies`

Create a QoS policy

update_policy

Default `rule:admin_only`

Operations

- **PUT** `/qos/policies/{id}`

Update a QoS policy

delete_policy

Default `rule:admin_only`

Operations

- **DELETE** `/qos/policies/{id}`

Delete a QoS policy

get_rule_type

Default `rule:regular_user`

Operations

- **GET** `/qos/rule-types`
- **GET** `/qos/rule-types/{rule_type}`

Get available QoS rule types

get_policy_bandwidth_limit_rule

Default `rule:regular_user`

Operations

- **GET** `/qos/policies/{policy_id}/bandwidth_limit_rules`
- **GET** `/qos/policies/{policy_id}/bandwidth_limit_rules/{rule_id}`

Get a QoS bandwidth limit rule

create_policy_bandwidth_limit_rule

Default `rule:admin_only`

Operations

- **POST** `/qos/policies/{policy_id}/bandwidth_limit_rules`

Create a QoS bandwidth limit rule

update_policy_bandwidth_limit_rule

Default `rule:admin_only`

Operations

- **PUT** `/qos/policies/{policy_id}/bandwidth_limit_rules/{rule_id}`

Update a QoS bandwidth limit rule

delete_policy_bandwidth_limit_rule

Default `rule:admin_only`

Operations

- **DELETE** `/qos/policies/{policy_id}/bandwidth_limit_rules/{rule_id}`

Delete a QoS bandwidth limit rule

get_policy_dscp_marking_rule

Default `rule:regular_user`

Operations

- **GET** `/qos/policies/{policy_id}/dscp_marking_rules`
- **GET** `/qos/policies/{policy_id}/dscp_marking_rules/{rule_id}`

Get a QoS DSCP marking rule

create_policy_dscp_marking_rule

Default `rule:admin_only`

Operations

- **POST** `/qos/policies/{policy_id}/dscp_marking_rules`

Create a QoS DSCP marking rule

update_policy_dscp_marking_rule

Default `rule:admin_only`

Operations

- **PUT** `/qos/policies/{policy_id}/dscp_marking_rules/{rule_id}`

Update a QoS DSCP marking rule

delete_policy_dscp_marking_rule

Default `rule:admin_only`

Operations

- **DELETE** `/qos/policies/{policy_id}/dscp_marking_rules/{rule_id}`

Delete a QoS DSCP marking rule

get_policy_minimum_bandwidth_rule

Default `rule:regular_user`

Operations

- **GET** `/qos/policies/{policy_id}/minimum_bandwidth_rules`
- **GET** `/qos/policies/{policy_id}/minimum_bandwidth_rules/{rule_id}`

Get a QoS minimum bandwidth rule

create_policy_minimum_bandwidth_rule

Default `rule:admin_only`

Operations

- **POST** `/qos/policies/{policy_id}/minimum_bandwidth_rules`

Create a QoS minimum bandwidth rule

update_policy_minimum_bandwidth_rule

Default `rule:admin_only`

Operations

- **PUT** `/qos/policies/{policy_id}/minimum_bandwidth_rules/{rule_id}`

Update a QoS minimum bandwidth rule

delete_policy_minimum_bandwidth_rule

Default `rule:admin_only`

Operations

- **DELETE** `/qos/policies/{policy_id}/minimum_bandwidth_rules/{rule_id}`

Delete a QoS minimum bandwidth rule

get_alias_bandwidth_limit_rule

Default `rule:get_policy_bandwidth_limit_rule`

Operations

- **GET** `/qos/alias_bandwidth_limit_rules/{rule_id}/`

Get a QoS bandwidth limit rule through alias

update_alias_bandwidth_limit_rule

Default `rule:update_policy_bandwidth_limit_rule`

Operations

- **PUT** `/qos/alias_bandwidth_limit_rules/{rule_id}/`

Update a QoS bandwidth limit rule through alias

delete_alias_bandwidth_limit_rule

Default `rule:delete_policy_bandwidth_limit_rule`

Operations

- **DELETE** `/qos/alias_bandwidth_limit_rules/{rule_id}/`

Delete a QoS bandwidth limit rule through alias

get_alias_dscp_marking_rule

Default `rule:get_policy_dscp_marking_rule`

Operations

- **GET** `/qos/alias_dscp_marking_rules/{rule_id}/`

Get a QoS DSCP marking rule through alias

update_alias_dscp_marking_rule

Default `rule:update_policy_dscp_marking_rule`

Operations

- **PUT** `/qos/alias_dscp_marking_rules/{rule_id}/`

Update a QoS DSCP marking rule through alias

delete_alias_dscp_marking_rule

Default `rule:delete_policy_dscp_marking_rule`

Operations

- **DELETE** `/qos/alias_dscp_marking_rules/{rule_id}/`

Delete a QoS DSCP marking rule through alias

get_alias_minimum_bandwidth_rule

Default `rule:get_policy_minimum_bandwidth_rule`

Operations

- **GET** `/qos/alias_minimum_bandwidth_rules/{rule_id}/`

Get a QoS minimum bandwidth rule through alias

`update_alias_minimum_bandwidth_rule`

Default `rule:update_policy_minimum_bandwidth_rule`

Operations

- **PUT** `/qos/alias_minimum_bandwidth_rules/{rule_id}/`

Update a QoS minimum bandwidth rule through alias

`delete_alias_minimum_bandwidth_rule`

Default `rule:delete_policy_minimum_bandwidth_rule`

Operations

- **DELETE** `/qos/alias_minimum_bandwidth_rules/{rule_id}/`

Delete a QoS minimum bandwidth rule through alias

`get_quota`

Default `rule:admin_only`

Operations

- **GET** `/quota`
- **GET** `/quota/{id}`

Get a resource quota

`update_quota`

Default `rule:admin_only`

Operations

- **PUT** `/quota/{id}`

Update a resource quota

`delete_quota`

Default `rule:admin_only`

Operations

- **DELETE** `/quota/{id}`

Delete a resource quota

`restrict_wildcard`

Default `(not field:rbac_policy:target_tenant=*) or rule:admin_only`

Definition of a wildcard target_tenant

`create_rbac_policy`

Default `rule:regular_user`

Operations

- **POST** `/rbac-policies`

Create an RBAC policy

`create_rbac_policy:target_tenant`

Default `rule:restrict_wildcard`

Operations

- **POST** `/rbac-policies`

Specify `target_tenant` when creating an RBAC policy

`update_rbac_policy`

Default `rule:admin_or_owner`

Operations

- **PUT** `/rbac-policies/{id}`

Update an RBAC policy

`update_rbac_policy:target_tenant`

Default `rule:restrict_wildcard and rule:admin_or_owner`

Operations

- **PUT** `/rbac-policies/{id}`

Update `target_tenant` attribute of an RBAC policy

`get_rbac_policy`

Default `rule:admin_or_owner`

Operations

- **GET** `/rbac-policies`
- **GET** `/rbac-policies/{id}`

Get an RBAC policy

`delete_rbac_policy`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/rbac-policies/{id}`

Delete an RBAC policy

`create_router`

Default `rule:regular_user`

Operations

- **POST** `/routers`

Create a router

create_router:distributed

Default `rule:admin_only`

Operations

- **POST** `/routers`

Specify `distributed` attribute when creating a router

create_router:ha

Default `rule:admin_only`

Operations

- **POST** `/routers`

Specify `ha` attribute when creating a router

create_router:external_gateway_info

Default `rule:admin_or_owner`

Operations

- **POST** `/routers`

Specify `external_gateway_info` information when creating a router

create_router:external_gateway_info:network_id

Default `rule:admin_or_owner`

Operations

- **POST** `/routers`

Specify `network_id` in `external_gateway_info` information when creating a router

create_router:external_gateway_info:enable_snat

Default `rule:admin_only`

Operations

- **POST** `/routers`

Specify `enable_snat` in `external_gateway_info` information when creating a router

create_router:external_gateway_info:external_fixed_ips

Default `rule:admin_only`

Operations

- **POST** `/routers`

Specify `external_fixed_ips` in `external_gateway_info` information when creating a router

get_router

Default `rule:admin_or_owner`

Operations

- **GET** `/routers`
- **GET** `/routers/{id}`

Get a router

get_router:distributed

Default `rule:admin_only`

Operations

- **GET** `/routers`
- **GET** `/routers/{id}`

Get `distributed` attribute of a router

get_router:ha

Default `rule:admin_only`

Operations

- **GET** `/routers`
- **GET** `/routers/{id}`

Get `ha` attribute of a router

update_router

Default `rule:admin_or_owner`

Operations

- **PUT** `/routers/{id}`

Update a router

update_router:distributed

Default `rule:admin_only`

Operations

- **PUT** `/routers/{id}`

Update `distributed` attribute of a router

update_router:ha

Default `rule:admin_only`

Operations

- **PUT** `/routers/{id}`

Update `ha` attribute of a router

update_router:external_gateway_info

Default `rule:admin_or_owner`

Operations

- **PUT** `/routers/{id}`

Update `external_gateway_info` information of a router

update_router:external_gateway_info:network_id

Default `rule:admin_or_owner`

Operations

- **PUT** `/routers/{id}`

Update `network_id` attribute of `external_gateway_info` information of a router

update_router:external_gateway_info:enable_snat

Default `rule:admin_only`

Operations

- **PUT** `/routers/{id}`

Update `enable_snat` attribute of `external_gateway_info` information of a router

update_router:external_gateway_info:external_fixed_ips

Default `rule:admin_only`

Operations

- **PUT** `/routers/{id}`

Update `external_fixed_ips` attribute of `external_gateway_info` information of a router

delete_router

Default `rule:admin_or_owner`

Operations

- **DELETE** `/routers/{id}`

Delete a router

add_router_interface

Default `rule:admin_or_owner`

Operations

- **PUT** `/routers/{id}/add_router_interface`

Add an interface to a router

remove_router_interface

Default `rule:admin_or_owner`

Operations

- **PUT** `/routers/{id}/remove_router_interface`

Remove an interface from a router

admin_or_sg_owner

Default `rule:context_is_admin or tenant_id:%(security_group:tenant_id)s`

Rule for admin or security group owner access

admin_owner_or_sg_owner

Default `rule:owner or rule:admin_or_sg_owner`

Rule for resource owner, admin or security group owner access

create_security_group

Default `rule:admin_or_owner`

Operations

- **POST** `/security-groups`

Create a security group

get_security_group

Default `rule:regular_user`

Operations

- **GET** `/security-groups`
- **GET** `/security-groups/{id}`

Get a security group

update_security_group

Default `rule:admin_or_owner`

Operations

- **PUT** `/security-groups/{id}`

Update a security group

delete_security_group

Default `rule:admin_or_owner`

Operations

- **DELETE** `/security-groups/{id}`

Delete a security group

create_security_group_rule

Default `rule:admin_or_owner`

Operations

- **POST** `/security-group-rules`

Create a security group rule

get_security_group_rule

Default `rule:admin_owner_or_sg_owner`

Operations

- **GET** `/security-group-rules`
- **GET** `/security-group-rules/{id}`

Get a security group rule

delete_security_group_rule

Default `rule:admin_or_owner`

Operations

- **DELETE** `/security-group-rules/{id}`

Delete a security group rule

create_segment

Default `rule:admin_only`

Operations

- **POST** `/segments`

Create a segment

get_segment

Default `rule:admin_only`

Operations

- **GET** `/segments`
- **GET** `/segments/{id}`

Get a segment

update_segment

Default `rule:admin_only`

Operations

- **PUT** `/segments/{id}`

Update a segment

delete_segment

Default `rule:admin_only`

Operations

- **DELETE** `/segments/{id}`

Delete a segment

get_service_provider

Default `rule:regular_user`

Operations

- **GET** `/service-providers`

Get service providers

create_subnet

Default `rule:admin_or_network_owner`

Operations

- **POST** `/subnets`

Create a subnet

`create_subnet:segment_id`

Default `rule:admin_only`

Operations

- **POST** `/subnets`

Specify `segment_id` attribute when creating a subnet

`create_subnet:service_types`

Default `rule:admin_only`

Operations

- **POST** `/subnets`

Specify `service_types` attribute when creating a subnet

`get_subnet`

Default `rule:admin_or_owner or rule:shared`

Operations

- **GET** `/subnets`
- **GET** `/subnets/{id}`

Get a subnet

`get_subnet:segment_id`

Default `rule:admin_only`

Operations

- **GET** `/subnets`
- **GET** `/subnets/{id}`

Get `segment_id` attribute of a subnet

`update_subnet`

Default `rule:admin_or_network_owner`

Operations

- **PUT** `/subnets/{id}`

Update a subnet

`update_subnet:segment_id`

Default `rule:admin_only`

Operations

- **PUT** `/subnets/{id}`

Update `segment_id` attribute of a subnet

`update_subnet:service_types`

Default `rule:admin_only`

Operations

- **PUT** `/subnets/{id}`

Update `service_types` attribute of a subnet

`delete_subnet`

Default `rule:admin_or_network_owner`

Operations

- **DELETE** `/subnets/{id}`

Delete a subnet

`shared_subnetpools`

Default `field:subnetpools:shared=True`

Definition of a shared subnetpool

`create_subnetpool`

Default `rule:regular_user`

Operations

- **POST** `/subnetpools`

Create a subnetpool

`create_subnetpool:shared`

Default `rule:admin_only`

Operations

- **POST** `/subnetpools`

Create a shared subnetpool

`create_subnetpool:is_default`

Default `rule:admin_only`

Operations

- **POST** `/subnetpools`

Specify `is_default` attribute when creating a subnetpool

`get_subnetpool`

Default `rule:admin_or_owner or rule:shared_subnetpools`

Operations

- **GET** `/subnetpools`
- **GET** `/subnetpools/{id}`

Get a subnetpool

update_subnetpool

Default rule:admin_or_owner

Operations

- **PUT** /subnetpools/{id}

Update a subnetpool

update_subnetpool:is_default

Default rule:admin_only

Operations

- **PUT** /subnetpools/{id}

Update is_default attribute of a subnetpool

delete_subnetpool

Default rule:admin_or_owner

Operations

- **DELETE** /subnetpools/{id}

Delete a subnetpool

onboard_network_subnets

Default rule:admin_or_owner

Operations

- **Put** /subnetpools/{id}/onboard_network_subnets

Onboard existing subnet into a subnetpool

add_prefixes

Default rule:admin_or_owner

Operations

- **Put** /subnetpools/{id}/add_prefixes

Add prefixes to a subnetpool

remove_prefixes

Default rule:admin_or_owner

Operations

- **Put** /subnetpools/{id}/remove_prefixes

Remove unallocated prefixes from a subnetpool

create_trunk

Default rule:regular_user

Operations

- **POST** `/trunks`

Create a trunk

`get_trunk`

Default `rule:admin_or_owner`

Operations

- **GET** `/trunks`
- **GET** `/trunks/{id}`

Get a trunk

`update_trunk`

Default `rule:admin_or_owner`

Operations

- **PUT** `/trunks/{id}`

Update a trunk

`delete_trunk`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/trunks/{id}`

Delete a trunk

`get_subports`

Default `rule:regular_user`

Operations

- **GET** `/trunks/{id}/get_subports`

List subports attached to a trunk

`add_subports`

Default `rule:admin_or_owner`

Operations

- **PUT** `/trunks/{id}/add_subports`

Add subports to a trunk

`remove_subports`

Default `rule:admin_or_owner`

Operations

- **PUT** `/trunks/{id}/remove_subports`

Delete subports from a trunk

CHAPTER TEN

COMMAND-LINE INTERFACE REFERENCE

10.1 neutron-debug

The **neutron-debug** client is an extension to the **neutron** command-line interface (CLI) for the OpenStack neutron-debug tool.

This chapter documents **neutron-debug** version `2.3.0`.

For help on a specific **neutron-debug** command, enter:

```
$ neutron-debug help COMMAND
```

10.1.1 neutron-debug usage

```
usage: neutron-debug [--version] [-v] [-q] [-h] [-r NUM]
                     [--os-service-type <os-service-type>]
                     [--os-endpoint-type <os-endpoint-type>]
                     [--service-type <service-type>]
                     [--endpoint-type <endpoint-type>]
                     [--os-auth-strategy <auth-strategy>] [--os-cloud
↪<cloud>]
                     [--os-auth-url <auth-url>]
                     [--os-tenant-name <auth-tenant-name> | --os-project-
↪name <auth-project-name>]
                     [--os-tenant-id <auth-tenant-id> | --os-project-id
↪<auth-project-id>]
                     [--os-username <auth-username>]
                     [--os-user-id <auth-user-id>]
                     [--os-user-domain-id <auth-user-domain-id>]
                     [--os-user-domain-name <auth-user-domain-name>]
                     [--os-project-domain-id <auth-project-domain-id>]
                     [--os-project-domain-name <auth-project-domain-name>]
                     [--os-cert <certificate>] [--os-cacert <ca-
↪certificate>]
                     [--os-key <key>] [--os-password <auth-password>]
                     [--os-region-name <auth-region-name>]
                     [--os-token <token>] [--http-timeout <seconds>]
                     [--os-url <url>] [--insecure] [--config-file CONFIG_
↪FILE]
                     <subcommand> ...
```

Subcommands

probe-create Create probe port - create port and interface within a network namespace.

probe-list List all probes.

probe-clear Clear all probes.

probe-delete Delete probe - delete port then delete the namespace.

probe-exec Execute commands in the namespace of the probe.

ping-all `ping-all` is an all-in-one command to ping all fixed IPs in a specified network.

10.1.2 neutron-debug optional arguments

--version Show programs version number and exit

-v, --verbose, --debug Increase verbosity of output and show tracebacks on errors. You can repeat this option.

-q, --quiet Suppress output except warnings and errors.

-h, --help Show this help message and exit

-r NUM, --retries NUM How many times the request to the Neutron server should be retried if it fails.

--os-service-type <os-service-type> Defaults to env[OS_NETWORK_SERVICE_TYPE] or network.

--os-endpoint-type <os-endpoint-type> Defaults to `env[OS_ENDPOINT_TYPE]` or public.

--service-type <service-type> DEPRECATED! Use os-service-type.

--endpoint-type <endpoint-type> DEPRECATED! Use os-endpoint-type.

--os-auth-strategy <auth-strategy> DEPRECATED! Only keystone is supported.

os-cloud <cloud> Defaults to env[OS_CLOUD].

--os-auth-url <auth-url> Authentication URL, defaults to env[OS_AUTH_URL].

--os-tenant-name <auth-tenant-name> Authentication tenant name, defaults to env[OS_TENANT_NAME].

--os-project-name <auth-project-name> Another way to specify tenant name. This option is mutually exclusive with os-tenant-name. Defaults to env[OS_PROJECT_NAME].

--os-tenant-id <auth-tenant-id> Authentication tenant ID, defaults to env[OS_TENANT_ID].

--os-project-id <auth-project-id> Another way to specify tenant ID. This option is mutually exclusive with os-tenant-id. Defaults to env[OS_PROJECT_ID].

--os-username <auth-username> Authentication username, defaults to env[OS_USERNAME].

--os-user-id <auth-user-id> Authentication user ID (Env: OS_USER_ID)

--os-user-domain-id <auth-user-domain-id> OpenStack user domain ID. Defaults to env[OS_USER_DOMAIN_ID].

--os-user-domain-name <auth-user-domain-name> OpenStack user domain name. Defaults to env[OS_USER_DOMAIN_NAME].

--os-project-domain-id <auth-project-domain-id> Defaults to env[OS_PROJECT_DOMAIN_ID].

--os-project-domain-name <auth-project-domain-name> Defaults to env[OS_PROJECT_DOMAIN_NAME].

--os-cert <certificate> Path of certificate file to use in SSL connection. This file can optionally be prepended with the private key. Defaults to env[OS_CERT].

--os-cacert <ca-certificate> Specify a CA bundle file to use in verifying a TLS (https) server certificate. Defaults to env[OS_CACERT].

--os-key <key> Path of client key to use in SSL connection. This option is not necessary if your key is prepended to your certificate file. Defaults to env[OS_KEY].

--os-password <auth-password> Authentication password, defaults to env[OS_PASSWORD].

--os-region-name <auth-region-name> Authentication region name, defaults to env[OS_REGION_NAME].

--os-token <token> Authentication token, defaults to env[OS_TOKEN].

--http-timeout <seconds> Timeout in seconds to wait for an HTTP response. Defaults to env[OS_NETWORK_TIMEOUT] or None if not specified.

--os-url <url> Defaults to env[OS_URL]

--insecure Explicitly allow neutronclient to perform insecure SSL (https) requests. The servers certificate will not be verified against any certificate authorities. This option should be used with caution.

--config-file CONFIG_FILE Config file for interface driver (You may also use l3_agent.ini)

10.1.3 neutron-debug probe-create command

```
usage: neutron-debug probe-create NET
```

Create probe port - create port and interface, then place it into the created network namespace.

Positional arguments

NET ID ID of the network in which the probe will be created.

10.1.4 neutron-debug probe-list command

```
usage: neutron-debug probe-list
```

List probes.

10.1.5 neutron-debug probe-clear command

```
usage: neutron-debug probe-clear
```

Clear all probes.

10.1.6 neutron-debug probe-delete command

```
usage: neutron-debug probe-delete <port-id>
```

Remove a probe.

Positional arguments

`<port-id>` ID of the probe to delete.

10.1.7 neutron-debug probe-exec command

```
usage: neutron-debug probe-exec <port-id> <command>
```

Execute commands in the namespace of the probe

10.1.8 neutron-debug ping-all command

```
usage: neutron-debug ping-all <port-id> --timeout <number>
```

All-in-one command to ping all fixed IPs in a specified network. A probe creation is not needed for this command. A new probe is created automatically. It will, however, need to be deleted manually when it is no longer needed. When there are multiple networks, the newly created probe will be attached to a random network and thus the ping will take place from within that random network.

Positional arguments

`<port-id>` ID of the port to use.

Optional arguments

--timeout <timeout in seconds> Optional ping timeout.

10.1.9 neutron-debug example

```
usage: neutron-debug create-probe <NET_ID>
```

Create a probe namespace within the network identified by `NET_ID`. The namespace will have the name of qprobe-<UUID of the probe port>

Note: For the following examples to function, the security group rules may need to be modified to allow the SSH (TCP port 22) or ping (ICMP) traffic into network.

```
usage: neutron-debug probe-exec <probe ID> "ssh <IP of instance>"
```

SSH to an instance within the network.

```
usage: neutron-debug ping-all <network ID>
```

Ping all instances on this network to verify they are responding.

```
usage: neutron-debug probe-exec <probe_ID> dhcping <VM_MAC address> -s <IP␣
↪of DHCP server>
```

Ping the DHCP server for this network using dhcping to verify it is working.

10.2 neutron-sanity-check

The **neutron-sanity-check** client is a tool that checks various sanity about the Networking service.

This chapter documents **neutron-sanity-check** version `10.0.0`.

10.2.1 neutron-sanity-check usage

```
usage: neutron-sanity-check [-h] [--arp_header_match] [--arp_responder]
                            [--bridge_firewalling] [--config-dir DIR]
                            [--config-file PATH] [--debug] [--dhcp_
↪release6]
                            [--dibbler_version] [--dnsmasq_version]
                            [--ebtables_installed] [--icmpv6_header_match]
                            [--ip6tables_installed] [--ip_nonlocal_bind]
                            [--iproute2_vxlan] [--ipset_installed]
                            [--keepalived_ipv6_support]
                            [--log-config-append PATH]
                            [--log-date-format DATE_FORMAT]
                            [--log-dir LOG_DIR] [--log-file PATH]
                            [--noarp_header_match] [--noarp_responder]
```

(continues on next page)

(continued from previous page)

```
                                    [--nobridge_firewalling] [--nodebug]
                                    [--nodhcp_release6] [--nodibbler_version]
                                    [--nodnsmasq_version] [--noebtables_installed]
                                    [--noicmpv6_header_match]
                                    [--noip6tables_installed] [--noip_nonlocal_
↪bind]
                                    [--noiproute2_vxlan] [--noipset_installed]
                                    [--nokeepalived_ipv6_support] [--nonova_notify]
                                    [--noovs_conntrack] [--noovs_geneve]
                                    [--noovs_patch] [--noovs_vxlan] [--noovsdb_
↪native]
                                    [--noread_netns] [--nouse-syslog] [--nova_
↪notify]
                                    [--noverbose] [--nowatch-log-file]
                                    [--ovs_conntrack] [--ovs_geneve] [--ovs_patch]
                                    [--ovs_vxlan] [--ovsdb_native] [--read_netns]
                                    [--state_path STATE_PATH]
                                    [--syslog-log-facility SYSLOG_LOG_FACILITY]
                                    [--use-syslog] [--verbose] [--version]
                                    [--watch-log-file]
```

10.2.2 neutron-sanity-check optional arguments

-h, --help show this help message and exit

--arp_header_match Check for ARP header match support

--arp_responder Check for ARP responder support

--bridge_firewalling Check bridge firewalling

--ip_nonlocal_bind Check ip_nonlocal_bind kernel option works with network namespaces.

--config-dir DIR Path to a config directory to pull `*.conf` files from. This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s) specified via previous config-file, arguments hence over-ridden options in the directory take precedence.

--config-file PATH Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence. Dafaults to `None`.

--debug, -d Print debugging output (set logging level to `DEBUG` instead of default `INFO` level).

--dhcp_release6 Check dhcp_release6 installation

--dibbler_version Check minimal dibbler version

--dnsmasq_version Check minimal dnsmasq version

--ebtables_installed Check ebtables installation

--icmpv6_header_match Check for ICMPv6 header match support

--ip6tables_installed Check ip6tables installation

--iproute2_vxlan Check for iproute2 vxlan support

--ipset_installed Check ipset installation

--keepalived_ipv6_support Check keepalived IPv6 support

--log-config-append PATH, --log_config PATH The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, `logging_context_format_string`).

--log-date-format DATE_FORMAT Format string for %(asctime)s in log records. Default: None. This option is ignored if `log_config_append` is set.

--log-dir LOG_DIR, --logdir LOG_DIR (Optional) The base directory used for relative `log-file` paths. This option is ignored if `log_config_append` is set.

--log-file PATH, --logfile PATH (Optional) Name of log file to output to. If no default is set, logging will go to stderr as defined by `use_stderr`. This option is ignored if `log_config_append` is set.

--noarp_header_match The inverse of arp_header_match

--noarp_responder The inverse of arp_responder

--nobridge_firewalling The inverse of bridge_firewalling

--nodebug The inverse of debug

--nodhcp_release6 The inverse of dhcp_release6

--nodibbler_version The inverse of dibbler_version

--nodnsmasq_version The inverse of dnsmasq_version

--noebtables_installed The inverse of ebtables_installed

--noicmpv6_header_match The inverse of icmpv6_header_match

--noip6tables_installed The inverse of ip6tables_installed

--noip_nonlocal_bind The inverse of ip_nonlocal_bind

--noiproute2_vxlan The inverse of iproute2_vxlan

--noipset_installed The inverse of ipset_installed

--nokeepalived_ipv6_support The inverse of keepalived_ipv6_support

--nonova_notify The inverse of nova_notify

--noovs_conntrack The inverse of ovs_conntrack

--noovs_geneve The inverse of ovs_geneve

--noovs_patch The inverse of ovs_patch

--noovs_vxlan The inverse of ovs_vxlan

--noovsdb_native The inverse of ovsdb_native

--noread_netns The inverse of read_netns

--nouse-syslog The inverse of use-syslog

--nova_notify Check for nova notification support

--noverbose The inverse of verbose

--nowatch-log-file The inverse of watch-log-file

--ovs_geneve Check for OVS Geneve support

--ovs_patch Check for patch port support

--ovs_vxlan Check for OVS vxlan support

--ovsdb_native Check ovsdb native interface support

--read_netns Check netns permission settings

--state_path STATE_PATH Where to store Neutron state files. This directory must be writable by the agent.

--syslog-log-facility SYSLOG_LOG_FACILITY Syslog facility to receive log lines. This option is ignored if `log_config_append` is set.

--use-syslog Use syslog for logging. Existing syslog format is **DEPRECATED** and will be changed later to honor RFC5424. This option is ignored if `log_config_append` is set.

--verbose, -v If set to `false`, the logging level will be set to `WARNING` instead of the default `INFO` level.

--version show programs version number and exit

--watch-log-file Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if `log_file` option is specified and Linux platform is used. This option is ignored if `log_config_append` is set.

10.3 neutron-status

The **neutron-status** provides routines for checking the status of Neutron deployment.

10.3.1 neutron-status usage

```
usage: neutron-status [-h] [--config-dir DIR] [--config-file PATH]
                      <category> <command>
```

Categories are:

- `upgrade`

Detailed descriptions are below.

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
neutron-status upgrade
```

These sections describe the available categories and arguments for **neutron-status**.

Command details

`neutron-status upgrade check` Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to databases and services.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

21.0.0 (Ussuri)

- A Check was added for NIC Switch agents to ensure nodes are running with kernel 3.13 or newer. This check serves as a notification for operators to ensure this requirement is fullfiled on relevant nodes.

CHAPTER

ELEVEN

OVN DRIVER

11.1 Migration Strategy

This document details an in-place migration strategy from ML2/OVS to ML2/OVN in either ovs-firewall or ovs-hybrid mode for a TripleO OpenStack deployment.

For non TripleO deployments, please refer to the file `migration/README.rst` and the ansible playbook `migration/migrate-to-ovn.yml`.

11.1.1 Overview

The migration process is orchestrated through the shell script ovn_migration.sh, which is provided with the OVN driver.

The administrator uses ovn_migration.sh to perform readiness steps and migration from the undercloud node. The readiness steps, such as host inventory production, DHCP and MTU adjustments, prepare the environment for the procedure.

Subsequent steps start the migration via Ansible.

Plan for a 24-hour wait after the setup-mtu-t1 step to allow VMs to catch up with the new MTU size. The default neutron ML2/OVS configuration has a dhcp_lease_duration of 86400 seconds (24h).

Also, if there are instances using static IP assignment, the administrator should be ready to update the MTU of those instances to the new value of 8 bytes less than the ML2/OVS (VXLAN) MTU value. For example, the typical 1500 MTU network value that makes VXLAN tenant networks use 1450 bytes of MTU will need to change to 1442 under Geneve. Or under the same overlay network, a GRE encapsulated tenant network would use a 1458 MTU, but again a 1442 MTU for Geneve.

If there are instances which use DHCP but dont support lease update during the T1 period the administrator will need to reboot them to ensure that MTU is updated inside those instances.

11.1.2 Steps for migration

Perform the following steps in the overcloud/undercloud

1. Ensure that you have updated to the latest openstack/neutron version.

Perform the following steps in the undercloud

1. Install python-networking-ovn-migration-tool.

```
# yum install python-networking-ovn-migration-tool
```

2. Create a working directory on the undercloud, and copy the ansible playbooks

```
$ mkdir ~/ovn_migration
$ cd ~/ovn_migration
$ cp -rfp /usr/share/ansible/networking-ovn-migration/playbooks .
```

3. Create `~/overcloud-deploy-ovn.sh` script in your `$HOME`. This script must source your stackrc file, and then execute an `openstack overcloud deploy` with your original deployment parameters, plus the following environment files, added to the end of the command in the following order:

   When your network topology is DVR and your compute nodes have connectivity to the external network:

```
-e /usr/share/openstack-tripleo-heat-templates/environments/services/
↪neutron-ovn-dvr-ha.yaml \
-e $HOME/ovn-extras.yaml
```

   When your compute nodes dont have external connectivity and you dont use DVR:

```
-e /usr/share/openstack-tripleo-heat-templates/environments/services/
↪neutron-ovn-ha.yaml \
-e $HOME/ovn-extras.yaml
```

   Make sure that all users have execution privileges on the script, because it will be called by ovn_migration.sh/ansible during the migration process.

```
$ chmod a+x ~/overcloud-deploy-ovn.sh
```

4. To configure the parameters of your migration you can set the environment variables that will be used by `ovn_migration.sh`. You can skip setting any values matching the defaults.

   - STACKRC_FILE - must point to your stackrc file in your undercloud. Default: ~/stackrc
   - OVERCLOUDRC_FILE - must point to your overcloudrc file in your undercloud. Default: ~/overcloudrc
   - OVERCLOUD_OVN_DEPLOY_SCRIPT - must point to the script described in step 1. Default: ~/overcloud-deploy-ovn.sh
   - UNDERCLOUD_NODE_USER - user used on the undercloud nodes Default: heat-admin
   - STACK_NAME - Name or ID of the heat stack Default: overcloud If the stack that is migrated differs from the default, please set this environment variable to the stack name or ID.
   - PUBLIC_NETWORK_NAME - Name of your public network. Default: public. To support migration validation, this network must have available floating IPs, and those floating IPs must be pingable from the undercloud. If thats not possible please configure VALIDATE_MIGRATION to False.

- IMAGE_NAME - Name/ID of the glance image to us for booting a test server. Default:cirros. If the image does not exist it will automatically download and use cirros during the pre-validation / post-validation process.
- VALIDATE_MIGRATION - Create migration resources to validate the migration. The migration script, before starting the migration, boot a server and validates that the server is reachable after the migration. Default: True.
- SERVER_USER_NAME - User name to use for logging into the migration instances. Default: cirros.
- DHCP_RENEWAL_TIME - DHCP renewal time in seconds to configure in DHCP agent configuration file. This renewal time is used only temporarily during migration to ensure a synchronized MTU switch across the networks. Default: 30

Warning: Please note that VALIDATE_MIGRATION requires enough quota (2 available floating ips, 2 networks, 2 subnets, 2 instances, and 2 routers as admin).

For example:

```
$ export PUBLIC_NETWORK_NAME=my-public-network
$ ovn_migration.sh .........
```

5. Run `ovn_migration.sh generate-inventory` to generate the inventory file - `hosts_for_migration` and `ansible.cfg`. Please review `hosts_for_migration` for correctness.

```
$ ovn_migration.sh generate-inventory
```

At this step the script will inspect the TripleO ansible inventory and generate an inventory of hosts, specifically tagged to work with the migration playbooks.

6. Run `ovn_migration.sh setup-mtu-t1`

```
$ ovn_migration.sh setup-mtu-t1
```

This lowers the T1 parameter of the internal neutron DHCP servers configuring the `dhcp_renewal_time` in /var/lib/config-data/puppet-generated/neutron/etc/neutron/dhcp_agent.ini in all the nodes where DHCP agent is running.

We lower the T1 parameter to make sure that the instances start refreshing the DHCP lease quicker (every 30 seconds by default) during the migration proccess. The reason why we force this is to make sure that the MTU update happens quickly across the network during step 8, this is very important because during those 30 seconds there will be connectivity issues with bigger packets (MTU missmatchess across the network), this is also why step 7 is very important, even though we reduce T1, the previous T1 value the instances leased from the DHCP server will be much higher (24h by default) and we need to wait those 24h to make sure they have updated T1. After migration the DHCP T1 parameter returns to normal values.

7. If you are using VXLAN or GRE tenant networking, `wait at least 24 hours` before continuing. This will allow VMs to catch up with the new MTU size of the next step.

Warning: If you are using VXLAN or GRE networks, this 24-hour wait step is critical. If you are using VLAN tenant networks you can proceed to the next step without delay.

Warning: If you have any instance with static IP assignment on VXLAN or GRE tenant networks, you must manually modify the configuration of those instances. If your instances dont honor the T1 parameter of DHCP they will need to be rebooted. to configure the new geneve MTU, which is the current VXLAN MTU minus 8 bytes. For instance, if the VXLAN-based MTU was 1450, change it to 1442.

Note: 24 hours is the time based on default configuration. It actually depends on /var/lib/config-data/puppet-generated/neutron/etc/neutron/dhcp_agent.ini dhcp_renewal_time and /var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf dhcp_lease_duration parameters. (defaults to 86400 seconds)

Note: Please note that migrating a deployment which uses VLAN for tenant/project networks is not recommended at this time because of a bug in core ovn, full support is being worked out here: https://mail.openvswitch.org/pipermail/ovs-dev/2018-May/347594.html

One way to verify that the T1 parameter has propagated to existing VMs is to connect to one of the compute nodes, and run `tcpdump` over one of the VM taps attached to a tenant network. If T1 propegation was a success, you should see that requests happen on an interval of approximately 30 seconds.

```
[heat-admin@overcloud-novacompute-0 ~]$ sudo tcpdump -i tap52e872c2-
↪e6 port 67 or port 68 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol␣
↪decode
listening on tap52e872c2-e6, link-type EN10MB (Ethernet), capture␣
↪size 262144 bytes
13:17:28.954675 IP 192.168.99.5.bootpc > 192.168.99.3.bootps: BOOTP/
↪DHCP, Request from fa:16:3e:6b:41:3d, length 300
13:17:28.961321 IP 192.168.99.3.bootps > 192.168.99.5.bootpc: BOOTP/
↪DHCP, Reply, length 355
13:17:56.241156 IP 192.168.99.5.bootpc > 192.168.99.3.bootps: BOOTP/
↪DHCP, Request from fa:16:3e:6b:41:3d, length 300
13:17:56.249899 IP 192.168.99.3.bootps > 192.168.99.5.bootpc: BOOTP/
↪DHCP, Reply, length 355
```

Note: This verification is not possible with cirros VMs. The cirros udhcpc implementation does not obey DHCP option 58 (T1). Please try this verification on a port that belongs to a full linux VM. We recommend you to check all the different types of workloads your system runs (Windows, different flavors of linux, etc..).

8. Run `ovn_migration.sh reduce-mtu`.

   This lowers the MTU of the pre migration VXLAN and GRE networks. The tool will ignore non-VXLAN/GRE networks, so if you use VLAN for tenant networks it will be fine if you find this step not doing anything.

```
$ ovn_migration.sh reduce-mtu
```

This step will go network by network reducing the MTU, and tagging with `adapted_mtu` the networks which have been already handled.

Every time a network is updated all the existing L3/DHCP agents connected to such network will update their internal leg MTU, instances will start fetching the new MTU as the DHCP T1 timer expires. As explained before, instances not obeying the DHCP T1 parameter will need to be restarted, and instances with static IP assignment will need to be manually updated.

9. Make TripleO `prepare the new container images` for OVN.

   If your deployment didnt have a containers-prepare-parameter.yaml, you can create one with:

   ```
   $ test -f $HOME/containers-prepare-parameter.yaml || \
         openstack tripleo container image prepare default \
             --output-env-file $HOME/containers-prepare-parameter.yaml
   ```

   If you had to create the file, please make sure its included at the end of your $HOME/overcloud-deploy-ovn.sh and $HOME/overcloud-deploy.sh

   Change the neutron_driver in the containers-prepare-parameter.yaml file to ovn:

   ```
   $ sed -i -E 's/neutron_driver:([ ]\w+)/neutron_driver: ovn/' $HOME/
   ↪containers-prepare-parameter.yaml
   ```

   You can verify with:

   ```
   $ grep neutron_driver $HOME/containers-prepare-parameter.yaml
   neutron_driver: ovn
   ```

   Then update the images:

   ```
   $ openstack tripleo container image prepare \
      --environment-file $HOME/containers-prepare-parameter.yaml
   ```

   Note: Its important to provide the full path to your containers-prepare-parameter.yaml otherwise the command will finish very quickly and wont work (current version doesnt seem to output any error).

   During this step TripleO will build a list of containers, pull them from the remote registry and push them to your deployment local registry.

10. Run `ovn_migration.sh start-migration` to kick start the migration process.

    ```
    $ ovn_migration.sh start-migration
    ```

    During this step, this is what will happen:

    - Create pre-migration resources (network and VM) to validate existing deployment and final migration.

- Update the overcloud stack to deploy OVN alongside reference implementation services using a temporary bridge br-migration instead of br-int.
- Start the migration process:
  1. generate the OVN north db by running neutron-ovn-db-sync util
  2. clone the existing resources from br-int to br-migration, so OVN can find the same resources UUIDS over br-migration
  3. re-assign ovn-controller to br-int instead of br-migration
  4. cleanup network namespaces (fip, snat, qrouter, qdhcp),
  5. remove any unnecessary patch ports on br-int
  6. remove br-tun and br-migration ovs bridges
  7. delete qr-*, ha-* and qg-* ports from br-int (via neutron netns cleanup)
- Delete neutron agents and neutron HA internal networks from the database via API.
- Validate connectivity on pre-migration resources.
- Delete pre-migration resources.
- Create post-migration resources.
- Validate connectivity on post-migration resources.
- Cleanup post-migration resources.
- Re-run deployment tool to update OVN on br-int, this step ensures that the TripleO database is updated with the final integration bridge.
- Run an extra validation round to ensure the final state of the system is fully operational.

Migration is complete !!!

11.2 Gaps from ML2/OVS

This is a list of some of the currently known gaps between ML2/OVS and OVN. It is not a complete list, but is enough to be used as a starting point for implementors working on closing these gaps. A TODO list for OVN is located at[1].

- Security Groups logging API

  Currently ML2/OVS, with the OpenvSwitch firewall, supports a log file where security groups events are logged to be consumed by a security entity. This allows users to have a way to check if an instance is trying to execute restricted operations, or access restricted ports in remote servers.

  This is a relatively new extension, support would need to be added to OVN.

- QoS DSCP support

  Currently ML2/OVS supports QoS DSCP tagging and egress bandwidth limiting. Those are basic QoS features that while integrated in the OVS/OVN C core are not integrated (or fully tested) in the neutron OVN mechanism driver.

[1] https://github.com/ovn-org/ovn/blob/master/TODO.rst

- QoS for Layer 3 IPs

  Currently the Neutron L3-agent supports floating IP and gateway IP bandwidth limiting based on Linux TC. Networking-ovn L3 had a prototype implementation[2] based on the meter of openvswitch[3] utility that has been abandoned. This is supported in user space datapath only, or kernel versions 4.15+[4].

- QoS Minimum Bandwidth support

  Currently ML2/OVS supports QoS Minimum Bandwidth limiting, but it is not supported in OVN.

- BGP support

  Currently ML2/OVS supports making a tenant subnet routable via BGP, and can announce host routes for both floating and fixed IP addresses.

- Baremetal provisioning with iPXE

  The core OVN DHCP server implementation does not have support for sending different boot options based on the `gpxe` DHCP Option (no. 175). Also, Ironic uses dnsmasq syntax when configuring the DHCP options for Neutron[5] which is not understood by the OVN driver.

- Availability Zones

  Availability zones are used to make network resources highly available by grouping nodes in separate zones which resources will be scheduled to. Neutron supports two types of availability zones: Network (DHCP agent) and router (L3 agent). The OVN team needs to assess each case to see how they would fit in the OVN model. For example, in the router availability zone case, the OVN driver should schedule the router ports on a Chassis (a node in OVN terms) where the availability zones match with the router availability zones[6].

- Routed provider networks

  Routed provider networks allow for a single provider network to represent multiple L2 domains (segments). The OVN driver does not understand this feature yet and will need to account for multiple physical networks associated with a single OVN Logical Switch (a network in Neutron terms)[7].

- QoS minimum bandwidth allocation in Placement API

  ML2/OVN integration with the Nova placement API to provide guaranteed minimum bandwidth for ports[8].

- IPv6 Prefix Delegation

  Currently ML2/OVN doesnt implement IPv6 prefix delegation. OVN logical routers have this capability implemented in[9] and we have an open RFE to fill this gap[10].

[2] https://review.opendev.org/#/c/539826/

[3] https://github.com/openvswitch/ovs/commit/66d89287269ca7e2f7593af0920e910d7f9bcc38

[4] https://github.com/torvalds/linux/blob/master/net/openvswitch/meter.h

[5] https://github.com/openstack/ironic/blob/123cb22c731f93d0c608d791b41e05884fe18c04/ironic/common/pxe_utils.py#L447-L462>

[6] https://docs.openstack.org/neutron/latest/admin/config-az.html

[7] https://bugs.launchpad.net/neutron/+bug/1865889

[8] https://specs.openstack.org/openstack/neutron-specs/specs/rocky/minimum-bandwidth-allocation-placement-api.html

[9] https://patchwork.ozlabs.org/project/openvswitch/patch/6aec0fb280f610a2083fbb6c61e251b1d237b21f.1576840560.git.lorenzo.bianconi@redhat.com/

[10] https://bugs.launchpad.net/neutron/+bug/1895972

11.2.1 References

11.3 OVN supported DHCP options

This is a list of the current supported DHCP options in ML2/OVN:

11.3.1 IP version 4

Option name / code	OVN value
arp-timeout	arp_cache_timeout
bootfile-name	bootfile_name
classless-static-route	classless_static_route
default-ttl	default_ttl
dns-server	dns_server
domain-name	domain_name
domain-search	domain_search_list
ethernet-encap	ethernet_encap
ip-forward-enable	ip_forward_enable
lease-time	lease_time
log-server	log_server
lpr-server	lpr_server
ms-classless-static-route	ms_classless_static_route
mtu	mtu
netmask	netmask
nis-server	nis_server
ntp-server	ntp_server
path-prefix	path_prefix
policy-filter	policy_filter
router-discovery	router_discovery
router	router
router-solicitation	router_solicitation
server-id	server_id
server-ip-address	tftp_server_address
swap-server	swap_server
T1	T1
T2	T2
tcp-ttl	tcp_ttl
tcp-keepalive	tcp_keepalive_interval
tftp-server-address	tftp_server_address
tftp-server	tftp_server
wpad	wpad
1	netmask
3	router
6	dns_server
7	log_server
9	lpr_server
15	domain_name

continues on next page

Table 1 – continued from previous page

Option name / code	OVN value
16	swap_server
19	ip_forward_enable
21	policy_filter
23	default_ttl
26	mtu
31	router_discovery
32	router_solicitation
35	arp_cache_timeout
36	ethernet_encap
37	tcp_ttl
38	tcp_keepalive_interval
41	nis_server
42	ntp_server
51	lease_time
54	server_id
58	T1
59	T2
66	tftp_server
67	bootfile_name
119	domain_search_list
121	classless_static_route
150	tftp_server_address
210	path_prefix
249	ms_classless_static_route
252	wpad

11.3.2 IP version 6

Option name / code	OVN value
dns-server	dns_server
domain-search	domain_search
ia-addr	ia_addr
server-id	server_id
2	server_id
5	ia_addr
23	dns_server
24	domain_search

11.3.3 OVN Database information

In OVN the DHCP options are stored on a table called `DHCP_Options` in the OVN Northbound database.

Lets add a DHCP option to a Neutron port:

```
$ neutron port-update --extra-dhcp-opt opt_name='server-ip-address',opt_
↪value='10.0.0.1' b4c3f265-369e-4bf5-8789-7caa9a1efb9c
Updated port: b4c3f265-369e-4bf5-8789-7caa9a1efb9c
```

To find that port in OVN we can use command below:

```
$ ovn-nbctl find Logical_Switch_Port name=b4c3f265-369e-4bf5-8789-
↪7caa9a1efb9c
...
dhcpv4_options      : 5f00d1a2-c57d-4d1f-83ea-09bf8be13288
dhcpv6_options      : []
...
```

For DHCP, the columns that we care about are the `dhcpv4_options` and `dhcpv6_options`. These columns has the uuids of entries in the `DHCP_Options` table with the DHCP information for this port.

```
$ ovn-nbctl list DHCP_Options 5f00d1a2-c57d-4d1f-83ea-09bf8be13288
_uuid               : 5f00d1a2-c57d-4d1f-83ea-09bf8be13288
cidr                : "10.0.0.0/26"
external_ids        : {"neutron:revision_number"="0", port_id="b4c3f265-
↪369e-4bf5-8789-7caa9a1efb9c", subnet_id="5157ed8b-e7f1-4c56-b789-
↪fa420098a687"}
options             : {classless_static_route="{169.254.169.254/32,10.0.0.
↪2, 0.0.0.0/0,10.0.0.1}", dns_server="{8.8.8.8}", domain_name="\
↪"openstackgate.local\"", lease_time="43200", log_server="127.0.0.3", mtu=
↪"1442", router="10.0.0.1", server_id="10.0.0.1", server_mac=
↪"fa:16:3e:dc:57:22", tftp_server_address="10.0.0.1"}
```

Here you can see that the option `tftp_server_address` has been set in the **options** column. Note that, the `tftp_server_address` option is the OVN translated name for `server-ip-address` (option 150). Take a look at the table in this document to find out more about the supported options and their counterpart names in OVN.

11.4 Frequently Asked Questions

Q: What are the key differences between ML2/ovs and ML2/ovn?

Detail	ml2/ovs	ml2/ovn
agent/server communication	rabbit mq messaging + RPC.	ovsdb protocol on the NorthBound and SouthBound databases.
l3ha API	routers expose an ha field that can be disabled or enabled by admin with a deployment default.	routers dont expose an ha field, and will make use of HA as soon as there is more than one network node available.
l3ha dataplane	qrouter namespace with keepalive process and an internal ha network for VRRP traffic.	ovn-controller configures specific OpenFlow rules, and enables BFD protocol over tunnel endpoints to detect connectivity issues to nodes.
DVR API	exposes the distributed flag on routers only modifiable by admin.	no distributed flag is shown or available on routers via API.
DVR dataplane	uses namespaces, veths, ip routing, ip rules and iptables on the compute nodes.	Uses OpenFlow rules on the compute nodes.
E/W traffic	goes through network nodes when the router is not distributed (DVR).	completely distributed in all cases.
Metadata Service	Metadata service is provided by the qrouters or dhcp namespaces in the network nodes.	Metadata is completely distributed across compute nodes, and served from the ovnmeta-xxxxx-xxxx namespace.
DHCP Service	DHCP is provided via qdhcp-xxxxx-xxx namespaces which run dnsmasq inside.	DHCP is provided by OpenFlow and ovn-controller, being distributed across computes.
Trunk Ports	Trunk ports are built by creating br-trunk-xxx bridges and patch ports.	Trunk ports live in br-int as OpenFlow rules, while subports are directly attached to br-int.

Q: Why cant I use the distributed or ha flags of routers?

Networking OVN implements HA and distributed in a transparent way for the administrator and users.

HA will be automatically used on routers as soon as more than two gateway nodes are detected. And distributed floating IPs will be used as soon as its configured (see next question).

Q: Does OVN support DVR or distributed L3 routing?

Yes, its controlled by a single flag in configuration.

DVR will be used for floating IPs if the ovn / enable_distributed_floating_ip flag is configured to True in the neutron server configuration, being a deployment wide setting. In contrast to ML2/ovs which was able to specify this setting per router (only admin).

Although ovn driver does not expose the distributed flag of routers throught the API.

Q: Does OVN support integration with physical switches?

OVN currently integrates with physical switches by optionally using them as VTEP gateways from logical to physical networks and via integrations provided by the Neutron ML2 framework, hierarchical port binding.

Q: Whats the status of HA for ovn driver and OVN?

Typically, multiple copies of neutron-server are run across multiple servers and uses a load balancer. The neutron ML2 mechanism driver provided by ovn driver supports this deployment model. DHCP and metadata services are distributed across compute nodes, and dont depend on the network nodes.

The network controller portion of OVN is distributed - an instance of the ovn-controller service runs on every hypervisor. OVN also includes some central components for control purposes.

ovn-northd is a centralized service that does some translation between the northbound and southbound databases in OVN. Currently, you only run this service once. You can manage it in an active/passive HA mode using something like Pacemaker. The OVN project plans to allow this service to be horizontally scaled both for scaling and HA reasons. This will allow it to be run in an active/active HA mode.

OVN also makes use of ovsdb-server for the OVN northbound and southbound databases. ovsdb-server supports active/passive HA using replication. For more information, see: http://docs.openvswitch.org/en/latest/topics/ovsdb-replication/

A typical deployment would use something like Pacemaker to manage the active/passive HA process. Clients would be pointed at a virtual IP address. When the HA manager detects a failure of the master, the virtual IP would be moved and the passive replica would become the new master.

See *OVN information* for links to more details on OVNs architecture.

CHAPTER

TWELVE

API REFERENCE

The reference of the OpenStack networking API is found at https://docs.openstack.org/api-ref/network/.

CHAPTER

THIRTEEN

NEUTRON FEATURE CLASSIFICATION

13.1 Introduction

This document describes how features are listed in *General Feature Support* and *Provider Network Support*.

13.1.1 Goals

The object of this document is to inform users whether or not features are complete, well documented, stable, and tested. This approach ensures good user experience for those well maintained features.

Note: Tests are specific to particular combinations of technologies. The plugins chosen for deployment make a big difference to whether or not features will work.

13.1.2 Concepts

These definitions clarify the terminology used throughout this document.

13.1.3 Feature status

- Immature
- Mature
- Required
- Deprecated (scheduled to be removed in a future release)

Immature

Immature features do not have enough functionality to satisfy real world use cases.

An immature feature is a feature being actively developed, which is only partially functional and upstream tested, most likely introduced in a recent release, and that will take time to mature thanks to feedback from downstream QA.

Users of these features will likely identify gaps and/or defects that were not identified during specification and code review.

Mature

A feature is considered mature if it satisfies the following criteria:

- Complete API documentation including concept and REST call definition.
- Complete Administrator documentation.
- Tempest tests that define the correct functionality of the feature.
- Enough functionality and reliability to be useful in real world scenarios.
- Low probability of support for the feature being dropped.

Required

Required features are core networking principles that have been thoroughly tested and have been implemented in real world use cases.

In addition they satisfy the same criteria for any mature features.

Note: Any new drivers must prove that they support all required features before they are merged into neutron.

Deprecated

Deprecated features are no longer supported and only security related fixes or development will happen towards them.

Deployment rating of features

The deployment rating shows only the state of the tests for each feature on a particular deployment.

Important: Despite the obvious parallels that could be drawn, this list is unrelated to the DefCore effort. See InteropWG

13.2 General Feature Support

> **Warning:** Please note, while this document is still being maintained, this is slowly being updated to re-group and classify features using the definitions described in here: *Introduction*.

This document covers the maturity and support of the Neutron API and its API extensions. Details about the API can be found at Networking API v2.0.

When considering which capabilities should be marked as mature the following general guiding principles were applied:

- **Inclusivity** - people have shown ability to make effective use of a wide range of network plugins and drivers with broadly varying feature sets. Aiming to keep the requirements as inclusive as possible, avoids second-guessing how a user wants to use their networks.
- **Bootstrapping** - a practical use case test is to consider that starting point for the network deploy is an empty data center with new machines and network connectivity. Then look at what are the minimum features required of the network service, in order to get user instances running and connected over the network.
- **Reality** - there are many networking drivers and plugins compatible with neutron. Each with their own supported feature set.

Summary

Feature	*Status*	**Linux Bridge**	Networking MidoNet	Networking ODL	Networking OVN	**Open vSwitch**
Networks	mandatory	✓	✓	✓	✓	✓
Subnets	mandatory	✓	✓	✓	✓	✓
Ports	mandatory	✓	✓	✓	✓	✓
Routers	mandatory	✓	✓	✓	✓	✓
Security Groups	mature	✓	✓	✓	✓	✓
External Networks	mature	✓	✓	✓	✓	✓
Distributed Virtual Routers	immature		✓	✓	✓	✓
L3 High Availability	immature	✓		✓	✓	✓
Quality of Service	mature	✓	✓	✓	✓	✓
Border Gateway Protocol	immature	?	✓	?	?	✓
DNS	mature	✓		✓	✓	✓
Trunk Ports	mature	✓			✓	✓
Metering	mature	✓			?	✓

Details

- **Networks Status: mandatory.**

  API Alias: core

  CLI commands:

  - `openstack network *`

  Notes: The ability to create, modify and delete networks. https://docs.openstack.org/api-ref/network/v2/#networks

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `complete`
  - **Networking ODL:** `complete`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

- **Subnets Status: mandatory.**

API Alias: core

CLI commands:

- `openstack subnet *`

Notes: The ability to create and manipulate subnets and subnet pools. https://docs.openstack.org/api-ref/network/v2/#subnets

Driver Support:

- **Linux Bridge:** `complete`
- **Networking MidoNet:** `complete`
- **Networking ODL:** `complete`
- **Networking OVN:** `complete`
- **Open vSwitch:** `complete`

• **Ports Status: mandatory.**

API Alias: core

CLI commands:

- `openstack port *`

Notes: The ability to create and manipulate ports. https://docs.openstack.org/api-ref/network/v2/#ports

Driver Support:

- **Linux Bridge:** `complete`
- **Networking MidoNet:** `complete`
- **Networking ODL:** `complete`
- **Networking OVN:** `complete`
- **Open vSwitch:** `complete`

• **Routers Status: mandatory.**

API Alias: router

CLI commands:

- `openstack router *`

Notes: The ability to create and manipulate routers. https://docs.openstack.org/api-ref/network/v2/#routers-routers

Driver Support:

- **Linux Bridge:** `complete`
- **Networking MidoNet:** `complete`
- **Networking ODL:** `complete`
- **Networking OVN:** `complete`
- **Open vSwitch:** `complete`

- **Security Groups Status: mature.**

  API Alias: security-group

  CLI commands:

  – `openstack security group *`

  Notes: Security groups are set by default, and can be modified to control ingress & egress traffic. https://docs.openstack.org/api-ref/network/v2/#security-groups-security-groups

  Driver Support:

  – **Linux Bridge:** `complete`

  – **Networking MidoNet:** `complete`

  – **Networking ODL:** `complete`

  – **Networking OVN:** `complete`

  – **Open vSwitch:** `complete`

- **External Networks Status: mature.**

  API Alias: external-net

  Notes: The ability to create an external network to provide internet access to and from instances using floating IP addresses and security group rules.

  Driver Support:

  – **Linux Bridge:** `complete`

  – **Networking MidoNet:** `complete`

  – **Networking ODL:** `complete`

  – **Networking OVN:** `complete`

  – **Open vSwitch:** `complete`

- **Distributed Virtual Routers Status: immature.**

  API Alias: dvr

  Notes: The ability to support the distributed virtual routers. https://wiki.openstack.org/wiki/Neutron/DVR

  Driver Support:

  – **Linux Bridge:** `missing`

  – **Networking MidoNet:** `complete`

  – **Networking ODL:** `partial`

  – **Networking OVN:** `partial`

  – **Open vSwitch:** `complete`

- **L3 High Availability Status: immature.**

  API Alias: l3-ha

  Notes: The ability to support the High Availability features and extensions. https://wiki.openstack.org/wiki/Neutron/L3_High_Availability_VRRP.

Driver Support:

- **Linux Bridge:** `complete`
- **Networking MidoNet:** `missing`
- **Networking ODL:** `partial`
- **Networking OVN:** `partial`
- **Open vSwitch:** `complete`

- **Quality of Service Status: mature.**

  API Alias: qos

  Notes: Support for Neutron Quality of Service policies and API. https://docs.openstack.org/api-ref/network/v2/#qos-policies-qos

  Driver Support:

  - **Linux Bridge:** `partial`
  - **Networking MidoNet:** `complete`
  - **Networking ODL:** `partial`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

- **Border Gateway Protocol Status: immature.**

  Notes: https://docs.openstack.org/api-ref/network/v2/#bgp-mpls-vpn-interconnection

  Driver Support:

  - **Linux Bridge:** `unknown`
  - **Networking MidoNet:** `complete`
  - **Networking ODL:** `unknown`
  - **Networking OVN:** `unknown`
  - **Open vSwitch:** `complete`

- **DNS Status: mature.**

  API Alias: dns-integration

  Notes: The ability to integrate with an external DNS as a Service. https://docs.openstack.org/neutron/latest/admin/config-dns-int.html

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `complete`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

- **Trunk Ports Status: mature.**

  API Alias: trunk

  Notes: Neutron extension to access lots of neutron networks over a single vNIC as tagged/encapsulated traffic. https://docs.openstack.org/api-ref/network/v2/#trunk-networking

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `missing`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

- **Metering Status: mature.**

  API Alias: metering

  Notes: Meter traffic at the L3 router levels. https://docs.openstack.org/api-ref/network/v2/#metering-labels-and-rules-metering-labels-metering-label-rules

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `missing`
  - **Networking OVN:** `unknown`
  - **Open vSwitch:** `complete`

Notes:

- **This document is a continuous work in progress**

13.3 Provider Network Support

> **Warning:** Please note, while this document is still being maintained, this is slowly being updated to re-group and classify features using the definitions described in here: *Introduction*.

This document covers the maturity and support for various network isolation technologies.

When considering which capabilities should be marked as mature the following general guiding principles were applied:

- **Inclusivity** - people have shown ability to make effective use of a wide range of network plugins and drivers with broadly varying feature sets. Aiming to keep the requirements as inclusive as possible, avoids second-guessing how a user wants to use their networks.
- **Bootstrapping** - a practical use case test is to consider that starting point for the network deploy is an empty data center with new machines and network connectivity. Then look at what are

the minimum features required of the network service, in order to get user instances running and connected over the network.

- **Reality** - there are many networking drivers and plugins compatible with neutron. Each with their own supported feature set.

Summary

Feature	*Status*	**Linux Bridge**	Networking MidoNet	Networking ODL	Networking OVN	**Open vSwitch**
VLAN provider network support	mature	✓		?	✓	✓
VXLAN provider network support	mature	✓		✓		✓
GRE provider network support	immature	?		✓		✓
Geneve provider network support	immature	?			✓	✓

Details

- **VLAN provider network support Status: mature.**

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `unknown`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

- **VXLAN provider network support Status: mature.**

  Driver Support:

  - **Linux Bridge:** `complete`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `complete`
  - **Networking OVN:** `missing`
  - **Open vSwitch:** `complete`

- **GRE provider network support Status: immature.**

  Driver Support:

  - **Linux Bridge:** `unknown`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `complete`
  - **Networking OVN:** `missing`

  - **Open vSwitch:** `complete`

- **Geneve provider network support Status: immature.**

  Driver Support:

  - **Linux Bridge:** `unknown`
  - **Networking MidoNet:** `missing`
  - **Networking ODL:** `missing`
  - **Networking OVN:** `complete`
  - **Open vSwitch:** `complete`

Notes:

- **This document is a continuous work in progress**

CHAPTER

FOURTEEN

CONTRIBUTOR GUIDE

This document describes Neutron for contributors of the project, and assumes that you are already familiar with Neutron from an *end-user perspective*.

14.1 Basic Information

14.1.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Neutron.

Communication

- IRC channel: #openstack-neutron
- Mailing lists prefix: [neutron]
- Team Meeting:

  This is general Neutron team meeting. The discussion in this meeting is about all things related to the Neutron project, like community goals, progress with blueprints, bugs, etc. There is also `On Demand Agenda` at the end of this meeting, where anyone can add a topic to discuss with the Neutron team.

  - time: http://eavesdrop.openstack.org/#Neutron_Team_Meeting
  - agenda: https://wiki.openstack.org/wiki/Network/Meetings
- Drivers team meeting:

  This is the meeting where Neutron drivers discuss about new RFEs.

  - time: http://eavesdrop.openstack.org/#Neutron_drivers_Meeting
  - agenda: https://wiki.openstack.org/wiki/Meetings/NeutronDrivers
- Neutron CI team meeting:

  This is the meeting where upstream CI issues are discussed every week. If You are interested in helping our CI to be green, thats good place to join and help.

  - time: http://eavesdrop.openstack.org/#Neutron_CI_team

  - agenda: https://etherpad.openstack.org/p/neutron-ci-meetings

- Neutron QoS team meeting:

  This is the meeting of the Neutron Quality of Service subteam.

  - time: http://eavesdrop.openstack.org/#Neutron_QoS_Meeting

- Neutron L3 team meeting:

  This is the meeting of the Neutron L3 subteam where all issues related to IPAM, L3 agents, etc. are discussed.

  - time: http://eavesdrop.openstack.org/#Neutron_L3_Sub-team_Meeting
  - agenda: https://etherpad.openstack.org/p/neutron-l3-subteam

Contacting the Core Team

The list of current Neutron core reviewers is available on gerrit. Overall structure of Neutron team is available in *Neutron teams*.

New Feature Planning

Neutron team uses `RFE (Request for Enhancements)` to propose new features. RFE should be submitted as a Launchpad bug first (see section *Reporting a Bug*). The title of RFE bug should starts with `[RFE]` tag. Such RFEs need to be discussed and approved by the *Neutron drivers team*. In some cases an additional spec proposed to the Neutron specs repo may be necessary. The complete process is described in detail in *Blueprints guide*.

Task Tracking

We track our tasks in Launchpad. If youre looking for some smaller, easier work item to pick up and get started on, search for the Low hanging fruit tag. List of all official tags which Neutron team is using is available on *bugs*. Every week, one of our team members is the *bug deputy* and at the end of the week such person usually sends report about new bugs to the mailing list openstack-discuss@lists.openstack.org or talks about it on our team meeting. This is also good place to look for some work to do.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Launchpad. More info about Launchpad usage can be found on OpenStack docs page.

Getting Your Patch Merged

All changes proposed to the Neutron or one of the Neutron stadium projects require two +2 votes from Neutron core reviewers before one of the core reviewers can approve patch by giving `Workflow +1` vote. More detailed guidelines for reviewers of Neutron patches are available at *Code reviews guide*.

Project Team Lead Duties

Neutrons PTL duties are described very well in the All common PTL duties guide. Additionally to what is described in this guide, Neutrons PTL duties are:

- triage new RFEs and prepare Neutron drivers team meeting,
- maintain list of the *stadium projects* health - if each project has gotten active team members and if it is following community and Neutrons guidelines and goals,
- maintain list of the *stadium projects lieutenants* - check if those people are still active in the projects, if their contact data are correct, maybe there is someone new who is active in the stadium project and could be added to this list.

Over the past few years, the Neutron team has followed a mentoring approach for:

- new contributors,
- potential new core reviewers,
- future PTLs.

The Neutron PTLs responsibility is to identify potential new core reviewers and help with their mentoring process. Mentoring of new contributors and potential core reviewers can be of course delegated to the other members of the Neutron team. Mentoring of future PTLs is responibility of the Neutron PTL.

14.2 Neutron Policies

14.2.1 Neutron Policies

In the Policies Guide, you will find documented policies for developing with Neutron. This includes the processes we use for blueprints and specs, bugs, contributor onboarding, core reviewer memberships, and other procedural items.

Blueprints and Specs

The Neutron team uses the neutron-specs repository for its specification reviews. Detailed information can be found on the wiki. Please also find additional information in the reviews.rst file.

The Neutron team does not enforce deadlines for specs. These can be submitted throughout the release cycle. The drivers team will review this on a regular basis throughout the release, and based on the load for the milestones, will assign these into milestones or move them to the backlog for selection into a future release.

Please note that we use a template for spec submissions. It is not required to fill out all sections in the template. Review of the spec may require filling in information left out by the submitter.

Sub-Projects and Specs

The neutron-specs repository is only meant for specs from Neutron itself, and the advanced services repositories as well. This includes FWaaS and VPNaaS. Other sub-projects are encouraged to fold their specs into their own devref code in their sub-project gerrit repositories. Please see additional comments in the Neutron teams *section* for reviewer requirements of the neutron-specs repository.

Neutron Request for Feature Enhancements

In Liberty the team introduced the concept of feature requests. Feature requests are tracked as Launchpad bugs, by tagging them with a set of tags starting with *rfe*, enabling the submission and review of feature requests before code is submitted. This allows the team to verify the validity of a feature request before the process of submitting a neutron-spec is undertaken, or code is written. It also allows the community to express interest in a feature by subscribing to the bug and posting a comment in Launchpad. The rfe tag should not be used for work that is already well-defined and has an assignee. If you are intending to submit code immediately, a simple bug report will suffice. Note the temptation to game the system exists, but given the history in Neutron for this type of activity, it will not be tolerated and will be called out as such in public on the mailing list.

RFEs can be submitted by anyone and by having the community vote on them in Launchpad, we can gauge interest in features. The drivers team will evaluate these on a weekly basis along with the specs. RFEs will be evaluated in the current cycle against existing project priorities and available resources.

The workflow for the life an RFE in Launchpad is as follows:

- The bug is submitted and will by default land in the New state. Anyone can make a bug an RFE by adding the *rfe* tag.
- As soon as a member of the neutron-drivers team acknowledges the bug, the *rfe* tag will be replaced with the *rfe-confirmed* tag. No assignee, or milestone is set at this time. The importance will be set to Wishlist to signal the fact that the report is indeed a feature or enhancement and there is no severity associated to it.
- A member of the neutron-drivers team replaces the *rfe-confirmed* tag with the *rfe-triaged* tag when he/she thinks its ready to be discussed in the drivers meeting. The bug will be in this state while the discussion is ongoing.
- The neutron-drivers team will evaluate the RFE and may advise the submitter to file a spec in neutron-specs to elaborate on the feature request, in case the RFE requires extra scrutiny, more design discussion, etc.
- The PTL will work with the Lieutenant for the area being identified by the RFE to evaluate resources against the current workload.
- A member of the Neutron release team (or the PTL) will register a matching Launchpad blueprint to be used for milestone tracking purposes, and for identifying the responsible assignee and approver. If the RFE has a spec the blueprint will have a pointer to the spec document, which will become available on specs.o.o. once it is approved and merged. The blueprint will then be linked to the original RFE bug report as a pointer to the discussion that led to the approval of the RFE. The blueprint submitter will also need to identify the following:
  - Priority: there will be only two priorities to choose from, High and Low. It is worth noting that priority is not to be confused with importance, which is a property of Launchpad Bugs. Priority gives an indication of how promptly a work item should be tackled to allow it to

complete. High priority is to be chosen for work items that must make substantial progress in the span of the targeted release, and deal with the following aspects:

  * OpenStack cross-project interaction and interoperability issues;
  * Issues that affect the existing systems usability;
  * Stability and testability of the platform;
  * Risky implementations that may require complex and/or pervasive changes to API and the logical model;

Low priority is to be chosen for everything else. RFEs without an associated blueprint are effectively equivalent to low priority items. Bear in mind that, even though staffing should take priorities into account (i.e. by giving more resources to high priority items over low priority ones), the open source reality is that they can both proceed at their own pace and low priority items can indeed complete faster than high priority ones, even though they are given fewer resources.

- Drafter: who is going to submit and iterate on the spec proposal; he/she may be the RFE submitter.
- Assignee: who is going to develop the bulk of the code, or the go-to contributor, if more people are involved. Typically this is the RFE submitter, but not necessarily.
- Approver: a member of the Neutron team who can commit enough time during the ongoing release cycle to ensure that code posted for review does not languish, and that all aspects of the feature development are taken care of (client, server changes and/or support from other projects if needed - tempest, nova, openstack-infra, devstack, etc.), as well as comprehensive testing. This is typically a core member who has enough experience with what it takes to get code merged, but other resources amongst the wider team can also be identified. Approvers are volunteers who show a specific interest in the blueprint specification, and have enough insight in the area of work so that they can make effective code reviews and provide design feedback. An approver will not work in isolation, as he/she can and will reach out for help to get the job done; however he/she is the main point of contact with the following responsibilities:

  * Pair up with the drafter/assignee in order to help skip development blockers.
  * Review patches associated with the blueprint: approver and assignee should touch base regularly and ping each other when new code is available for review, or if review feedback goes unaddressed.
  * Reach out to other reviewers for feedback in areas that may step out of the zone of her/his confidence.
  * Escalate issues, and raise warnings to the release team/PTL if the effort shows slow progress. Approver and assignee are key parts to land a blueprint: should the approver and/or assignee be unable to continue the commitment during the release cycle, it is the Approvers responsibility to reach out the release team/PTL so that replacements can be identified.
  * Provide a status update during the Neutron IRC meeting, if required.

Approver assignments must be carefully identified to ensure that no-one overcommits. A Neutron contributor develops code himself/herself, and if he/she is an approver of more than a couple of blueprints in a single cycle/milestone (depending on the complexity of the spec), it may mean that he/she is clearly oversubscribed.

The Neutron team will review the status of blueprints targeted for the milestone during their weekly meeting to ensure a smooth progression of the work planned. Blueprints for which resources cannot be identified will have to be deferred.

- In either case (a spec being required or not), once the discussion has happened and there is positive consensus on the RFE, the report is approved, and its tag will move from *rfe-triaged* to *rfe-approved*.
- An RFE can be occasionaly marked as rfe-postponed if the team identifies a dependency between the proposed RFE and other pending tasks that prevent the RFE from being worked on immediately.
- Once an RFE is approved, it needs volunteers. Approved RFEs that do not have an assignee but sound relatively simple or limited in scope (e.g. the addition of a new API with no ramification in the plugin backends), should be promoted during team meetings or the ML so that volunteers can pick them up and get started with neutron development. The team will regularly scan *rfe-approved* or *rfe-postponed* RFEs to see what their latest status is and mark them incomplete if no assignees can be found, or they are no longer relevant.
- As for setting the milestone (both for RFE bugs or blueprints), the current milestone is always chosen, assuming that work will start as soon as the feature is approved. Work that fails to complete by the defined milestone will roll over automatically until it gets completed or abandoned.
- If the code fails to merge, the bug report may be marked as incomplete, unassigned and untargeted, and it will be garbage collected by the Launchpad Janitor if no-one takes over in time. Renewed interest in the feature will have to go through RFE submission process once again.

In summary:

State	Meaning
New	This is where all RFEs start, as filed by the community.
Incomplete	Drivers/LTs - Move to this state to mean, more needed before proceeding
Confirmed	Drivers/LTs - Move to this state to mean, yeah, I see that you filed it
Triaged	Drivers/LTs - Move to this state to mean, discussion is ongoing
Wont Fix	Drivers/LTs - Move to this state to reject an RFE.

Once the triaging (discussion is complete) and the RFE is approved, the tag goes from rfe to rfe-approved, and at this point the bug report goes through the usual state transition. Note, that the importance will be set to wishlist, to reflect the fact that the bug report is indeed not a bug, but a new feature or enhancement. This will also help have RFEs that are not followed up by a blueprint standout in the Launchpad milestone dashboards.

The drivers team will be discussing the following bug reports during their IRC meeting:

- New RFEs
- Incomplete RFEs
- Confirmed RFEs
- Triaged RFEs

RFE Submission Guidelines

Before we dive into the guidelines for writing a good RFE, it is worth mentioning that depending on your level of engagement with the Neutron project and your role (user, developer, deployer, operator, etc.), you are more than welcome to have a preliminary discussion of a potential RFE by reaching out to other people involved in the project. This usually happens by posting mails on the relevant mailing lists (e.g. openstack-discuss - include [neutron] in the subject) or on #openstack-neutron IRC channel on Freenode. If current ongoing code reviews are related to your feature, posting comments/questions on gerrit may also be a way to engage. Some amount of interaction with Neutron developers will give you an idea of the plausibility and form of your RFE before you submit it. That said, this is not mandatory.

When you submit a bug report on https://bugs.launchpad.net/neutron/+filebug, there are two fields that must be filled: summary and further information. The summary must be brief enough to fit in one line: if you cant describe it in a few words it may mean that you are either trying to capture more than one RFE at once, or that you are having a hard time defining what you are trying to solve at all.

The further information section must be a description of what you would like to see implemented in Neutron. The description should provide enough details for a knowledgeable developer to understand what is the existing problem in the current platform that needs to be addressed, or what is the enhancement that would make the platform more capable, both for a functional and a non-functional standpoint. To this aim it is important to describe why you believe the RFE should be accepted, and motivate the reason why without it Neutron is a poorer platform. The description should be self contained, and no external references should be necessary to further explain the RFE.

In other words, when you write an RFE you should ask yourself the following questions:

- What is that I (specify what user - a user can be a human or another system) cannot do today when interacting with Neutron? On the other hand, is there a Neutron component X that is unable to accomplish something?
- Is there something that you would like Neutron handle better, ie. in a more scalable, or in a more reliable way?
- What is that I would like to see happen after the RFE is accepted and implemented?
- Why do you think it is important?

Once you are happy with what you wrote, add rfe as tag, and submit. Do not worry, we are here to help you get it right! Happy hacking.

Missing your target

There are occasions when a spec will be approved and the code will not land in the cycle it was targeted at. For these cases, the work flow to get the spec into the next release is as follows:

- During the RC window, the PTL will create a directory named <release> under the backlog directory in the neutron specs repo, and he/she will move all specs that did not make the release to this directory.
- Anyone can propose a patch to neutron-specs which moves a spec from the previous release into the new release directory.

The specs which are moved in this way can be fast-tracked into the next release. Please note that it is required to re-propose the spec for the new release.

Documentation

The above process involves two places where any given feature can start to be documented - namely in the RFE bug, and in the spec - and in addition to those Neutron has a substantial *developer reference guide* (aka devref), and user-facing docs such as the *networking guide*. So it might be asked:

- What is the relationship between all of those?
- What is the point of devref documentation, if everything has already been described in the spec?

The answers have been beautifully expressed in an openstack-dev post:

1. RFE: I want X
2. Spec: I plan to implement X like this
3. devref: How X is implemented and how to extend it
4. OS docs: API and guide for using X

Once a feature X has been implemented, we shouldnt have to go to back to its RFE bug or spec to find information on it. The devref may reuse a lot of content from the spec, but the spec is not maintained and the implementation may differ in some ways from what was intended when the spec was agreed. The devref should be kept current with refactorings, etc., of the implementation.

Devref content should be added as part of the implementation of a new feature. Since the spec is not maintained after the feature is implemented, the devref should include a maintained version of the information from the spec.

If a feature requires OS docs (4), the feature patch shall include the new, or updated, documentation changes. If the feature is purely a developer facing thing, (4) is not needed.

Neutron Bugs

Neutron (client, core, FwaaS, VPNaaS) maintains all of its bugs in the following Launchpad projects:

- Launchpad Neutron
- Launchpad python-neutronclient

Neutron Bugs Team In Launchpad

The Neutron Bugs team in Launchpad is used to allow access to the projects above. Members of the above group have the ability to set bug priorities, target bugs to releases, and other administrative tasks around bugs. The administrators of this group are the members of the neutron-drivers-core gerrit group. Non administrators of this group include anyone who is involved with the Neutron project and has a desire to assist with bug triage.

If you would like to join this Launchpad group, its best to reach out to a member of the above mentioned neutron-drivers-core team in #openstack-neutron on Freenode and let them know why you would like to be a member. The team is more than happy to add additional bug triage capability, but it helps to know who is requesting access, and IRC is a quick way to make the connection.

As outlined below the bug deputy is a volunteer who wants to help with defect management. Permissions will have to be granted assuming that people sign up on the deputy role. The permission wont be given freely, a person must show some degree of prior involvement.

Neutron Bug Deputy

Neutron maintains the notion of a bug deputy. The bug deputy plays an important role in the Neutron community. As a large project, Neutron is routinely fielding many bug reports. The bug deputy is responsible for acting as a first contact for these bug reports and performing initial screening/triaging. The bug deputy is expected to communicate with the various Neutron teams when a bug has been triaged. In addition, the bug deputy should be reporting High and Critical priority bugs.

To avoid burnout, and to give a chance to everyone to gain experience in defect management, the Neutron bug deputy is a rotating role. The rotation will be set on a period (typically one or two weeks) determined by the team during the weekly Neutron IRC meeting and/or according to holidays. During the Neutron IRC meeting we will expect a volunteer to step up for the period. Members of the Neutron core team are invited to fill in the role, however non-core Neutron contributors who are interested are also encouraged to take up the role.

This contributor is going to be the bug deputy for the period, and he/she will be asked to report to the team during the subsequent IRC meeting. The PTL will also work with the team to assess that everyone gets his/her fair share at fulfilling this duty. It is reasonable to expect some imbalance from time to time, and the team will work together to resolve it to ensure that everyone is 100% effective and well rounded in their role as _custodian_ of Neutron quality. Should the duty load be too much in busy times of the release, the PTL and the team will work together to assess whether more than one deputy is necessary in a given period.

The presence of a bug deputy does not mean the rest of the team is simply off the hook for the period, in fact the bug deputy will have to actively work with the Lieutenants/Drivers, and these should help in getting the bug report moving down the resolution pipeline.

During the period a member acts as bug deputy, he/she is expected to watch bugs filed against the Neutron projects (as listed above) and do a first screening to determine potential severity, tagging, logstash queries, other affected projects, affected releases, etc.

From time to time bugs will be filed and auto-assigned by members of the core team to get them to a swift resolution. Obviously, the deputy is exempt from screening these.

Finally, the PTL will work with the deputy to produce a brief summary of the issues of the week to be shared with the larger team during the weekly IRC meeting and tracked in the meeting notes. If for some reason the deputy is not going to attend the team meeting to report, the deputy should consider sending a brief report to the openstack-discuss@ mailing list in advance of the meeting.

Getting Ready to Serve as the Neutron Bug Deputy

If you are interested in serving as the Neutron bug deputy, there are several steps you will need to follow in order to be prepared.

- Request to be added to the neutron-bugs team in Launchpad. This request will be approved when you are assigned a bug deputy slot.
- Read this page in full. Keep this document in mind at all times as it describes the duties of the bug deputy and how to triage bugs particularly around setting the importance and tags of bugs.
- Sign up for neutron bug emails from LaunchPad.
  - Navigate to the LaunchPad Neutron bug list.
  - On the right hand side, click on Subscribe to bug mail.

    - In the pop-up that is displayed, keep the recipient as Yourself, and your subscription something useful like Neutron Bugs. You can choose either option for how much mail you get, but keep in mind that getting mail for all changes - while informative - will result in several dozen emails per day at least.
    - Do the same for the LaunchPad python-neutronclient bug list.
- Configure the information you get from LaunchPad to make visible additional information, especially the age of the bugs. You accomplish that by clicking the little gear on the left hand side of the screen at the top of the bugs list. This provides an overview of information for each bug on a single page.
- Optional: Set up your mail client to highlight bug email that indicates a new bug has been filed, since those are the ones you will be wanting to triage. Filter based on email from @bugs.launchpad.net with [NEW] in the subject line.
- Volunteer during the course of the Neutron team meeting, when volunteers to be bug deputy are requested (usually towards the beginning of the meeting).
- View your scheduled week on the Neutron Meetings page.
- During your shift, if it is feasible for your timezone, plan on attending the Neutron Drivers meeting. That way if you have tagged any bugs as RFE, you can be present to discuss them.

Bug Deputy routines in your week

- Scan New bugs to triage. If it doesnt have enough info to triage, ask more info and mark it Incomplete. If you could confirm it by yourself, mark it Confirmed. Otherwise, find someone familiar with the topic and ask his/her help.
- Scan Incomplete bugs to see if it got more info. If it was, make it back to New.
- Repeat the above routines for bugs filed in your week at least. If you can, do the same for older bugs.
- Take a note of bugs you processed. At the end of your week, post a report on openstack-discuss mailing list.

Plugin and Driver Repositories

Many plugins and drivers have backend code that exists in another repository. These repositories may have their own Launchpad projects for bugs. The teams working on the code in these repos assume full responsibility for bug handling in those projects. For this reason, bugs whose solution would exist solely in the plugin/driver repo should not have Neutron in the affected projects section. However, you should add Neutron (Or any other project) to that list only if you expect that a patch is needed to that repo in order to solve the bug.

Its also worth adding that some of these projects are part of the so called Neutron stadium. Because of that, their release is managed centrally by the Neutron release team; requests for releases need to be funnelled and screened properly before they can happen. Release request process is described *here*.

Bug Screening Best Practices

When screening bug reports, the first step for the bug deputy is to assess how well written the bug report is, and whether there is enough information for anyone else besides the bug submitter to reproduce the bug and come up with a fix. There is plenty of information on the OpenStack Bugs on how to write a good bug report and to learn how to tell a good bug report from a bad one. Should the bug report not adhere to these best practices, the bug deputys first step would be to redirect the submitter to this section, invite him/her to supply the missing information, and mark the bug report as Incomplete. For future submissions, the reporter can then use the template provided below to ensure speedy triaging. Done often enough, this practice should (ideally) ensure that in the long run, only good bug reports are going to be filed.

Bug Report Template

The more information you provide, the higher the chance of speedy triaging and resolution: identifying the problem is half the solution. To this aim, when writing a bug report, please consider supplying the following details and following these suggestions:

- Summary (Bug title): keep it small, possibly one line. If you cannot describe the issue in less than 100 characters, you are probably submitting more than one bug at once.
- Further information (Bug description): conversely from other bug trackers, Launchpad does not provide a structured way of submitting bug-related information, but everything goes in this section. Therefore, you are invited to break down the description in the following fields:
  - High level description: provide a brief sentence (a couple of lines) of what are you trying to accomplish, or would like to accomplish differently; the why is important, but can be omitted if obvious (not to you of course).
  - Pre-conditions: what is the initial state of your system? Please consider enumerating resources available in the system, if useful in diagnosing the problem. Who are you? A regular user or a super-user? Are you describing service-to-service interaction?
  - Step-by-step reproduction steps: these can be actual neutron client commands or raw API requests; Grab the output if you think it is useful. Please, consider using paste.o.o for long outputs as Launchpad poorly format the description field, making the reading experience somewhat painful.
  - Expected output: what did you hope to see? How would you have expected the system to behave? A specific error/success code? The output in a specific format? Or more than a user was supposed to see, or less?
  - Actual output: did the system silently fail (in this case log traces are useful)? Did you get a different response from what you expected?
  - Version:
    * OpenStack version (Specific stable branch, or git hash if from trunk);
    * Linux distro, kernel. For a distro, its also worth knowing specific versions of client and server, not just major release;
    * Relevant underlying processes such as openvswitch, iproute etc;
    * DevStack or other _deployment_ mechanism?

  - Environment: what services are you running (core services like DB and AMQP broker, as well as Nova/hypervisor if it matters), and which type of deployment (clustered servers); if you are running DevStack, is it a single node? Is it multi-node? Are you reporting an issue in your own environment or something you encountered in the OpenStack CI Infrastructure, aka the Gate?
  - Perceived severity: what would you consider the importance to be?
- Tags (Affected component): try to use the existing tags by relying on auto-completion. Please, refrain from creating new ones, if you need new official *tags*, please reach out to the PTL. If you would like a fix to be backported, please add a backport-potential tag. This does not mean you are gonna get the backport, as the stable team needs to follow the stable branch policy for merging fixes to stable branches.
- Attachments: consider attaching logs, truncated log snippets are rarely useful. Be proactive, and consider attaching redacted configuration files if you can, as that will speed up the resolution process greatly.

Bug Triage Process

The process of bug triaging consists of the following steps:

- Check if a bug was filed for a correct component (project). If not, either change the project or mark it as Invalid.
- For bugs that affect documentation proceed like this. If documentation affects:
  - the ReST API, add the api-ref tag to the bug.
  - the OpenStack manuals, like the Networking Guide or the Configuration Reference, create a patch for the affected files in the documentation directory in this repository. For a layout of the how the documentation directory is structured see the effective neutron guide
  - developer documentation (devref), set the bug to Confirmed for the project Neutron, otherwise set it to Invalid.
- Check if a similar bug was filed before. Rely on your memory if Launchpad is not clever enough to spot a duplicate upon submission. You may also check already verified bugs for Neutron and python-neutronclient to see if the bug has been reported. If so, mark it as a duplicate of the previous bug.
- Check if the bug meets the requirements of a good bug report, by checking that the *guidelines* are being followed. Omitted information is still acceptable if the issue is clear nonetheless; use your good judgement and your experience. Consult another core member/PTL if in doubt. If the bug report needs some love, mark the bug as Incomplete, point the submitter to this document and hope he/she turns around quickly with the missing information.

If the bug report is sound, move next:

- Revise tags as recommended by the submitter. Ensure they are official tags. If the bug report talks about deprecating features or config variables, add a deprecation tag to the list.
- As deputy one is usually excused not to process RFE bugs which are the responsibility of the drivers team members.
- Depending on ease of reproduction (or if the issue can be spotted in the code), mark it as Confirmed. If you are unable to assess/triage the issue because you do not have access to a repro en-

vironment, consider reaching out the *Lieutenant*, go-to person for the affected component; he/she may be able to help: assign the bug to him/her for further screening. If the bug already has an assignee, check that a patch is in progress. Sometimes more than one patch is required to address an issue, make sure that there is at least one patch that Closes the bug or document/question what it takes to mark the bug as fixed.

- If the bug indicates test or gate failure, look at the failures for that test over time using OpenStack Health or OpenStack Logstash. This can help to validate whether the bug identifies an issue that is occurring all of the time, some of the time, or only for the bug submitter.
- If the bug is the result of a misuse of the system, mark the bug either as Wont fix, or Opinion if you are still on the fence and need other peoples input.
- Assign the importance after reviewing the proposed severity. Bugs that obviously break core and widely used functionality should get assigned as High or Critical importance. The same applies to bugs that were filed for gate failures.
- Choose a milestone, if you can. Targeted bugs are especially important close to the end of the release.
- (Optional). Add comments explaining the issue and possible strategy of fixing/working around the bug. Also, as good as some are at adding all thoughts to bugs, it is still helpful to share the in-progress items that might not be captured in a bug description or during our weekly meeting. In order to provide some guidance and reduce ramp up time as we rotate, tagging bugs with needs-attention can be useful to quickly identify what reports need further screening/eyes on.

Check for Bugs with the timeout-abandon tag:

- Search for any bugs with the timeout abandon tag: Timeout abandon. This tag indicates that the bug had a patch associated with it that was automatically abandoned after a timing out with negative feedback.
- For each bug with this tag, determine if the bug is still valid and update the status accordingly. For example, if another patch fixed the bug, ensure its marked as Fix Released. Or, if that was the only patch for the bug and its still valid, mark it as Confirmed.
- After ensuring the bug report is in the correct state, remove the timeout-abandon tag.

You are done! Iterate.

Bug Expiration Policy and Bug Squashing

More can be found at this Launchpad page. In a nutshell, in order to make a bug report expire automatically, it needs to be unassigned, untargeted, and marked as Incomplete.

The OpenStack community has had Bug Days but they have not been wildly successful. In order to keep the list of open bugs set to a manageable number (more like <100+, rather than closer to 1000+), at the end of each release (in feature freeze and/or during less busy times), the PTL with the help of team will go through the list of open (namely new, opinion, in progress, confirmed, triaged) bugs, and do a major sweep to have the Launchpad Janitor pick them up. This gives 60 days grace period to reporters/assignees to come back and revive the bug. Assuming that at regime, bugs are properly reported, acknowledged and fix-proposed, losing unaddressed issues is not going to be a major issue, but brief stats will be collected to assess how the team is doing over time.

Tagging Bugs

Launchpads Bug Tracker allows you to create ad-hoc groups of bugs with tagging.

In the Neutron team, we have a list of agreed tags that we may apply to bugs reported against various aspects of Neutron itself. The list of approved tags used to be available on the wiki, however the section has been moved here, to improve collaborative editing, and keep the information more current. By using a standard set of tags, each explained on this page, we can avoid confusion. A bug report can have more than one tag at any given time.

Proposing New Tags

New tags, or changes in the meaning of existing tags (or deletion), are to be proposed via patch to this section. After discussion, and approval, a member of the bug team will create/delete the tag in Launchpad. Each tag covers an area with an identified go-to contact or *Lieutenant*, who can provide further insight. Bug queries are provided below for convenience, more will be added over time if needed.

Tag	Description	Contact
access-control	A bug affecting RBAC and policy.json	Miguel Lavalle
api	A bug affecting the API layer	Akihiro Motoki
api-ref	A bug affecting the API reference	Akihiro Motoki
auto-allocated-topology	A bug affecting get-me-a-network	N/A
baremetal	A bug affecting Ironic support	N/A
db	A bug affecting the DB layer	Nate Johnston
deprecation	To track config/feature deprecations	Neutron PTL/drivers
dns	A bug affecting DNS integration	Miguel Lavalle
doc	A bug affecting in-tree doc	Akihiro Motoki
fullstack	A bug in the fullstack subtree	Rodolfo Alonso Hernandez
functional-tests	A bug in the functional tests subtree	Rodolfo Alonso Hernandez
fwaas	A bug affecting neutron-fwaas	Nate Johnston
gate-failure	A bug affecting gate stability	Slawek Kaplonski
ipv6	A bug affecting IPv6 support	Brian Haley
l2-pop	A bug in L2 Population mech driver	Miguel Lavalle
l3-bgp	A bug affecting neutron-dynamic-routing	Tobias Urdin/ Jens Harbott
l3-dvr-backlog	A bug affecting distributed routing	Yulong Liu/ Brian Haley
l3-ha	A bug affecting L3 HA (vrrp)	Brian Haley
l3-ipam-dhcp	A bug affecting L3/DHCP/metadata	Miguel Lavalle
lib	An issue affecting neutron-lib	Neutron PTL
linuxbridge	A bug affecting ML2/linuxbridge	N/A
loadimpact	Performance penalty/improvements	Miguel Lavalle
logging	An issue with logging guidelines	Matt Riedemann
low-hanging-fruit	Starter bugs for new contributors	Miguel Lavalle
metering	A bug affecting the metering layer	N/A
needs-attention	A bug that needs further screening	PTL/Bug Deputy
opnfv	Reported by/affecting OPNFV initiative	Drivers team
ops	Reported by or affecting operators	Drivers Team
oslo	An interop/cross-project issue	Bernard Cafarelli/ Rodolfo Alonso Hernand
ovn	A bug affecting ML2/OVN	Jakub Libosvar/ Lucas Alvares Gomes

continues on next pa

Table 1 – continued from previous page

Tag	Description	Contact
ovn-octavia-provider	A bug affecting OVN Octavia provider driver	Maciej Jozefczyk/ Brian Haley
ovs	A bug affecting ML2/OVS	Miguel Lavalle
ovs-fw	A bug affecting OVS firewall	Miguel Lavalle
ovsdb-lib	A bug affecting OVSDB library	Terry Wilson
qos	A bug affecting ML2/QoS	Slawek Kaplonski
rfe	Feature enhancements being screened	Drivers Team
rfe-confirmed	Confirmed feature enhancements	Drivers Team
rfe-triaged	Triaged feature enhancements	Drivers Team
rfe-approved	Approved feature enhancements	Drivers Team
rfe-postponed	Postponed feature enhancements	Drivers Team
sg-fw	A bug affecting security groups	Brian Haley
sriov-pci-pt	A bug affecting Sriov/PCI PassThrough	Moshe Levi
tempest	A bug in tempest subtree tests	Rodolfo Alonso Hernandez
troubleshooting	An issue affecting ease of debugging	Nate Johnston
unittest	A bug affecting the unit test subtree	Rodolfo Alonso Hernandez
usability	UX, interoperability, feature parity	PTL/Drivers Team
vpnaas	A bug affecting neutron-vpnaas	Dongcan Ye
xxx-backport-potential	Cherry-pick request for stable team	Bernard Cafarelli/ Brian Haley

Access Control

- Access Control - All bugs
- Access Control - In progress

API

- API - All bugs
- API - In progress

API Reference

- API Reference - All bugs
- API Reference - In progress

Auto Allocated Topology

- Auto Allocated Topology - All bugs
- Auto Allocated Topology - In progress

Baremetal

- Baremetal - All bugs
- Baremetal - In progress

DB

- DB - All bugs
- DB - In progress

Deprecation

- Deprecation - All bugs
- DeprecationB - In progress

DNS

- DNS - All bugs
- DNS - In progress

DOC

- DOC - All bugs
- DOC - In progress

Fullstack

- Fullstack - All bugs
- Fullstack - In progress

Functional Tests

- Functional tests - All bugs
- Functional tests - In progress

FWAAS

- FWaaS - All bugs
- FWaaS - In progress

Gate Failure

- Gate failure - All bugs
- Gate failure - In progress

IPV6

- IPv6 - All bugs
- IPv6 - In progress

L2 Population

- L2 Pop - All bugs
- L2 Pop - In progress

L3 BGP

- L3 BGP - All bugs
- L3 BGP - In progress

L3 DVR Backlog

- L3 DVR - All bugs
- L3 DVR - In progress

L3 HA

- L3 HA - All bugs
- L3 HA - In progress

L3 IPAM DHCP

- L3 IPAM DHCP - All bugs
- L3 IPAM DHCP - In progress

Lib

- Lib - All bugs

LinuxBridge

- LinuxBridge - All bugs
- LinuxBridge - In progress

Load Impact

- Load Impact - All bugs
- Load Impact - In progress

Logging

- Logging - All bugs
- Logging - In progress

Low hanging fruit

- Low hanging fruit - All bugs
- Low hanging fruit - In progress

Metering

- Metering - All bugs
- Metering - In progress

Needs Attention

- Needs Attention - All bugs

OPNFV

- OPNFV - All bugs

Operators/Operations (ops)

- Ops - All bugs

OSLO

- Oslo - All bugs
- Oslo - In progress

OVN

- OVN - All bugs
- OVN - In progress

OVN Octavia Provider driver

- OVN Octavia Provider driver - All bugs
- OVN Octavia Provider driver - In progress

OVS

- OVS - All bugs
- OVS - In progress

OVS Firewall

- OVS Firewall - All bugs
- OVS Firewall - In progress

OVSDB Lib

- OVSDB Lib - All bugs
- OVSDB Lib - In progress

QoS

- QoS - All bugs
- QoS - In progress

RFE

- RFE - All bugs
- RFE - In progress

RFE-Confirmed

- RFE-Confirmed - All bugs

RFE-Triaged

- RFE-Triaged - All bugs

RFE-Approved

- RFE-Approved - All bugs
- RFE-Approved - In progress

RFE-Postponed

- RFE-Postponed - All bugs
- RFE-Postponed - In progress

SRIOV-PCI PASSTHROUGH

- SRIOV/PCI-PT - All bugs
- SRIOV/PCI-PT - In progress

SG-FW

- Security groups - All bugs
- Security groups - In progress

Tempest

- Tempest - All bugs
- Tempest - In progress

Troubleshooting

- Troubleshooting - All bugs
- Troubleshooting - In progress

Unit test

- Unit test - All bugs
- Unit test - In progress

Usability

- UX - All bugs
- UX - In progress

VPNAAS

- VPNaaS - All bugs
- VPNaaS - In progress

Backport/RC potential

List of all `Backport/RC potential` bugs for stable releases can be found on launchpad. Pointer to Launchpads page with list of such bugs for any stable release can be built by using link:

https://bugs.launchpad.net/neutron/+bugs?field.tag={STABLE_BRANCH}-backport-potential

where `STABLE_BRANCH` is always name of one of the 3 latest releases.

Contributor Onboarding

For new contributors, the following are useful onboarding information.

Contributing to Neutron

Work within Neutron is discussed on the openstack-discuss mailing list, as well as in the #openstack-neutron IRC channel. While these are great channels for engaging Neutron, the bulk of discussion of patches and code happens in gerrit itself.

With regards to gerrit, code reviews are a great way to learn about the project. There is also a list of low or wishlist priority bugs which are ideal for a new contributor to take on. If you havent done so you should setup a Neutron development environment so you can actually run the code. Devstack is the usual convenient environment to setup such an environment. See devstack.org or NeutronDevstack for more information on using Neutron with devstack.

Helping with documentation can also be a useful first step for a newcomer. Here is a list of tagged documentation and API reference bugs:

- Documentation bugs
- Api-ref bugs

IRC Information and Etiquette

The main IRC channel for Neutron is #openstack-neutron.

Neutron Team Structure

Neutron Core Reviewers

The Neutron Core Reviewer Team is responsible for many things related to Neutron. A lot of these things include mundane tasks such as the following:

- Ensuring the bug count is low
- Curating the gate and triaging failures
- Working on integrating shared code from projects such as Oslo
- Ensuring documentation is up to date and remains relevant
- Ensuring the level of testing for Neutron is adequate and remains relevant as features are added
- Helping new contributors with questions as they peel back the covers of Neutron
- Answering questions and participating in mailing list discussions
- Interfacing with other OpenStack teams and ensuring they are going in the same parallel direction
- Reviewing and merging code into the neutron tree

In essence, core reviewers share the following common ideals:

1. They share responsibility in the projects success.

2. They have made a long-term, recurring time investment to improve the project.
3. They spend their time doing what needs to be done to ensure the projects success, not necessarily what is the most interesting or fun.

A core reviewers responsibility doesnt end up with merging code. The above lists are adding context around these responsibilities.

Core Review Hierarchy

As Neutron has grown in complexity, it has become impossible for any one person to know enough to merge changes across the entire codebase. Areas of expertise have developed organically, and it is not uncommon for existing cores to defer to these experts when changes are proposed. Existing cores should be aware of the implications when they do merge changes outside the scope of their knowledge. It is with this in mind we propose a new system built around Lieutenants through a model of trust.

In order to scale development and responsibility in Neutron, we have adopted a Lieutenant system. The PTL is the leader of the Neutron project, and ultimately responsible for decisions made in the project. The PTL has designated Lieutenants in place to help run portions of the Neutron project. The Lieutenants are in charge of their own areas, and they can propose core reviewers for their areas as well. The core reviewer addition and removal polices are in place below. The Lieutenants for each system, while responsible for their area, ultimately report to the PTL. The PTL may opt to have regular one on one meetings with the lieutenants. The PTL will resolve disputes in the project that arise between areas of focus, core reviewers, and other projects. Please note Lieutenants should be leading their own area of focus, not doing all the work themselves.

As was mentioned in the previous section, a cores responsibilities do not end with merging code. They are responsible for bug triage and gate issues among other things. Lieutenants have an increased responsibility to ensure gate and bug triage for their area of focus is under control.

Neutron Lieutenants

The following are the current Neutron Lieutenants.

Area	Lieutenant	IRC nick
API	Akihiro Motoki	amotoki
DB	Nate Johnston	njohnston
Built-In Control Plane	Miguel Lavalle	mlavalle
Client	Akihiro Motoki	amotoki
Docs	Akihiro Motoki	amotoki
Infra	Rodolfo Alonso Hernandez	ralonsoh
	YAMAMOTO Takashi	yamamoto
L3	Brian Haley	haleyb
	Miguel Lavalle	mlavalle
	Yulong Liu	liuyulong
Testing	Rodolfo Alonso Hernandez	ralonsoh

Some notes on the above:

- Built-In Control Plane means the L2 agents, DHCP agents, SGs, metadata agents and ML2.
- The client includes commands installed server side.

- L3 includes the L3 agent, DVR, Dynamic routing and IPAM.
- Note these areas may change as the project evolves due to code refactoring, new feature areas, and libification of certain pieces of code.
- Infra means interactions with infra from a neutron perspective

Sub-project Lieutenants

Neutron also consists of several plugins, drivers, and agents that are developed effectively as sub-projects within Neutron in their own git repositories. Lieutenants are also named for these sub-projects to identify a clear point of contact and leader for that area. The Lieutenant is also responsible for updating the core review team for the sub-projects repositories.

Area	Lieutenant	IRC nick
networking-bgpvpn / networking-bagpipe	Lajos Katona	lajoskatona
	Thomas Morin	tmorin
neutron-dynamic-routing	Tobias Urdin	tobias-urdin
	Jens Harbott	frickler
neutron-fwaas	Nate Johnston	njohnston
neutron-vpnaas	YAMAMOTO Takashi	yamamoto
	Dongcan Ye	yedongcan
networking-midonet	Ryu Ishimoto	ryu25
	YAMAMOTO Takashi	yamamoto
networking-odl	Lajos Katona	lajoskatona
networking-ovn	Lucas Alvares Gomes	lucasagomes
networking-sfc	Dharmendra Kushwaha	dkushwaha

Existing Core Reviewers

Existing core reviewers have been reviewing code for a varying degree of cycles. With the new plan of Lieutenants and ownership, its fair to try to understand how they fit into the new model. Existing core reviewers seem to mostly focus in particular areas and are cognizant of their own strengths and weaknesses. These members may not be experts in all areas, but know their limits, and will not exceed those limits when reviewing changes outside their area of expertise. The model is built on trust, and when that trust is broken, responsibilities will be taken away.

Lieutenant Responsibilities

In the hierarchy of Neutron responsibilities, Lieutenants are expected to partake in the following additional activities compared to other core reviewers:

- Ensuring feature requests for their areas have adequate testing and documentation coverage.
- Gate triage and resolution. Lieutenants are expected to work to keep the Neutron gate running smoothly by triaging issues, filing elastic recheck queries, and closing gate bugs.
- Triaging bugs for the specific areas.

Neutron Teams

Given all of the above, Neutron has a number of core reviewer teams with responsibility over the areas of code listed below:

Neutron Core Reviewer Team

Neutron core reviewers have merge rights to the following git repositories:

- openstack/neutron
- openstack/python-neutronclient

Please note that as we adopt to the system above with core specialty in particular areas, we expect this broad core team to shrink as people naturally evolve into an area of specialization.

Core Reviewer Teams for Plugins and Drivers

The plugin decomposition effort has led to having many drivers with code in separate repositories with their own core reviewer teams. For each one of these repositories in the following repository list, there is a core team associated with it:

- Neutron project team

These teams are also responsible for handling their own specs/RFEs/features if they choose to use them. However, by choosing to be a part of the Neutron project, they submit to oversight and veto by the Neutron PTL if any issues arise.

Neutron Specs Core Reviewer Team

Neutron specs core reviewers have +2 rights to the following git repositories:

- openstack/neutron-specs

The Neutron specs core reviewer team is responsible for reviewing specs targeted to all Neutron git repositories (Neutron + Advanced Services). It is worth noting that specs reviewers have the following attributes which are potentially different than code reviewers:

- Broad understanding of cloud and networking technologies
- Broad understanding of core OpenStack projects and technologies
- An understanding of the effect approved specs have on the teams development capacity for each cycle

Specs core reviewers may match core members of the above mentioned groups, but the group can be extended to other individuals, if required.

Drivers Team

The drivers team is the group of people who have full rights to the specs repo. This team, which matches Launchpad Neutron Drivers team, is instituted to ensure a consistent architectural vision for the Neutron project, and to continue to disaggregate and share the responsibilities of the Neutron PTL. The team is in charge of reviewing and commenting on *RFEs*, and working with specification contributors to provide guidance on the process that govern contributions to the Neutron project as a whole. The team meets regularly to go over RFEs and discuss the project roadmap. Anyone is welcome to join and/or read the meeting notes.

Release Team

The release team is a group of people with some additional gerrit permissions primarily aimed at allowing release management of Neutron sub-projects. These permissions include:

- Ability to push signed tags to sub-projects whose releases are managed by the Neutron release team as opposed to the OpenStack release team.
- Ability to push merge commits for Neutron or other sub-projects.
- Ability to approve changes in all Neutron git repositories. This is required as the team needs to be able to quickly unblock things if needed, especially at release time.

Code Merge Responsibilities

While everyone is encouraged to review changes for these repositories, members of the Neutron core reviewer group have the ability to +2/-2 and +A changes to these repositories. This is an extra level of responsibility not to be taken lightly. Correctly merging code requires not only understanding the code itself, but also how the code affects things like documentation, testing, and interactions with other projects. It also means you pay attention to release milestones and understand if a patch youre merging is marked for the release, especially critical during the feature freeze.

The bottom line here is merging code is a responsibility Neutron core reviewers have.

Adding or Removing Core Reviewers

A new Neutron core reviewer may be proposed at anytime on the openstack-discuss mailing list. Typically, the Lieutenant for a given area will propose a new core reviewer for their specific area of coverage, though the Neutron PTL may propose new core reviewers as well. The proposal is typically made after discussions with existing core reviewers. Once a proposal has been made, three existing Neutron core reviewers from the Lieutenants area of focus must respond to the email with a +1. If the member is being added by a Lieutenant from an area of focus with less than three members, a simple majority will be used to determine if the vote is successful. Another Neutron core reviewer from the same area of focus can vote -1 to veto the proposed new core reviewer. The PTL will mediate all disputes for core reviewer additions.

The PTL may remove a Neutron core reviewer at any time. Typically when a member has decreased their involvement with the project through a drop in reviews and participation in general project development, the PTL will propose their removal and remove them. Please note there is no voting or vetoing of core reviewer removal. Members who have previously been a core reviewer may be fast-tracked back into a core reviewer role if their involvement picks back up and the existing core reviewers support their re-instatement.

Neutron Core Reviewer Membership Expectations

Neutron core reviewers have the following expectations:

- Reasonable attendance at the weekly Neutron IRC meetings.
- **Participation in Neutron discussions on the mailing list, as well as** in-channel in #openstack-neutron.
- Participation in Neutron related design summit sessions at the OpenStack Summits.

Please note in-person attendance at design summits, mid-cycles, and other code sprints is not a requirement to be a Neutron core reviewer. The Neutron team will do its best to facilitate virtual attendance at all events. Travel is not to be taken lightly, and we realize the costs involved for those who partake in attending these events.

In addition to the above, code reviews are the most important requirement of Neutron core reviewers. Neutron follows the documented OpenStack code review guidelines. We encourage all people to review Neutron patches, but core reviewers are required to maintain a level of review numbers relatively close to other core reviewers. There are no hard statistics around code review numbers, but in general we use 30, 60, 90 and 180 day stats when examining review stats.

- 30 day review stats
- 60 day review stats
- 90 day review stats
- 180 day review stats

There are soft-touch items around being a Neutron core reviewer as well. Gaining trust with the existing Neutron core reviewers is important. Being able to work together with the existing Neutron core reviewer team is critical as well. Being a Neutron core reviewer means spending a significant amount of time with the existing Neutron core reviewers team on IRC, the mailing list, at Summits, and in reviews. Ensuring you participate and engage here is critical to becoming and remaining a core reviewer.

Neutron Gate Failure Triage

This page provides guidelines for spotting and assessing neutron gate failures. Some hints for triaging failures are also provided.

Spotting Gate Failures

This can be achieved using several tools:

- Grafana dashboard
- logstash

For checking gate failures with logstash the following query will return failures for a specific job:

> build_status:FAILURE AND message:Finished AND build_name:check-tempest-dsvm-neutron AND build_queue:gate

And divided by the total number of jobs executed:

> message:Finished AND build_name:check-tempest-dsvm-neutron AND build_queue:gate

It will return the failure rate in the selected period for a given job. It is important to remark that failures in the check queue might be misleading as the problem causing the failure is most of the time in the patch being checked. Therefore it is always advisable to work on failures occurred in the gate queue. However, these failures are a precious resource for assessing frequency and determining root cause of failures which manifest in the gate queue.

The step above will provide a quick outlook of where things stand. When the failure rate raises above 10% for a job in 24 hours, its time to be on alert. 25% is amber alert. 33% is red alert. Anything above 50% means that probably somebody from the infra team has already a contract out on you. Whether you are relaxed, in alert mode, or freaking out because you see a red dot on your chest, it is always a good idea to check on daily bases the elastic-recheck pages.

Under the gate pipeline tab, you can see gate failure rates for already known bugs. The bugs in this page are ordered by decreasing failure rates (for the past 24 hours). If one of the bugs affecting Neutron is among those on top of that list, you should check that the corresponding bug is already assigned and somebody is working on it. If not, and there is not a good reason for that, it should be ensured somebody gets a crack at it as soon as possible. The other part of the story is to check for uncategorized failures. This is where failures for new (unknown) gate breaking bugs end up; on the other hand also infra error causing job failures end up here. It should be duty of the diligent Neutron developer to ensure the classification rate for neutron jobs is as close as possible to 100%. To this aim, the diligent Neutron developer should adopt the procedure outlined in the following sections.

Troubleshooting Tempest jobs

1. Open logs for failed jobs and look for logs/testr_results.html.gz.
2. **If that file is missing, check console.html and see where the job failed.**
    1. If there is a failure in devstack-gate-cleanup-host.txt its likely to be an infra issue.
    2. If the failure is in devstacklog.txt it could a devstack, neutron, or infra issue.
3. However, most of the time the failure is in one of the tempest tests. Take note of the error message and go to logstash.
4. On logstash, search for occurrences of this error message, and try to identify the root cause for the failure (see below).
5. File a bug for this failure, and push an *Elastic Recheck Query* for it.
6. **If you are confident with the area of this bug, and you have time, assign it to yourself; otherwise look for an**
   assignee or talk to the Neutrons bug czar to find an assignee.

Troubleshooting functional/fullstack job

1. Go to the job link provided by Jenkins CI.
2. Look at logs/testr_results.html.gz for which particular test failed.
3. More logs from a particular test are stored at logs/dsvm-functional-logs/<path_of_the_test> (or dsvm-fullstack-logs for fullstack job).
4. Find the error in the logs and search for similar errors in existing launchpad bugs. If no bugs were reported, create a new bug report. Dont forget to put a snippet of the trace into the new launchpad bug. If the log file for a particular job doesnt contain any trace, pick the one from testr_results.html.gz.
5. Create an *Elastic Recheck Query*

Advanced Troubleshooting of Gate Jobs

As a first step of troubleshooting a failing gate job, you should always check the logs of the job as described above. Unfortunately, sometimes when a tempest/functional/fullstack job is failing, it might be hard to reproduce it in a local environment, and might also be hard to understand the reason of such a failure from only reading the logs of the failed job. In such cases there are some additional ways to debug the job directly on the test node in a `live` setting.

This can be done in two ways:

1. Using the remote_pdb python module and `telnet` to directly access the python debugger while in the failed test.

   To achieve this, you need to send a `Do not merge` patch to gerrit with changes as described below:

   - Add an iptables rule to accept incoming telnet connections to remote_pdb. This can be done in one of the ansible roles used in the test job. Like for example in `neutron/roles/configure_functional_tests` file for functional tests:

     ```
     sudo iptables -I openstack-INPUT -p tcp -m state --state NEW -m␣
     ↪tcp --dport 44444 -j ACCEPT
     ```

   - Increase the `OS_TEST_TIMEOUT` value to make the test wait longer when remote_pdb is active to make debugging easier. This change can also be done in the ansible role mentioned above:

     ```
     export OS_TEST_TIMEOUT=999999
     ```

     Please note that the overall job will be limited by the job timeout, and that cannot be changed from within the job.

   - To make it easier to find the IP address of the test node, you should add to the ansible role so it prints the IPs configured on the test node. For example:

     ```
     hostname -I
     ```

   - Add the package `remote_pdb` to the `test-requirements.txt` file. That way it will be automatically installed in the venv of the test before it is run:

```
$ tail -1 test-requirements.txt
remote_pdb
```

- Finally, you need to import and call the remote_pdb module in the part of your test code where you want to start the debugger:

```
$ diff --git a/neutron/tests/fullstack/test_connectivity.py b/
↪neutron/tests/fullstack/test_connectivity.py
index c8650b0..260207b 100644
--- a/neutron/tests/fullstack/test_connectivity.py
+++ b/neutron/tests/fullstack/test_connectivity.py
@@ -189,6 +189,8 @@ class
TestLinuxBridgeConnectivitySameNetwork(BaseConnectivitySameNetworkTest):
         ]

     def test_connectivity(self):
+        import remote_pdb; remote_pdb.set_trace('0.0.0.0',␣
↪port=44444)
+
self._test_connectivity()
```

  Please note that discovery of public IP addresses is necessary because by default remote_pdb will only bind to the `127.0.0.1` IP address. Above is just an example of one of possible method, there could be other ways to do this as well.

When all the above changes are done, you must commit them and go to the Zuul status page to find the status of the tests for your `Do not merge` patch. Open the console log for your job and wait there until `remote_pdb` is started. You then need to find the IP address of the test node in the console log. This is necessary to connect via `telnet` and start debugging. It will be something like:

```
RemotePdb session open at 172.99.68.50:44444, waiting for connection .
↪..
```

An example of such a `Do not merge` patch described above can be found at https://review.opendev.org/#/c/558259/.

Please note that after adding new packages to the `requirements.txt` file, the `requirements-check` job for your test patch will fail, but it is not important for debugging.

2. If root access to the test node is necessary, for example, to check if VMs have really been spawned, or if router/dhcp namespaces have been configured properly, etc., you can ask a member of the infra-team to hold the job for troubleshooting. You can ask someone to help with that on the `openstack-infra` IRC channel. In that case, the infra-team will need to add your SSH key to the test node, and configure things so that if the job fails, the node will not be destroyed. You will then be able to SSH to it and debug things further. Please remember to tell the infra-team when you finish debugging so they can unlock and destroy the node being held.

The above two solutions can be used together. For example, you should be able to connect to the test node with both methods:

- using `remote_pdb` to connect via `telnet`;
- using `SSH` to connect as a root to the test node.

You can then ask the infra-team to add your key to the specific node on which you have already started your `remote_pdb` session.

Root Causing a Gate Failure

Time-based identification, i.e. find the naughty patch by log scavenging.

Filing An Elastic Recheck Query

The elastic recheck page has all the current open ER queries. To file one, please see the ER Wiki.

Neutron Code Reviews

Code reviews are a critical component of all OpenStack projects. Neutron accepts patches from many diverse people with diverse backgrounds, employers, and experience levels. Code reviews provide a way to enforce a level of consistency across the project, and also allow for the careful on boarding of contributions from new contributors.

Neutron Code Review Practices

Neutron follows the code review guidelines as set forth for all OpenStack projects. It is expected that all reviewers are following the guidelines set forth on that page.

In addition to that, the following rules are to follow:

- Any change that requires a new feature from Neutron runtime dependencies requires special review scrutiny to make sure such a change does not break a supported platform (examples of those platforms are latest Ubuntu LTS or CentOS). Runtime dependencies include but are not limited to: kernel, daemons and tools as defined in `oslo.rootwrap` filter files, runlevel management systems, as well as other elements of Neutron execution environment.

  Note: For some components, the list of supported platforms can be wider than usual. For example, Open vSwitch agent is expected to run successfully in Win32 runtime environment.

  1. All such changes must be tagged with `UpgradeImpact` in their commit messages.
  2. Reviewers are then advised to make an effort to check if the newly proposed runtime dependency is fulfilled on supported platforms.
  3. Specifically, reviewers and authors are advised to use existing gate and experimental platform specific jobs to validate those patches. To trigger experimental jobs, use the usual protocol (posting `check experimental` comment in Gerrit). CI will then execute and report back a baseline of Neutron tests for platforms of interest and will provide feedback on the effect of the runtime change required.
  4. If review identifies that the proposed change would break a supported platform, advise to rework the patch so that its no longer breaking the platform. One of the common ways of achieving that is gracefully falling back to alternative means on older platforms, another is hiding the new code behind a conditional, potentially controlled with a `oslo.config` option.

     Note: Neutron team retains the right to remove any platform conditionals in future releases. Platform owners are expected to accommodate in due course, or otherwise see their platforms broken. The team also retains the right to discontinue support for unresponsive platforms.

5. The change should also include a new sanity check that would help interested parties to identify their platform limitation in timely manner.

- Special attention should also be paid to changes in Neutron that can impact the Stadium and the wider family of networking-related projects (referred to as sub-projects below). These changes include:

  1. Renaming or removal of methods.
  2. Addition or removal of positional arguments.
  3. Renaming or removal of constants.

  To mitigate the risk of impacting the sub-projects with these changes, the following measures are suggested:

  1. Use of the online tool codesearch to ascertain how the proposed changes will affect the code of the sub-projects.
  2. Review the results of the non-voting check and 3rd party CI jobs executed by the sub-projects against the proposed change, which are returned by Zuul in the changes Gerrit page.

  When impacts are identified as a result of the above steps, every effort must be made to work with the affected sub-projects to resolve the issues.

- Any change that modifies or introduces a new API should have test coverage in neutron-tempest-plugin or tempest test suites. There should be at least one API test added for a new feature, but it is preferred that both API and scenario tests be added where it is appropriate.

  Scenario tests should cover not only the base level of new functionality, but also standard ways in which the functionality can be used. For example, if the feature adds a new kind of networking (like e.g. trunk ports) then tests should make sure that instances can use IPs provided by that networking, can be migrated, etc.

  It is also preferred that some negative test cases, like API tests to ensure that correct HTTP error is returned when wrong data is provided, will be added where it is appropriate.

- It is usually enough for any mechanical changes, like e.g. translation imports or imports of updated CI templates, to have only one +2 Code-Review vote to be approved. If there is any uncertainty about a specific patch, it is better to wait for review from another core reviewer before approving the patch.

Neutron Spec Review Practices

In addition to code reviews, Neutron also maintains a BP specification git repository. Detailed instructions for the use of this repository are provided here. It is expected that Neutron core team members are actively reviewing specifications which are pushed out for review to the specification repository. In addition, there is a neutron-drivers team, composed of a handful of Neutron core reviewers, who can approve and merge Neutron specs.

Some guidelines around this process are provided below:

- Once a specification has been pushed, it is expected that it will not be approved for at least 3 days after a first Neutron core reviewer has reviewed it. This allows for additional cores to review the specification.
- For blueprints which the core team deems of High or Critical importance, core reviewers may be assigned based on their subject matter expertise.
- Specification priority will be set by the PTL with review by the core team once the specification is approved.

Tracking Review Statistics

Stackalytics provides some nice interfaces to track review statistics. The links are provided below. These statistics are used to track not only Neutron core reviewer statistics, but also to track review statistics for potential future core members.

- 30 day review stats
- 60 day review stats
- 90 day review stats
- 180 day review stats

Pre-release check list

This page lists things to cover before a Neutron release and will serve as a guide for next release managers.

Server

Major release

A Major release is cut off once per development cycle and has an assigned name (Liberty, Mitaka,)

Prior to major release,

1. consider blocking all patches that are not targeted for the new release;
2. consider blocking trivial patches to keep the gate clean;
3. revise the current list of blueprints and bugs targeted for the release; roll over anything that does not fit there, or wont make it (note that no new features land in master after so called feature freeze is claimed by release team; there is a feature freeze exception (FFE) process described in release engineering documentation in more details: http://docs.openstack.org/project-team-guide/release-management.html);
4. start collecting state for targeted features from the team. For example, propose a post-mortem patch for neutron-specs as in: https://review.opendev.org/#/c/286413/
5. revise deprecation warnings collected in latest Jenkins runs: some of them may indicate a problem that should be fixed prior to release (see deprecations.txt file in those log directories); also, check whether any Launchpad bugs with the deprecation tag need a clean-up or a follow-up in the context of the release being planned;

6. check that release notes and sample configuration files render correctly, arrange clean-up if needed;
7. ensure all doc links are valid by running `tox -e linkcheck` and addressing any broken links.

New major release process contains several phases:

1. master branch is blocked for patches that are not targeted for the release;
2. the whole team is expected to work on closing remaining pieces targeted for the release;
3. once the team is ready to release the first release candidate (RC1), either PTL or one of release liaisons proposes a patch for openstack/releases repo. For example, see: https://review.opendev.org/#/c/292445/
4. once the openstack/releases patch lands, release team creates a new stable branch using hash values specified in the patch;
5. at this point, master branch is open for patches targeted to the next release; PTL unblocks all patches that were blocked in step 1;
6. if additional patches are identified that are critical for the release and must be shipped in the final major build, corresponding bugs are tagged with <release>-rc-potential in Launchpad, fixes are prepared and land in master branch, and are then backported to the newly created stable branch;
7. if patches landed in the release stable branch as per the previous step, a new release candidate that would include those patches should be requested by PTL in openstack/releases repo;
8. eventually, the latest release candidate requested by PTL becomes the final major release of the project.

Release candidate (RC) process allows for stabilization of the final release.

The following technical steps should be taken before the final release is cut off:

1. the latest alembic scripts are tagged with a milestone label. For example, see: https://review.opendev.org/#/c/288212/

In the new stable branch, you should make sure that:

1. .gitreview file points to the new branch;
2. if the branch uses constraints to manage gated dependency versions, the default constraints file name points to corresponding stable branch in openstack/requirements repo;
3. if the branch fetches any other projects as dependencies, e.g. by using tox_install.sh as an install_command in tox.ini, git repository links point to corresponding stable branches of those dependency projects.

Note that some of those steps may be covered by the OpenStack release team.

In the opened master branch, you should:

1. update CURRENT_RELEASE in neutron.db.migration.cli to point to the next release name.

While preparing the next release and even in the middle of development, its worth keeping the infrastructure clean. Consider using these tools to declutter the project infrastructure:

1. declutter Gerrit:

```
<neutron>/tools/abandon_old_reviews.sh
```

2. declutter Launchpad:

```
<release-tools>/pre_expire_bugs.py neutron --day <back-to-the-
↪beginning-of-the-release>
```

Minor release

A Minor release is created from an existing stable branch after the initial major release, and usually contains bug fixes and small improvements only. The minor release frequency should follow the release schedule for the current series. For example, assuming the current release is Rocky, stable branch releases should coincide with milestones R1, R2, R3 and the final release. Stable branches can be also released more frequently if needed, for example, if there is a major bug fix that has merged recently.

The following steps should be taken before claiming a successful minor release:

1. a patch for openstack/releases repo is proposed and merged.

Minor version number should be bumped always in cases when new release contains a patch which introduces for example:

1. new OVO version for an object,
2. new configuration option added,
3. requirement change,
4. API visible change,

The above list doesnt cover all possible cases. Those are only examples of fixes which require bump of minor version number but there can be also other types of changes requiring the same.

Changes that require the minor version number to be bumped should always have a release note added.

In other cases only patch number can be bumped.

Client

Most tips from the Server section apply to client releases too. Several things to note though:

1. when preparing for a major release, pay special attention to client bits that are targeted for the release. Global openstack/requirements freeze happens long before first RC release of server components. So if you plan to land server patches that depend on a new client, make sure you dont miss the requirements freeze. After the freeze is in action, there is no easy way to land more client patches for the planned target. All this may push an affected feature to the next development cycle.

Neutron Third-party CI

What Is Expected of Third Party CI System for Neutron

As of the Liberty summit, Neutron no longer *requires* a third-party CI, but it is strongly encouraged, as internal neutron refactoring can break external plugins and drivers at any time.

Neutron expects any Third Party CI system that interacts with gerrit to follow the requirements set by the Infrastructure team[1] as well as the Neutron Third Party CI guidelines below. Please ping the PTL in #openstack-neutron or send an email to the openstack-discuss ML (with subject [neutron]) with any questions. Be aware that the Infrastructure documentation as well as this document are living documents and undergo changes. Track changes to the infrastructure documentation using this url[2] (and please review the patches) and check this doc on a regular basis for updates.

What Changes to Run Against

If your code is a neutron plugin or driver, you should run against every neutron change submitted, except for docs, tests, tools, and top-level setup files. You can skip your CI runs for such exceptions by using `skip-if` and `all-files-match-any` directives in Zuul. You can see a programmatic example of the exceptions here[3].

If your code is in a neutron-*aas repo, you should run against the tests for that repo. You may also run against every neutron change, if your service driver is using neutron interfaces that are not provided by your service plugin (e.g. firewall/fwaas_plugin_v2.py). If you are using only plugin interfaces, it should be safe to test against only the service repo tests.

What Tests To Run

Network API tests (git link). Network scenario tests (The test_network_* tests here). Any tests written specifically for your setup. http://opendev.org/openstack/tempest/tree/tempest/api/network

Run with the test filter: network. This will include all neutron specific tests as well as any other tests that are tagged as requiring networking. An example tempest setup for devstack-gate:

```
export DEVSTACK_GATE_NEUTRON=1
export DEVSTACK_GATE_TEMPEST_REGEX='(?!.*\[.*\bslow\b.*\
↪])((network)|(neutron))'
```

Third Party CI Voting

The Neutron team encourages you to NOT vote -1 with a third-party CI. False negatives are noisy to the community, and have given -1 from third-party CIs a bad reputation. Really bad, to the point of people ignoring them all. Failure messages are useful to those doing refactors, and provide you feedback on the state of your plugin.

If you insist on voting, by default, the infra team will not allow voting by new 3rd party CI systems. The way to get your 3rd party CI system to vote is to talk with the Neutron PTL, who will let infra know the system is ready to vote. The requirements for a new system to be given voting rights are as follows:

- A new system must be up and running for a month, with a track record of voting on the sandbox system.
- A new system must correctly run and pass tests on patches for the third party driver/plugin for a month.

[1] http://ci.openstack.org/third_party.html
[2] https://review.opendev.org/#/q/status:open+project:openstack-infra/system-config+branch:master+topic:third-party,n,z
[3] https://github.com/openstack-infra/project-config/blob/master/dev/zuul/layout.yaml

- A new system must have a logfile setup and retention setup similar to the below.

Once the system has been running for a month, the owner of the third party CI system can contact the Neutron PTL to have a conversation about getting voting rights upstream.

The general process to get these voting rights is outlined here. Please follow that, taking note of the guidelines Neutron also places on voting for its CI systems.

A third party system can have its voting rights removed as well. If the system becomes unstable (stops running, voting, or start providing inaccurate results), the Neutron PTL or any core reviewer will make an attempt to contact the owner and copy the openstack-discuss mailing list. If no response is received within 2 days, the Neutron PTL will remove voting rights for the third party CI system. If a response is received, the owner will work to correct the issue. If the issue cannot be addressed in a reasonable amount of time, the voting rights will be temporarily removed.

Log & Test Results Filesystem Layout

Third-Party CI systems MUST provide logs and configuration data to help developers troubleshoot test failures. A third-party CI that DOES NOT post logs should be a candidate for removal, and new CI systems MUST post logs before they can be awarded voting privileges.

Third party CI systems should follow the filesystem layout convention of the OpenStack CI system. Please store your logs as viewable in a web browser, in a directory structure. Requiring the user to download a giant tarball is not acceptable, and will be reason to not allow your system to vote from the start, or cancel its voting rights if this changes while the system is running.

At the root of the results - there should be the following:

- console.html.gz - contains the output of stdout of the test run
- local.conf / localrc - contains the setup used for this run
- logs - contains the output of detail test log of the test run

The above logs must be a directory, which contains the following:

- Log files for each screen session that DevStack creates and launches an OpenStack component in
- Test result files
- testr_results.html.gz
- tempest.txt.gz

List of existing plugins and drivers

https://wiki.openstack.org/wiki/Neutron_Plugins_and_Drivers#Existing_Plugin_and_Drivers

References

Recheck Failed CI jobs in Neutron

This document provides guidelines on what to do in case your patch fails one of the Jenkins CI jobs. In order to discover potential bugs hidden in the code or tests themselves, its very helpful to check failed scenarios to investigate the cause of the failure. Sometimes the failure will be caused by the patch being tested, while other times the failure can be caused by a previously untracked bug. Such failures are usually related to tests that interact with a live system, like functional, fullstack and tempest jobs.

Before issuing a recheck on your patch, make sure that the gate failure is not caused by your patch. Failed job can be also caused by some infra issue, for example unable to fetch things from external resources like git or pip due to outage. Such failures outside of OpenStack world are not worth tracking in launchpad and you can recheck leaving couple of words what went wrong. Data about gate stability is collected and visualized via Grafana.

Please, do not recheck without providing the bug number for the failed job. For example, do not just put an empty recheck comment but find the related bug number and put a recheck bug ###### comment instead. If a bug does not exist yet, create one so other team members can have a look. It helps us maintain better visibility of gate failures. You can find how to troubleshoot gate failures in the *Gate Failure Triage* documentation.

14.3 Neutron Stadium

14.3.1 Neutron Stadium

This section contains information on policies and procedures for the so called Neutron Stadium. The Neutron Stadium is the list of projects that show up in the OpenStack Governance Document.

The list includes projects that the Neutron PTL and core team are directly involved in, and manage on a day to day basis. To do so, the PTL and team ensure that common practices and guidelines are followed throughout the Stadium, for all aspects that pertain software development, from inception, to coding, testing, documentation and more.

The Stadium is not to be intended as a VIP club for OpenStack networking projects, or an upper tier within OpenStack. It is simply the list of projects the Neutron team and PTL claim responsibility for when producing Neutron deliverables throughout the release cycles.

For more details on the Stadium, and what it takes for a project to be considered an integral part of the Stadium, please read on.

Stadium Governance

Background

Neutron grew to become a big monolithic codebase, and its core team had a tough time making progress on a number of fronts, like adding new features, ensuring stability, etc. During the Kilo timeframe, a decomposition effort started, where the codebase got disaggregated into separate repos, like the high level services, and the various third-party solutions for L2 and L3 services, and the Stadium was officially born.

These initiatives enabled the various individual teams in charge of the smaller projects the opportunity to iterate faster and reduce the time to feature. This has been due to the increased autonomy and implicit trust model that made the lack of oversight of the PTL and the Neutron drivers/core team acceptable for a small number of initiatives. When the proposed arrangement allowed projects to be automatically enlisted as a Neutron project based simply on description, and desire for affiliation, the number of projects included in the Stadium started to grow rapidly, which created a number of challenges for the PTL and the drivers team.

In fact, it became harder and harder to ensure consistency in the APIs, architecture, design, implementation and testing of the overarching project; all aspects of software development, like documentation, integration, release management, maintenance, and upgrades started to being neglected for some projects and that led to some unhappy experiences.

The point about uniform APIs is particularly important, because the Neutron platform is so flexible that a project can take a totally different turn in the way it exposes functionality, that it is virtually impossible for the PTL and the drivers team to ensure that good API design principles are being followed over time. In a situation where each project is on its own, that might be acceptable, but allowing independent API evolution while still under the Neutron umbrella is counterproductive.

These challenges led the Neutron team to find a better balance between autonomy and consistency and lay down criteria that more clearly identify when a project can be eligible for inclusion in the Neutron governance.

This document describes these criteria, and document the steps involved to maintain the integrity of the Stadium, and how to ensure this integrity be maintained over time when modifications to the governance are required.

When is a project considered part of the Stadium?

In order to be considered part of the Stadium, a project must show a track record of alignment with the Neutron core project. This means showing proof of adoption of practices as led by the Neutron core team. Some of these practices are typically already followed by the most mature OpenStack projects:

- Exhaustive documentation: it is expected that each project will have a *developer*, *user/operator* and API documentations available.
- Exhaustive OpenStack CI coverage: unit, functional, and tempest coverage using OpenStack CI (upstream) resources so that Grafana and OpenStack Health support is available. Access to CI resources and historical data by the team is key to ensuring stability and robustness of a project. In particular, it is of paramount importance to ensure that DB models/migrations are tested functionally to prevent data inconsistency issues or unexpected DB logic errors due to schema/models mismatch. For more details, please look at the following resources:

  - https://review.opendev.org/#/c/346091/
  - https://review.opendev.org/#/c/346272/
  - https://review.opendev.org/#/c/346083/

  More Database related information can be found on:

  - *Alembic Migrations*
  - *Neutron Database Layer*

  Bear in mind that many projects have been transitioning their codebase and tests to fully support Python 3+, and it is important that each Stadium project supports Python 3+ the same way Neutron core does. For more information on how to do testing, please refer to the *Neutron testing documentation*.

- Good release footprint, according to the chosen release model.
- Adherence to deprecation and stable backports policies.
- Demonstrated ability to do upgrades and/or rolling upgrades, where applicable. This means having grenade support on top of the CI coverage as described above.
- Client bindings and CLI developed according to the OpenStack Client plugin model.

On top of the above mentioned criteria, the following also are taken into consideration:

- A project must use, adopt and implement open software and technologies.
- A project must integrate with Neutron via one of the supported, advertised and maintained public Python APIs. REST API does not qualify (the project python-neutronclient is an exception).
- It adopts neutron-lib (with related hacking rules applied), and has proof of good decoupling from Neutron core internals.
- It provides an API that adopts API guidelines as set by the Neutron core team, and that relies on an open implementation.
- It adopts modular interfaces to provide networking services: this means that L2/7 services are provided in the form of ML2 mech drivers and service plugins respectively. A service plugin can expose a driver interface to support multiple backend technologies, and/or adopt the flavor framework as necessary.

Adding or removing projects to the Stadium

When a project is to be considered part of the Stadium, proof of compliance to the aforementioned practices will have to be demonstrated typically for at least two OpenStack releases. Application for inclusion is to be considered only within the first milestone of each OpenStack cycle, which is the time when the PTL and Neutron team do release planning, and have the most time available to discuss governance issues.

Projects part of the Neutron Stadium have typically the first milestone to get their house in order, during which time reassessment happens; if removed, because of substantial lack of meeting the criteria, a project cannot reapply within the same release cycle it has been evicted.

The process for proposing a repo into openstack/ and under the Neutron governance is to propose a patch to the openstack/governance repository. For example, to propose networking-foo, one would add the following entry under Neutron in reference/projects.yaml:

```
- repo: openstack/networking-foo
  tags:
    - name: release:independent
```

Typically this is a patch that the PTL, in collaboration with the projects point of contact, will shepherd through the review process. This step is undertaken once it is clear that all criteria are met. The next section provides an informal checklist that shows what steps a project needs to go through in order to enable the PTL and the TC to vote positively on the proposed inclusion.

Once a project is included, it abides by the Neutron *RFE submission process*, where specifications to neutron-specs are required for major API as well as major architectural changes that may require core

Neutron platform enhancements.

Checklist

- How to integrate documentation into docs.o.o: The documentation website has a section for project developer documentation. Each project in the Neutron Stadium must have an entry under the Networking Sub Projects section that points to the developer documentation for the project, available at `https://docs.openstack.org/<your-project>/latest/`. This is a two step process that involves the following:
  - Build the artefacts: this can be done by following example https://review.opendev.org/#/c/293399/.
  - Publish the artefacts: this can be done by following example https://review.opendev.org/#/c/216448/.

  More information can also be found on the project creator guide.
- How to integrate into Grafana: Grafana is a great tool that provides the ability to display historical series, like failure rates of OpenStack CI jobs. A few examples that added dashboards over time are:
  - Neutron.
  - Networking-OVN.
  - Networking-Midonet.

  Any subproject must have a Grafana dashboard that shows failure rates for at least Gate and Check queues.
- How to integrate into neutron-libs CI: there are a number of steps required to integrate with neutron-lib CI and adopt neutron-lib in general. One step is to validate that neutron-lib master is working with the master of a given project that uses neutron-lib. For example patch introduced such support for the Neutron project. Any subproject that wants to do the same would need to adopt the following few lines:
  1. https://review.opendev.org/#/c/338603/4/jenkins/jobs/projects.yaml@4685
  2. https://review.opendev.org/#/c/338603/3/zuul/layout.yaml@8501
  3. https://review.opendev.org/#/c/338603/4/grafana/neutron.yaml@39

  Line 1 and 2 respectively add a job to the periodic queue for the project, whereas line 3 introduced the failure rate trend for the periodic job to spot failure spikes etc. Make sure your project has the following:
  1. https://review.opendev.org/#/c/357086/
  2. https://review.opendev.org/#/c/359143/
- How to port api-ref over to neutron-lib: to publish the subproject API reference into the Networking API guide you must contribute the API documentation into neutron-libs api-ref directory as done in the WADL/REST transition patch. Once this is done successfully, a link to the subproject API will show under the published table of content. An RFE bug tracking this effort effectively initiates the request for Stadium inclusion, where all the aspects as outlined in this documented are reviewed by the PTL.

- How to port API definitions over the neutron-lib: the most basic steps to port API definitions over to neutron-lib are demonstrated in the following patches:
  - https://review.opendev.org/#/c/353131/
  - https://review.opendev.org/#/c/353132/

  The neutron-lib patch introduces the elements that define the API, and testing coverage validates that the resource and actions maps use valid keywords. API reference documentation is provided alongside the definition to keep everything in one place. The neutron patch uses the Neutron extension framework to plug the API definition on top of the Neutron API backbone. The change can only merge when there is a released version of neutron-lib.

- How to integrate into the openstack release: every project in the Stadium must have release notes. In order to set up release notes, please see the patches below for an example on how to set up reno:
  - https://review.opendev.org/#/c/320904/
  - https://review.opendev.org/#/c/243085/

  For release documentation related to Neutron, please check the *Neutron Policies*. Once, everything is set up and your project is released, make sure you see an entry on the release page (e.g. Pike. Make sure you release according to the project declared release model.

- How to port OpenStack Client over to python-neutronclient: client API bindings and client command line interface support must be developed in python-neutronclient under osc module. If your project requires one or both, consider looking at the following example on how to contribute these two python-neutronclient according to the OSC framework and guidelines:
  - https://review.opendev.org/#/c/340624/
  - https://review.opendev.org/#/c/340763/
  - https://review.opendev.org/#/c/352653/

  More information on how to develop python-openstackclient plugins can be found on the following links:
  - https://docs.openstack.org/python-openstackclient/latest/contributor/plugins.html
  - https://docs.openstack.org/python-openstackclient/latest/contributor/humaninterfaceguide.html

  It is worth prefixing the commands being added with the keyword network to avoid potential clash with other commands with similar names. This is only required if the command object name is highly likely to have an ambiguous meaning.

Sub-Project Guidelines

This document provides guidance for those who maintain projects that consume main neutron or neutron advanced services repositories as a dependency. It is not meant to describe projects that are not tightly coupled with Neutron code.

Code Reuse

At all times, avoid using any Neutron symbols that are explicitly marked as private (those have an underscore at the start of their names).

Try to avoid copy pasting the code from Neutron to extend it. Instead, rely on enormous number of different plugin entry points provided by Neutron (L2 agent extensions, API extensions, service plugins, core plugins, ML2 mechanism drivers, etc.)

Requirements

Neutron dependency

Subprojects usually depend on neutron repositories, by using -e https:// schema to define such a dependency. The dependency *must not* be present in requirements lists though, and instead belongs to tox.ini deps section. This is because next pbr library releases do not guarantee -e https:// dependencies will work.

You may still put some versioned neutron dependency in your requirements list to indicate the dependency for anyone who packages your subproject.

Explicit dependencies

Each neutron project maintains its own lists of requirements. Subprojects that depend on neutron while directly using some of those libraries that neutron maintains as its dependencies must not rely on the fact that neutron will pull the needed dependencies for them. Direct library usage requires that this library is mentioned in requirements lists of the subproject.

The reason to duplicate those dependencies is that neutron team does not stick to any backwards compatibility strategy in regards to requirements lists, and is free to drop any of those dependencies at any time, breaking anyone who could rely on those libraries to be pulled by neutron itself.

Automated requirements updates

At all times, subprojects that use neutron as a dependency should make sure their dependencies do not conflict with neutrons ones.

Core neutron projects maintain their requirements lists by utilizing a so-called proposal bot. To keep your subproject in sync with neutron, it is highly recommended that you register your project in openstack/requirements:projects.txt file to enable the bot to update requirements for you.

Once a subproject opts in global requirements synchronization, it should enable check-requirements jobs in project-config. For example, see this patch.

Stable branches

Stable branches for subprojects should be created at the same time when corresponding neutron stable branches are created. This is to avoid situations when a postponed cut-off results in a stable branch that contains some patches that belong to the next release. This would require reverting patches, and this is something you should avoid.

Make sure your neutron dependency uses corresponding stable branch for neutron, not master.

Note that to keep requirements in sync with core neutron repositories in stable branches, you should make sure that your project is registered in openstack/requirements:projects.txt *for the branch in question*.

Subproject stable branches are supervised by horizontal neutron-stable-maint team.

More info on stable branch process can be found on the following page.

Stable merge requirements

Merges into stable branches are handled by members of the neutron-stable-maint gerrit group. The reason for this is to ensure consistency among stable branches, and compliance with policies for stable backports.

For sub-projects who participate in the Neutron Stadium effort and who also create and utilize stable branches, there is an expectation around what is allowed to be merged in these stable branches. The Stadium projects should be following the stable branch policies as defined by on the Stable Branch wiki. This means that, among other things, no features are allowed to be backported into stable branches.

Releases

It is suggested that sub-projects cut off new releases from time to time, especially for stable branches. It will make the life of packagers and other consumers of your code easier.

Sub-Project Release Process

All subproject releases are managed by global OpenStack Release Managers team. The neutron-release team handles only the following operations:

- Make stable branches end of life

To release a sub-project, follow the following steps:

- For projects which have not moved to post-versioning, we need to push an alpha tag to avoid pbr complaining. A member of the neutron-release group will handle this.
- A sub-project owner should modify setup.cfg to remove the version (if you have one), which moves your project to post-versioning, similar to all the other Neutron projects. You can skip this step if you dont have a version in setup.cfg.
- A sub-project owner proposes a patch to openstack/releases repository with the intended git hash. The Neutron release liaison should be added in Gerrit to the list of reviewers for the patch.

Note: New major tag versions should conform to SemVer requirements, meaning no year numbers should be used as a major version. The switch to SemVer is advised at earliest convenience for all new major releases.

Note: Before Ocata, when releasing the very first release in a stable series, a sub-project owner would need to request a new stable branch creation during Gerrit review, but not anymore. See the following email for more details.

- The Neutron release liaison votes with +1 for the openstack/releases patch.
- The releases will now be on PyPI. A sub-project owner should verify this by going to an URL similar to this.
- A sub-project owner should next go to Launchpad and release this version using the Release Now button for the release itself.
- If a sub-project uses the delay-release option, a sub-project owner should update any bugs that were fixed with this release to Fix Released in Launchpad. This step is not necessary if the sub-project uses the direct-release option, which is the default.[1]
- The new release will be available on OpenStack Releases.
- A sub-project owner should add the next milestone to the Launchpad series, or if a new series is required, create the new series and a new milestone.

Note: You need to be careful when picking a git commit to base new releases on. In most cases, youll want to tag the *merge* commit that merges your last commit in to the branch. This bug shows an instance where this mistake was caught. Notice the difference between the incorrect commit and the correct one which is the merge commit. `git log 6191994..22dd683 --oneline` shows that the first one misses a handful of important commits that the second one catches. This is the nature of merging to master.

To make a branch end of life, follow the following steps:

- A member of neutron-release will abandon all open change reviews on the branch.
- A member of neutron-release will push an EOL tag on the branch. (eg. icehouse-eol)
- A sub-project owner should request the infrastructure team to delete the branch by sending an email to the infrastructure mailing list, not by bothering the infrastructure team on IRC.
- A sub-project owner should tweak jenkins jobs in project-config if any.

[1] http://lists.openstack.org/pipermail/openstack-dev/2015-December/081724.html

References

14.4 Developer Guide

In the Developer Guide, you will find information on Neutrons lower level programming APIs. There are sections that cover the core pieces of Neutron, including its database, message queue, and scheduler components. There are also subsections that describe specific plugins inside Neutron. Finally, the developer guide includes information about Neutron testing infrastructure.

14.4.1 Effective Neutron: 100 specific ways to improve your Neutron contributions

There are a number of skills that make a great Neutron developer: writing good code, reviewing effectively, listening to peer feedback, etc. The objective of this document is to describe, by means of examples, the pitfalls, the good and bad practices that we as project encounter on a daily basis and that make us either go slower or accelerate while contributing to Neutron.

By reading and collaboratively contributing to such a knowledge base, your development and review cycle becomes shorter, because you will learn (and teach to others after you) what to watch out for, and how to be proactive in order to prevent negative feedback, minimize programming errors, writing better tests, and so on and so forthin a nutshell, how to become an effective Neutron developer.

The notes below are meant to be free-form and brief by design. They are not meant to replace or duplicate OpenStack documentation, or any project-wide documentation initiative like peer-review notes or the team guide. For this reason, references are acceptable and should be favored, if the shortcut is deemed useful to expand on the distilled information. We will try to keep these notes tidy by breaking them down into sections if it makes sense. Feel free to add, adjust, remove as you see fit. Please do so, taking into consideration yourself and other Neutron developers as readers. Capture your experience during development and review and add any comment that you believe will make your life and others easier.

Happy hacking!

Developing better software

Plugin development

Document common pitfalls as well as good practices done during plugin development.

- Use mixin classes as last resort. They can be a powerful tool to add behavior but their strength is also a weakness, as they can introduce unpredictable behavior to the MRO, amongst other issues.
- In lieu of mixins, if you need to add behavior that is relevant for ML2, consider using the extension manager.
- If you make changes to the DB class methods, like calling methods that can be inherited, think about what effect that may have to plugins that have controller backends.
- If you make changes to the ML2 plugin or components used by the ML2 plugin, think about the effect that may have to other plugins.

- When adding behavior to the L2 and L3 db base classes, do not assume that there is an agent on the other side of the message broker that interacts with the server. Plugins may not rely on agents at all.
- Be mindful of required capabilities when you develop plugin extensions. The Extension description provides the ability to specify the list of required capabilities for the extension you are developing. By declaring this list, the server will not start up if the requirements are not met, thus avoiding leading the system to experience undetermined behavior at runtime.

Database interaction

Document common pitfalls as well as good practices done during database development.

- first() does not raise an exception.
- Do not use delete() to remove objects. A delete query does not load the object so no sqlalchemy events can be triggered that would do things like recalculate quotas or update revision numbers of parent objects. For more details on all of the things that can go wrong using bulk delete operations, see the Warning sections in the link above.
- For PostgreSQL if youre using GROUP BY everything in the SELECT list must be an aggregate SUM(), COUNT(), etc or used in the GROUP BY.

  The incorrect variant:

  ```
  q = query(Object.id, Object.name,
            func.count(Object.number)).group_by(Object.name)
  ```

  The correct variant:

  ```
  q = query(Object.id, Object.name,
            func.count(Object.number)).group_by(Object.id, Object.name)
  ```

- Beware of the InvalidRequestError exception. There is even a Neutron bug registered for it. Bear in mind that this error may also occur when nesting transaction blocks, and the innermost block raises an error without proper rollback. Consider if savepoints can fit your use case.
- When designing data models that are related to each other, be careful to how you model the relationships loading strategy. For instance a joined relationship can be very efficient over others (some examples include router gateways or network availability zones).
- If you add a relationship to a Neutron object that will be referenced in the majority of cases where the object is retrieved, be sure to use the lazy=joined parameter to the relationship so the related objects are loaded as part of the same query. Otherwise, the default method is select, which emits a new DB query to retrieve each related object adversely impacting performance. For example, see patch 88665 which resulted in a significant improvement since router retrieval functions always include the gateway interface.
- Conversely, do not use lazy=joined if the relationship is only used in corner cases because the JOIN statement comes at a cost that may be significant if the relationship contains many objects. For example, see patch 168214 which reduced a subnet retrieval by ~90% by avoiding a join to the IP allocation table.
- When writing extensions to existing objects (e.g. Networks), ensure that they are written in a way that the data on the object can be calculated without additional DB lookup. If thats not possible, ensure the DB lookup is performed once in bulk during a list operation. Otherwise a list call for

a 1000 objects will change from a constant small number of DB queries to 1000 DB queries. For example, see patch 257086 which changed the availability zone code from the incorrect style to a database friendly one.

- Sometimes in code we use the following structures:

```
def create():
   with context.session.begin(subtransactions=True):
       create_something()
       try:
           _do_other_thing_with_created_object()
       except Exception:
           with excutils.save_and_reraise_exception():
               delete_something()

def _do_other_thing_with_created_object():
   with context.session.begin(subtransactions=True):
       ....
```

The problem is that when exception is raised in `_do_other_thing_with_created_object` it is caught in except block, but the object cannot be deleted in except section because internal transaction from `_do_other_thing_with_created_object` has been rolled back. To avoid this nested transactions should be used. For such cases help function `safe_creation` has been created in `neutron/db/_utils.py`. So, the example above should be replaced with:

```
_safe_creation(context, create_something, delete_something,
               _do_other_thing_with_created_object)
```

Where nested transaction is used in _do_other_thing_with_created_object function.

The `_safe_creation function can also be passed the ``transaction=False` argument to prevent any transaction from being created just to leverage the automatic deletion on exception logic.

- Beware of ResultProxy.inserted_primary_key which returns a list of last inserted primary keys not the last inserted primary key:

```
result = session.execute(mymodel.insert().values(**values))
# result.inserted_primary_key is a list even if we inserted a unique␣
↪row!
```

- Beware of pymysql which can silently unwrap a list with an element (and hide a wrong use of ResultProxy.inserted_primary_key for example):

```
e.execute("create table if not exists foo (bar integer)")
e.execute(foo.insert().values(bar=1))
e.execute(foo.insert().values(bar=[2]))
```

The 2nd insert should crash (list provided, integer expected). It crashes at least with mysql and postgresql backends, but succeeds with pymysql because it transforms them into:

```
INSERT INTO foo (bar) VALUES (1)
INSERT INTO foo (bar) VALUES ((2))
```

System development

Document common pitfalls as well as good practices done when invoking system commands and interacting with linux utils.

- When a patch requires a new platform tool or a new feature in an existing tool, check if common platforms ship packages with the aforementioned feature. Also, tag such a patch with `UpgradeImpact` to raise its visibility (as these patches are brought up to the attention of the core team during team meetings). More details in *review guidelines*.
- When a patch or the code depends on a new feature in the kernel or in any platform tools (dnsmasq, ip, Open vSwitch etc.), consider introducing a new sanity check to validate deployments for the expected features. Note that sanity checks *must not* check for version numbers of underlying platform tools because distributions may decide to backport needed features into older versions. Instead, sanity checks should validate actual features by attempting to use them.

Eventlet concurrent model

Document common pitfalls as well as good practices done when using eventlet and monkey patching.

- Do not use with_lockmode(update) on SQL queries without protecting the operation with a lockutils semaphore. For some SQLAlchemy database drivers that operators may choose (e.g. MySQLdb) it may result in a temporary deadlock by yielding to another coroutine while holding the DB lock. The following wiki provides more details: https://wiki.openstack.org/wiki/OpenStack_and_SQLAlchemy#MySQLdb_.2B_eventlet_.3D_sad

Mocking and testing

Document common pitfalls as well as good practices done when writing tests, any test. For anything more elaborate, please visit the testing section.

- Preferring low level testing versus full path testing (e.g. not testing database via client calls). The former is to be favored in unit testing, whereas the latter is to be favored in functional testing.
- Prefer specific assertions (assert(Not)In, assert(Not)IsInstance, assert(Not)IsNone, etc) over generic ones (assertTrue/False, assertEqual) because they raise more meaningful errors:

```
def test_specific(self):
    self.assertIn(3, [1, 2])
    # raise meaningful error: "MismatchError: 3 not in [1, 2]"

def test_generic(self):
    self.assertTrue(3 in [1, 2])
    # raise meaningless error: "AssertionError: False is not true"
```

- Use the pattern self.assertEqual(expected, observed) not the opposite, it helps reviewers to understand which one is the expected/observed value in non-trivial assertions. The expected and observed values are also labeled in the output when the assertion fails.
- Prefer specific assertions (assertTrue, assertFalse) over assertEqual(True/False, observed).
- Dont write tests that dont test the intended code. This might seem silly but its easy to do with a lot of mocks in place. Ensure that your tests break as expected before your code change.

- Avoid heavy use of the mock library to test your code. If your code requires more than one mock to ensure that it does the correct thing, it needs to be refactored into smaller, testable units. Otherwise we depend on fullstack/tempest/api tests to test all of the real behavior and we end up with code containing way too many hidden dependencies and side effects.
- All behavior changes to fix bugs should include a test that prevents a regression. If you made a change and it didnt break a test, it means the code was not adequately tested in the first place, its not an excuse to leave it untested.
- Test the failure cases. Use a mock side effect to throw the necessary exceptions to test your except clauses.
- Dont mimic existing tests that violate these guidelines. We are attempting to replace all of these so more tests like them create more work. If you need help writing a test, reach out to the testing lieutenants and the team on IRC.
- Mocking open() is a dangerous practice because it can lead to unexpected bugs like bug 1503847. In fact, when the built-in open method is mocked during tests, some utilities (like debtcollector) may still rely on the real thing, and may end up using the mock rather what they are really looking for. If you must, consider using OpenFixture, but it is better not to mock open() at all.

Documentation

The documenation for Neutron that exists in this repository is broken down into the following directories based on content:

- doc/source/admin/ - feature-specific configuration documentation aimed at operators.
- doc/source/configuration - stubs for auto-generated configuration files. Only needs updating if new config files are added.
- doc/source/contributor/internals - developer documentation for lower-level technical details.
- doc/source/contributor/policies - neutron team policies and best practices.
- doc/source/install - install-specific documentation for standing-up network-enabled nodes.

Additional documentation resides in the neutron-lib repository:

- api-ref - API reference documentation for Neutron resource and API extensions.

Backward compatibility

Document common pitfalls as well as good practices done when extending the RPC Interfaces.

- Make yourself familiar with *Upgrade review guidelines*.

Deprecation

Sometimes we want to refactor things in a non-backward compatible way. In most cases you can use debtcollector to mark things for deprecation. Config items have deprecation options supported by oslo.config.

The deprecation process must follow the standard deprecation requirements. In terms of neutron development, this means:

- A launchpad bug to track the deprecation.
- A patch to mark the deprecated items. If the deprecation affects users (config items, API changes) then a release note must be included.
- Wait at least one cycle and at least three months linear time.
- A patch that removes the deprecated items. Make sure to refer to the original launchpad bug in the commit message of this patch.

Scalability issues

Document common pitfalls as well as good practices done when writing code that needs to process a lot of data.

Translation and logging

Document common pitfalls as well as good practices done when instrumenting your code.

- Make yourself familiar with OpenStack logging guidelines to avoid littering the logs with traces logged at inappropriate levels.
- The logger should only be passed unicode values. For example, do not pass it exceptions or other objects directly (LOG.error(exc), LOG.error(port), etc.). See https://docs.openstack.org/oslo.log/latest/user/migration.html#no-more-implicit-conversion-to-unicode-str for more details.
- Dont pass exceptions into LOG.exception: it is already implicitly included in the log message by Python logging module.
- Dont use LOG.exception when there is no exception registered in current thread context: Python 3.x versions before 3.5 are known to fail on it.

Project interfaces

Document common pitfalls as well as good practices done when writing code that is used to interface with other projects, like Keystone or Nova.

Documenting your code

Document common pitfalls as well as good practices done when writing docstrings.

Landing patches more rapidly

Scoping your patch appropriately

- Do not make multiple changes in one patch unless absolutely necessary. Cleaning up nearby functions or fixing a small bug you noticed while working on something else makes the patch very difficult to review. It also makes cherry-picking and reverting very difficult. Even apparently minor changes such as reformatting whitespace around your change can burden reviewers and cause merge conflicts.
- If a fix or feature requires code refactoring, submit the refactoring as a separate patch than the one that changes the logic. Otherwise its difficult for a reviewer to tell the difference between mistakes in the refactor and changes required for the fix/feature. If its a bug fix, try to implement the fix before the refactor to avoid making cherry-picks to stable branches difficult.
- Consider your reviewers time before submitting your patch. A patch that requires many hours or days to review will sit in the todo list until someone has many hours or days free (which may never happen.) If you can deliver your patch in small but incrementally understandable and testable pieces you will be more likely to attract reviewers.

Nits and pedantic comments

Document common nits and pedantic comments to watch out for.

- Make sure you spell correctly, the best you can, no-one wants rebase generators at the end of the release cycle!
- The odd pep8 error may cause an entire CI run to be wasted. Consider running validation (pep8 and/or tests) before submitting your patch. If you keep forgetting consider installing a git hook so that Git will do it for you.
- Sometimes, new contributors want to dip their toes with trivial patches, but we at OpenStack *love* bike shedding and their patches may sometime stall. In some extreme cases, the more trivial the patch, the higher the chances it fails to merge. To ensure we as a team provide/have a frustration-free experience new contributors should be redirected to fixing low-hanging-fruit bugs that have a tangible positive impact to the codebase. Spelling mistakes, and docstring are fine, but there is a lot more that is relatively easy to fix and has a direct impact to Neutron users.

Reviewer comments

- Acknowledge them one by one by either clicking Done or by replying extensively. If you do not, the reviewer wont know whether you thought it was not important, or you simply forgot. If the reply satisfies the reviewer, consider capturing the input in the code/document itself so that its for reviewers of newer patchsets to see (and other developers when the patch merges).
- Watch for the feedback on your patches. Acknowledge it promptly and act on it quickly, so that the reviewer remains engaged. If you disappear for a week after you posted a patchset, it is very likely that the patch will end up being neglected.
- Do not take negative feedback personally. Neutron is a large project with lots of contributors with different opinions on how things should be done. Many come from widely varying cultures and languages so the English, text-only feedback can unintentionally come across as harsh. Getting a -1 means reviewers are trying to help get the patch into a state that can be merged, it doesnt just mean they are trying to block it. Its very rare to get a patch merged on the first iteration that makes everyone happy.

Code Review

- You should visit OpenStack How To Review wiki
- Stay focussed and review what matters for the release. Please check out the Neutron section for the Gerrit dashboard. The output is generated by this tool.

IRC

- IRC is a place where you can speak with many of the Neutron developers and core reviewers. For more information you should visit OpenStack IRC wiki Neutron IRC channel is #openstack-neutron
- There are weekly IRC meetings related to many different projects/teams in Neutron. A full list of these meetings and their date/time can be found in OpenStack IRC Meetings. It is important to attend these meetings in the area of your contribution and possibly mention your work and patches.
- When you have questions regarding an idea or a specific patch of yours, it can be helpful to find a relevant person in IRC and speak with them about it. You can find a users IRC nickname in their launchpad account.
- Being available on IRC is useful, since reviewers can contact you directly to quickly clarify a review issue. This speeds up the feedback loop.
- Each area of Neutron or sub-project of Neutron has a specific lieutenant in charge of it. You can most likely find these lieutenants on IRC, it is advised however to try and send public questions to the channel rather then to a specific person if possible. (This increase the chances of getting faster answers to your questions). A list of the areas and lieutenants nicknames can be found at *Core Reviewers*.

Commit messages

Document common pitfalls as well as good practices done when writing commit messages. For more details see Git commit message best practices. This is the TL;DR version with the important points for committing to Neutron.

- One liners are bad, unless the change is trivial.
- Use `UpgradeImpact` when the change could cause issues during the upgrade from one version to the next.
- `APIImpact` should be used when the api-ref in neutron-lib must be updated to reflect the change, and only as a last resort. Rather, the ideal workflow includes submitting a corresponding neutron-lib api-ref change along with the implementation, thereby removing the need to use `APIImpact`.
- Make sure the commit message doesnt have any spelling/grammar errors. This is the first thing reviewers read and they can be distracting enough to invite -1s.
- Describe what the change accomplishes. If its a bug fix, explain how this code will fix the problem. If its part of a feature implementation, explain what component of the feature the patch implements. Do not just describe the bug, thats what launchpad is for.
- Use the Closes-Bug: #BUG-NUMBER tag if the patch addresses a bug. Submitting a bugfix without a launchpad bug reference is unacceptable, even if its trivial. Launchpad is how bugs are tracked so fixes without a launchpad bug are a nightmare when users report the bug from an older version and the Neutron team cant tell if/why/how its been fixed. Launchpad is also how backports are identified and tracked so patches without a bug report cannot be picked to stable branches.
- Use the Implements: blueprint NAME-OF-BLUEPRINT or Partially-Implements: blueprint NAME-OF-BLUEPRINT for features so reviewers can determine if the code matches the spec that was agreed upon. This also updates the blueprint on launchpad so its easy to see all patches that are related to a feature.
- If its not immediately obvious, explain what the previous code was doing that was incorrect. (e.g. code assumed it would never get None from a function call)
- Be specific in your commit message about what the patch does and why it does this. For example, Fixes incorrect logic in security groups is not helpful because the code diff already shows that you are modifying security groups. The message should be specific enough that a reviewer looking at the code can tell if the patch does what the commit says in the most appropriate manner. If the reviewer has to guess why you did something, lots of your time will be wasted explaining why certain changes were made.

Dealing with Zuul

Document common pitfalls as well as good practices done when dealing with OpenStack CI.

- When you submit a patch, consider checking its status in the queue. If you see a job failures, you might as well save time and try to figure out in advance why it is failing.
- Excessive use of recheck to get test to pass is discouraged. Please examine the logs for the failing test(s) and make sure your change has not tickled anything that might be causing a new failure or race condition. Getting your change in could make it even harder to debug what is actually broken later on.

14.4.2 Setting Up a Development Environment

This page describes how to setup a working Python development environment that can be used in developing Neutron on Ubuntu, Fedora or Mac OS X. These instructions assume youre already familiar with Git and Gerrit, which is a code repository mirror and code review toolset , however if you arent please see this Git tutorial for an introduction to using Git and this guide for a tutorial on using Gerrit and Git for code contribution to OpenStack projects.

Following these instructions will allow you to run the Neutron unit tests. If you want to be able to run Neutron in a full OpenStack environment, you can use the excellent DevStack project to do so. There is a wiki page that describes setting up Neutron using DevStack.

Getting the code

Grab the code:

```
git clone https://opendev.org/openstack/neutron.git
cd neutron
```

About ignore files

In the .gitignore files, add patterns to exclude files created by tools integrated, such as test frameworks from the projects recommended workflow, rendered documentation and package builds.

Dont add patterns to exculde files created by preferred personal like for example editors, IDEs or operating system. These should instead be maintained outside the repository, for example in a ~/.gitignore file added with:

```
git config --global core.excludesfile '~/.gitignore'
```

Ignores files for all repositories that you work with.

Testing Neutron

See *Testing Neutron*.

14.4.3 Deploying a development environment with vagrant

The vagrant directory contains a set of vagrant configurations which will help you deploy Neutron with ovn driver for testing or development purposes.

We provide a sparse multinode architecture with clear separation between services. In the future we will include all-in-one and multi-gateway architectures.

Vagrant prerequisites

Those are the prerequisites for using the vagrant file definitions

1. Install VirtualBox and Vagrant. Alternatively you can use parallels or libvirt vagrant plugin.
2. Install plug-ins for Vagrant:

```
$ vagrant plugin install vagrant-cachier
$ vagrant plugin install vagrant-vbguest
```

3. On Linux hosts, you can enable instances to access external networks such as the Internet by enabling IP forwarding and configuring SNAT from the IP address range of the provider network interface (typically vboxnet1) on the host to the external network interface on the host. For example, if the `eth0` network interface on the host provides external network connectivity:

```
# sysctl -w net.ipv4.ip_forward=1
# sysctl -p
# iptables -t nat -A POSTROUTING -s 10.10.0.0/16 -o eth0 -j MASQUERADE
```

   Note: These commands do not persist after rebooting the host.

Sparse architecture

The Vagrant scripts deploy OpenStack with Open Virtual Network (OVN) using four nodes (five if you use the optional ovn-vtep node) to implement a minimal variant of the reference architecture:

1. ovn-db: Database node containing the OVN northbound (NB) and southbound (SB) databases via the Open vSwitch (OVS) database and `ovn-northd` services.
2. ovn-controller: Controller node containing the Identity service, Image service, control plane portion of the Compute service, control plane portion of the Networking service including the `ovn` ML2 driver, and the dashboard. In addition, the controller node is configured as an NFS server to support instance live migration between the two compute nodes.
3. ovn-compute1 and ovn-compute2: Two compute nodes containing the Compute hypervisor, `ovn-controller` service for OVN, metadata agents for the Networking service, and OVS services. In addition, the compute nodes are configured as NFS clients to support instance live migration between them.
4. ovn-vtep: Optional. A node to run the HW VTEP simulator. This node is not started by default but can be started by running vagrant up ovn-vtep after doing a normal vagrant up.

During deployment, Vagrant creates three VirtualBox networks:

1. Vagrant management network for deployment and VM access to external networks such as the Internet. Becomes the VM `eth0` network interface.
2. OpenStack management network for the OpenStack control plane, OVN control plane, and OVN overlay networks. Becomes the VM `eth1` network interface.
3. OVN provider network that connects OpenStack instances to external networks such as the Internet. Becomes the VM `eth2` network interface.

Requirements

The default configuration requires approximately 12 GB of RAM and supports launching approximately four OpenStack instances using the `m1.tiny` flavor. You can change the amount of resources for each VM in the `instances.yml` file.

Deployment

1. Follow the pre-requisites described in *Vagrant prerequisites*
2. Clone the `neutron` repository locally and change to the `neutron/vagrant/ovn/sparse` directory:

```
$ git clone https://opendev.org/openstack/neutron.git
$ cd neutron/vagrant/ovn/sparse
```

3. If necessary, adjust any configuration in the `instances.yml` file.
   - If you change any IP addresses or networks, avoid conflicts with the host.
   - For evaluating large MTUs, adjust the `mtu` option. You must also change the MTU on the equivalent `vboxnet` interfaces on the host to the same value after Vagrant creates them. For example:

   ```
   # ip link set dev vboxnet0 mtu 9000
   # ip link set dev vboxnet1 mtu 9000
   ```

4. Launch the VMs and grab some coffee:

```
$ vagrant up
```

5. After the process completes, you can use the `vagrant status` command to determine the VM status:

```
$ vagrant status
Current machine states:

ovn-db                    running (virtualbox)
ovn-controller            running (virtualbox)
ovn-vtep                  running (virtualbox)
ovn-compute1              running (virtualbox)
ovn-compute2              running (virtualbox)
```

6. You can access the VMs using the following commands:

```
$ vagrant ssh ovn-db
$ vagrant ssh ovn-controller
$ vagrant ssh ovn-vtep
$ vagrant ssh ovn-compute1
$ vagrant ssh ovn-compute2
```

Note: If you prefer to use the VM console, the password for the `root` account is `vagrant`. Since ovn-controller is set as the primary in the Vagrantfile, the command `vagrant ssh` (without specifying the name) will connect ssh to that virtual machine.

7. Access OpenStack services via command-line tools on the `ovn-controller` node or via the dashboard from the host by pointing a web browser at the IP address of the `ovn-controller` node.

   Note: By default, OpenStack includes two accounts: `admin` and `demo`, both using password `password`.

8. After completing your tasks, you can destroy the VMs:

```
$ vagrant destroy
```

14.4.4 Contributing new extensions to Neutron

Introduction

Neutron has a pluggable architecture, with a number of extension points. This documentation covers aspects relevant to contributing new Neutron v2 core (aka monolithic) plugins, ML2 mechanism drivers, and L3 service plugins. This document will initially cover a number of process-oriented aspects of the contribution process, and proceed to provide a how-to guide that shows how to go from 0 LOCs to successfully contributing new extensions to Neutron. In the remainder of this guide, we will try to use practical examples as much as we can so that people have working solutions they can start from.

This guide is for a developer who wants to have a degree of visibility within the OpenStack Networking project. If you are a developer who wants to provide a Neutron-based solution without interacting with the Neutron community, you are free to do so, but you can stop reading now, as this guide is not for you.

Plugins and drivers for non-reference implementations are known as third-party code. This includes code for supporting vendor products, as well as code for supporting open-source networking implementations.

Before the Kilo release these plugins and drivers were included in the Neutron tree. During the Kilo cycle the third-party plugins and drivers underwent the first phase of a process called decomposition. During this phase, each plugin and driver moved the bulk of its logic to a separate git repository, while leaving a thin shim in the neutron tree together with the DB models and migrations (and perhaps some config examples).

During the Liberty cycle the decomposition concept was taken to its conclusion by allowing third-party code to exist entirely out of tree. Further extension mechanisms have been provided to better support external plugins and drivers that alter the API and/or the data model.

In the Mitaka cycle we will **require** all third-party code to be moved out of the neutron tree completely.

Outside the tree can be anything that is publicly available: it may be a repo on opendev.org for instance, a tarball, a pypi package, etc. A plugin/drivers maintainer team self-governs in order to promote sharing, reuse, innovation, and release of the out-of-tree deliverable. It should not be required for any member of the core team to be involved with this process, although core members of the Neutron team can participate in whichever capacity is deemed necessary to facilitate out-of-tree development.

This guide is aimed at you as the maintainer of code that integrates with Neutron but resides in a separate repository.

Contribution Process

If you want to extend OpenStack Networking with your technology, and you want to do it within the visibility of the OpenStack project, follow the guidelines and examples below. Well describe best practices for:

- Design and Development;
- Testing and Continuous Integration;
- Defect Management;
- Backport Management for plugin specific code;
- DevStack Integration;
- Documentation;

Once you have everything in place you may want to add your project to the list of Neutron sub-projects. See *Adding or removing projects to the Stadium* for details.

Design and Development

Assuming you have a working repository, any development to your own repo does not need any blueprint, specification or bugs against Neutron. However, if your project is a part of the Neutron Stadium effort, you are expected to participate in the principles of the Four Opens, meaning your design should be done in the open. Thus, it is encouraged to file documentation for changes in your own repository.

If your code is hosted on opendev.org then the gerrit review system is automatically provided. Contributors should follow the review guidelines similar to those of Neutron. However, you as the maintainer have the flexibility to choose who can approve/merge changes in your own repo.

It is recommended (but not required, see *policies*) that you set up a third-party CI system. This will provide a vehicle for checking the third-party code against Neutron changes. See *Testing and Continuous Integration* below for more detailed recommendations.

Design documents can still be supplied in form of Restructured Text (RST) documents, within the same third-party library repo. If changes to the common Neutron code are required, an *RFE* may need to be filed. However, every case is different and you are invited to seek guidance from Neutron core reviewers about what steps to follow.

Testing and Continuous Integration

The following strategies are recommendations only, since third-party CI testing is not an enforced requirement. However, these strategies are employed by the majority of the plugin/driver contributors that actively participate in the Neutron development community, since they have learned from experience how quickly their code can fall out of sync with the rapidly changing Neutron core code base.

- You should run unit tests in your own external library (e.g. on opendev.org where Jenkins setup is for free).
- Your third-party CI should validate third-party integration with Neutron via functional testing. The third-party CI is a communication mechanism. The objective of this mechanism is as follows:

- it communicates to you when someone has contributed a change that potentially breaks your code. It is then up to you maintaining the affected plugin/driver to determine whether the failure is transient or real, and resolve the problem if it is.
- it communicates to a patch author that they may be breaking a plugin/driver. If they have the time/energy/relationship with the maintainer of the plugin/driver in question, then they can (at their discretion) work to resolve the breakage.
- it communicates to the community at large whether a given plugin/driver is being actively maintained.
- A maintainer that is perceived to be responsive to failures in their third-party CI jobs is likely to generate community goodwill.

It is worth noting that if the plugin/driver repository is hosted on opendev.org, due to current openstack-infra limitations, it is not possible to have third-party CI systems participating in the gate pipeline for the repo. This means that the only validation provided during the merge process to the repo is through unit tests. Post-merge hooks can still be exploited to provide third-party CI feedback, and alert you of potential issues. As mentioned above, third-party CI systems will continue to validate Neutron core commits. This will allow them to detect when incompatible changes occur, whether they are in Neutron or in the third-party repo.

Defect Management

Bugs affecting third-party code should *not* be filed in the Neutron project on launchpad. Bug tracking can be done in any system you choose, but by creating a third-party project in launchpad, bugs that affect both Neutron and your code can be more easily tracked using launchpads also affects project feature.

Security Issues

Here are some answers to how to handle security issues in your repo, taken from this mailing list message:

- How should security your issues be managed?

The OpenStack Vulnerability Management Team (VMT) follows a documented process which can basically be reused by any project-team when needed.

- Should the OpenStack security team be involved?

The OpenStack VMT directly oversees vulnerability reporting and disclosure for a subset of OpenStack source code repositories. However, they are still quite happy to answer any questions you might have about vulnerability management for your own projects even if theyre not part of that set. Feel free to reach out to the VMT in public or in private.

Also, the VMT is an autonomous subgroup of the much larger OpenStack Security project-team. Theyre a knowledgeable bunch and quite responsive if you want to get their opinions or help with security-related issues (vulnerabilities or otherwise).

- Does a CVE need to be filed?

It can vary widely. If a commercial distribution such as Red Hat is redistributing a vulnerable version of your software, then they may assign one anyway even if you dont request one yourself. Or the reporter may request one; the reporter may even be affiliated with an organization who has already assigned/obtained a CVE before they initiate contact with you.

- Do the maintainers need to publish OSSN or equivalent documents?

OpenStack Security Advisories (OSSA) are official publications of the OpenStack VMT and only cover VMT-supported software. OpenStack Security Notes (OSSN) are published by editors within the OpenStack Security project-team on more general security topics and may even cover issues in non-OpenStack software commonly used in conjunction with OpenStack, so its at their discretion as to whether they would be able to accommodate a particular issue with an OSSN.

However, these are all fairly arbitrary labels, and what really matters in the grand scheme of things is that vulnerabilities are handled seriously, fixed with due urgency and care, and announced widely not just on relevant OpenStack mailing lists but also preferably somewhere with broader distribution like the Open Source Security mailing list. The goal is to get information on your vulnerabilities, mitigating measures and fixes into the hands of the people using your software in a timely manner.

- Anything else to consider here?

The OpenStack VMT is in the process of trying to reinvent itself so that it can better scale within the context of the Big Tent. This includes making sure the policy/process documentation is more consumable and reusable even by project-teams working on software outside the scope of our charter. Its a work in progress, and any input is welcome on how we can make this function well for everyone.

Backport Management Strategies

This section applies only to third-party maintainers who had code in the Neutron tree during the Kilo and earlier releases. It will be obsolete once the Kilo release is no longer supported.

If a change made to out-of-tree third-party code needs to be back-ported to in-tree code in a stable branch, you may submit a review without a corresponding master branch change. The change will be evaluated by core reviewers for stable branches to ensure that the backport is justified and that it does not affect Neutron core code stability.

DevStack Integration Strategies

When developing and testing a new or existing plugin or driver, the aid provided by DevStack is incredibly valuable: DevStack can help get all the software bits installed, and configured correctly, and more importantly in a predictable way. For DevStack integration there are a few options available, and they may or may not make sense depending on whether you are contributing a new or existing plugin or driver.

If you are contributing a new plugin, the approach to choose should be based on Extras.d Hooks externally hosted plugins. With the extra.d hooks, the DevStack integration is co-located with the third-party integration library, and it leads to the greatest level of flexibility when dealing with DevStack based dev/test deployments.

One final consideration is worth making for third-party CI setups: if Devstack Gate is used, it does provide hook functions that can be executed at specific times of the devstack-gate-wrap script run. For example, the Neutron Functional job uses them. For more details see devstack-vm-gate-wrap.sh.

Documentation

For a layout of the how the documentation directory is structured see the effective neutron guide

Project Initial Setup

The how-to below assumes that the third-party library will be hosted on opendev.org. This lets you tap in the entire OpenStack CI infrastructure and can be a great place to start from to contribute your new or existing driver/plugin. The list of steps below are summarized version of what you can find on http://docs.openstack.org/infra/manual/creators.html. They are meant to be the bare minimum you have to complete in order to get you off the ground.

- Create a public repository: this can be a personal opendev.org repo or any publicly available git repo, e.g. `https://github.com/john-doe/foo.git`. This would be a temporary buffer to be used to feed the one on opendev.org.
- Initialize the repository: if you are starting afresh, you may *optionally* want to use cookiecutter to get a skeleton project. You can learn how to use cookiecutter on https://opendev.org/openstack-dev/cookiecutter. If you want to build the repository from an existing Neutron module, you may want to skip this step now, build the history first (next step), and come back here to initialize the remainder of the repository with other files being generated by the cookiecutter (like tox.ini, setup.cfg, setup.py, etc.).
- Create a repository on opendev.org. For this you need the help of the OpenStack infra team. It is worth noting that you only get one shot at creating the repository on opendev.org. This is the time you get to choose whether you want to start from a clean slate, or you want to import the repo created during the previous step. In the latter case, you can do so by specifying the upstream section for your project in project-config/gerrit/project.yaml. Steps are documented on the Repository Creators Guide.
- Ask for a Launchpad user to be assigned to the core team created. Steps are documented in this section.
- Fix, fix, fix: at this point you have an external base to work on. You can develop against the new opendev.org project, the same way you work with any other OpenStack project: you have pep8, docs, and python CI jobs that validate your patches when posted to Gerrit. For instance, one thing you would need to do is to define an entry point for your plugin or driver in your own setup.cfg similarly as to how it is done in the setup.cfg for ODL.
- Define an entry point for your plugin or driver in setup.cfg
- Create third-party CI account: if you do not already have one, follow instructions for third-party CI to get one.

Internationalization support

OpenStack is committed to broad international support. Internationalization (I18n) is one of important areas to make OpenStack ubiquitous. Each project is recommended to support i18n.

This section describes how to set up translation support. The description in this section uses the following variables:

- repository : `openstack/${REPOSITORY}` (e.g., `openstack/networking-foo`)
- top level python path : `${MODULE_NAME}` (e.g., `networking_foo`)

oslo.i18n

- Each subproject repository should have its own oslo.i18n integration wrapper module `${MODULE_NAME}/_i18n.py`. The detail is found at https://docs.openstack.org/oslo.i18n/latest/user/usage.html.

  Note: **DOMAIN** name should match your **module** name `${MODULE_NAME}`.

- Import `_()` from your `${MODULE_NAME}/_i18n.py`.

  Warning: Do not use `_()` in the builtins namespace which is registered by **gettext.install()** in `neutron/__init__.py`. It is now deprecated as described in oslo.18n documentation.

Setting up translation support

You need to create or edit the following files to start translation support:

- setup.cfg
- babel.cfg

We have a good example for an oslo project at https://review.opendev.org/#/c/98248/.

Add the following to `setup.cfg`:

```
[extract_messages]
keywords = _ gettext ngettext l_ lazy_gettext
mapping_file = babel.cfg
output_file = ${MODULE_NAME}/locale/${MODULE_NAME}.pot

[compile_catalog]
directory = ${MODULE_NAME}/locale
domain = ${MODULE_NAME}

[update_catalog]
domain = ${MODULE_NAME}
output_dir = ${MODULE_NAME}/locale
input_file = ${MODULE_NAME}/locale/${MODULE_NAME}.pot
```

Note that `${MODULE_NAME}` is used in all names.

Create `babel.cfg` with the following contents:

```
[python: **.py]
```

Enable Translation

To update and import translations, you need to make a change in project-config. A good example is found at https://review.opendev.org/#/c/224222/. After doing this, the necessary jobs will be run and push/pull a message catalog to/from the translation infrastructure.

Integrating with the Neutron system

Configuration Files

The `data_files` in the `[files]` section of `setup.cfg` of Neutron shall not contain any third-party references. These shall be located in the same section of the third-party repos own `setup.cfg` file.

- Note: Care should be taken when naming sections in configuration files. When the Neutron service or an agent starts, oslo.config loads sections from all specified config files. This means that if a section [foo] exists in multiple config files, duplicate settings will collide. It is therefore recommended to prefix section names with a third-party string, e.g. [vendor_foo].

Since Mitaka, configuration files are not maintained in the git repository but should be generated as follows:

```
``tox -e genconfig``
```

If a tox environment is unavailable, then you can run the following script instead to generate the configuration files:

```
./tools/generate_config_file_samples.sh
```

It is advised that subprojects do not keep their configuration files in their respective trees and instead generate them using a similar approach as Neutron does.

ToDo: Inclusion in OpenStack documentation? Is there a recommended way to have third-party config options listed in the configuration guide in docs.openstack.org?

Database Models and Migrations

A third-party repo may contain database models for its own tables. Although these tables are in the Neutron database, they are independently managed entirely within the third-party code. Third-party code shall **never** modify neutron core tables in any way.

Each repo has its own *expand* and *contract* alembic migration branches. A third-party repos alembic migration branches may operate only on tables that are owned by the repo.

- Note: Care should be taken when adding new tables. To prevent collision of table names it is **required** to prefix them with a vendor/plugin string.
- Note: A third-party maintainer may opt to use a separate database for their tables. This may complicate cases where there are foreign key constraints across schemas for DBMS that do not support this well. Third-party maintainer discretion advised.

The database tables owned by a third-party repo can have references to fields in neutron core tables. However, the alembic branch for a plugin/driver repo shall never update any part of a table that it does not own.

Note: What happens when a referenced item changes?

- **Q:** If a drivers table has a reference (for example a foreign key) to a neutron core table, and the referenced item is changed in neutron, what should you do?
- **A:** Fortunately, this should be an extremely rare occurrence. Neutron core reviewers will not allow such a change unless there is a very carefully thought-out design decision behind it. That design will include how to address any third-party code affected. (This is another good reason why you should stay actively involved with the Neutron developer community.)

The `neutron-db-manage` alembic wrapper script for neutron detects alembic branches for installed third-party repos, and the upgrade command automatically applies to all of them. A third-party repo must register its alembic migrations at installation time. This is done by providing an entrypoint in setup.cfg as follows:

For a third-party repo named `networking-foo`, add the alembic_migrations directory as an entrypoint in the `neutron.db.alembic_migrations` group:

```
[entry_points]
neutron.db.alembic_migrations =
    networking-foo = networking_foo.db.migration:alembic_migrations
```

ToDo: neutron-db-manage autogenerate The alembic autogenerate command needs to support branches in external repos. Bug #1471333 has been filed for this.

DB Model/Migration Testing

Here is a *template functional test* third-party maintainers can use to develop tests for model-vs-migration sync in their repos. It is recommended that each third-party CI sets up such a test, and runs it regularly against Neutron master.

Entry Points

The Python setuptools installs all entry points for packages in one global namespace for an environment. Thus each third-party repo can define its packages own `[entry_points]` in its own `setup.cfg` file.

For example, for the `networking-foo` repo:

```
[entry_points]
console_scripts =
    neutron-foo-agent = networking_foo.cmd.eventlet.agents.foo:main
neutron.core_plugins =
    foo_monolithic = networking_foo.plugins.monolithic.plugin:FooPluginV2
neutron.service_plugins =
    foo_l3 = networking_foo.services.l3_router.l3_foo:FooL3ServicePlugin
neutron.ml2.type_drivers =
    foo_type = networking_foo.plugins.ml2.drivers.foo:FooType
neutron.ml2.mechanism_drivers =
    foo_ml2 = networking_foo.plugins.ml2.drivers.foo:FooDriver
```

(continues on next page)

(continued from previous page)

```
neutron.ml2.extension_drivers =
    foo_ext = networking_foo.plugins.ml2.drivers.foo:FooExtensionDriver
```

- Note: It is advisable to include `foo` in the names of these entry points to avoid conflicts with other third-party packages that may get installed in the same environment.

API Extensions

Extensions can be loaded in two ways:

1. Use the `append_api_extensions_path()` library API. This method is defined in `neutron/api/extensions.py` in the neutron tree.
2. Leverage the `api_extensions_path` config variable when deploying. See the example config file `etc/neutron.conf` in the neutron tree where this variable is commented.

Service Providers

If your project uses service provider(s) the same way VPNAAS does, you specify your service provider in your `project_name.conf` file like so:

```
[service_providers]
# Must be in form:
# service_provider=<service_type>:<name>:<driver>[:default][,...]
```

In order for Neutron to load this correctly, make sure you do the following in your code:

```
from neutron.db import servicetype_db
service_type_manager = servicetype_db.ServiceTypeManager.get_instance()
service_type_manager.add_provider_configuration(
    YOUR_SERVICE_TYPE,
    pconf.ProviderConfiguration(YOUR_SERVICE_MODULE, YOUR_SERVICE_TYPE))
```

This is typically required when you instantiate your service plugin class.

Interface Drivers

Interface (VIF) drivers for the reference implementations are defined in `neutron/agent/linux/interface.py`. Third-party interface drivers shall be defined in a similar location within their own repo.

The entry point for the interface driver is a Neutron config option. It is up to the installer to configure this item in the `[default]` section. For example:

```
[default]
interface_driver = networking_foo.agent.linux.interface.FooInterfaceDriver
```

ToDo: Interface Driver port bindings. `VIF_TYPE_*` constants in `neutron_lib/api/definitions/portbindings.py` should be moved from neutron core to the repositories where their drivers are implemented. We need to provide some config or hook mechanism for

VIF types to be registered by external interface drivers. For Nova, selecting the VIF driver can be done outside of Neutron (using the new os-vif python library?). Armando and Akihiro to discuss.

Rootwrap Filters

If a third-party repo needs a rootwrap filter for a command that is not used by Neutron core, then the filter shall be defined in the third-party repo.

For example, to add a rootwrap filters for commands in repo `networking-foo`:

- In the repo, create the file: `etc/neutron/rootwrap.d/foo.filters`
- In the repos `setup.cfg` add the filters to data_files:

```
[files]
data_files =
    etc/neutron/rootwrap.d =
        etc/neutron/rootwrap.d/foo.filters
```

Extending python-neutronclient

The maintainer of a third-party component may wish to add extensions to the Neutron CLI client. Thanks to https://review.opendev.org/148318 this can now be accomplished. See Client Command Extensions.

Other repo-split items

(These are still TBD.)

- Splitting policy.json? **ToDo** Armando will investigate.
- Generic instructions (or a template) for installing an out-of-tree plugin or driver for Neutron. Possibly something for the networking guide, and/or a template that plugin/driver maintainers can modify and include with their package.

14.4.5 Neutron public API

Neutron main tree serves as a library for multiple subprojects that rely on different modules from neutron.* namespace to accommodate their needs. Specifically, advanced service repositories and open source or vendor plugin/driver repositories do it.

Neutron modules differ in their API stability a lot, and there is no part of it that is explicitly marked to be consumed by other projects.

That said, there are modules that other projects should definitely avoid relying on.

Breakages

Neutron API is not very stable, and there are cases when a desired change in neutron tree is expected to trigger breakage for one or more external repositories under the neutron tent. Below you can find a list of known incompatible changes that could or are known to trigger those breakages. The changes are listed in reverse chronological order (newer at the top).

- change: QoS plugin refactor
  - commit: I863f063a0cfbb464cedd00bddc15dd853cbb6389
  - **solution: implement the new abstract methods in** neutron.extensions.qos.QoSPluginBase.
  - severity: Low (some out-of-tree plugins might be affected).
- change: Consume ConfigurableMiddleware from oslo_middleware.
  - commit: If7360608f94625b7d0972267b763f3e7d7624fee
  - **solution: switch to oslo_middleware.base.ConfigurableMiddleware;** stop using neutron.wsgi.Middleware and neutron.wsgi.Debug.
  - severity: Low (some out-of-tree plugins might be affected).
- change: Consume sslutils and wsgi modules from oslo.service.
  - commit: Ibfdf07e665fcfcd093a0e31274e1a6116706aec2
  - solution: switch using oslo_service.wsgi.Router; stop using neutron.wsgi.Router.
  - severity: Low (some out-of-tree plugins might be affected).
- change: oslo.service adopted.
  - commit: 6e693fc91dd79cfbf181e3b015a1816d985ad02c
  - solution: switch using oslo_service.* namespace; stop using ANY neutron.openstack.* contents.
  - severity: low (plugins must not rely on that subtree).
- change: oslo.utils.fileutils adopted.
  - commit: I933d02aa48260069149d16caed02b020296b943a
  - solution: switch using oslo_utils.fileutils module; stop using neutron.openstack.fileutils module.
  - severity: low (plugins must not rely on that subtree).
- change: Reuse callers session in DB methods.
  - commit: 47dd65cf986d712e9c6ca5dcf4420dfc44900b66
  - solution: Add context to args and reuse.
  - severity: High (mostly undetected, because 3rd party CI run Tempest tests only).
- change: switches to oslo.log, removes neutron.openstack.common.log.
  - commit: 22328baf1f60719fcaa5b0fbd91c0a3158d09c31
  - solution: a) switch to oslo.log; b) copy log module into your tree and use it (may not work due to conflicts between the module and oslo.log configuration options).

  - severity: High (most CI systems are affected).

- change: Implements reorganize-unit-test-tree spec.
  - commit: 1105782e3914f601b8f4be64939816b1afe8fb54
  - solution: Code affected need to update existing unit tests to reflect new locations.
  - severity: High (mostly undetected, because 3rd party CI run Tempest tests only).
- change: drop linux/ovs_lib compat layer.
  - commit: 3bbf473b49457c4afbfc23fd9f59be8aa08a257d
  - solution: switch to using neutron/agent/common/ovs_lib.py.
  - severity: High (most CI systems are affected).

14.4.6 Client command extension support

The client command extension adds support for extending the neutron client while considering ease of creation.

The full document can be found in the python-neutronclient repository: https://docs.openstack.org/python-neutronclient/latest/contributor/client_command_extensions.html

14.4.7 Alembic Migrations

Introduction

The migrations in the alembic/versions contain the changes needed to migrate from older Neutron releases to newer versions. A migration occurs by executing a script that details the changes needed to upgrade the database. The migration scripts are ordered so that multiple scripts can run sequentially to update the database.

The Migration Wrapper

The scripts are executed by Neutrons migration wrapper `neutron-db-manage` which uses the Alembic library to manage the migration. Pass the `--help` option to the wrapper for usage information.

The wrapper takes some options followed by some commands:

```
neutron-db-manage <options> <commands>
```

The wrapper needs to be provided with the database connection string, which is usually provided in the `neutron.conf` configuration file in an installation. The wrapper automatically reads from `/etc/neutron/neutron.conf` if it is present. If the configuration is in a different location:

```
neutron-db-manage --config-file /path/to/neutron.conf <commands>
```

Multiple `--config-file` options can be passed if needed.

Instead of reading the DB connection from the configuration file(s) the `--database-connection` option can be used:

```
neutron-db-manage --database-connection mysql+pymysql://root:secret@127.0.
↪0.1/neutron?charset=utf8 <commands>
```

The `branches`, `current`, and `history` commands all accept a `--verbose` option, which, when passed, will instruct `neutron-db-manage` to display more verbose output for the specified command:

```
neutron-db-manage current --verbose
```

For some commands the wrapper needs to know the entrypoint of the core plugin for the installation. This can be read from the configuration file(s) or specified using the `--core_plugin` option:

```
neutron-db-manage --core_plugin neutron.plugins.ml2.plugin.Ml2Plugin
↪<commands>
```

When giving examples below of using the wrapper the options will not be shown. It is assumed you will use the options that you need for your environment.

For new deployments you will start with an empty database. You then upgrade to the latest database version via:

```
neutron-db-manage upgrade heads
```

For existing deployments the database will already be at some version. To check the current database version:

```
neutron-db-manage current
```

After installing a new version of Neutron server, upgrading the database is the same command:

```
neutron-db-manage upgrade heads
```

To create a script to run the migration offline:

```
neutron-db-manage upgrade heads --sql
```

To run the offline migration between specific migration versions:

```
neutron-db-manage upgrade <start version>:<end version> --sql
```

Upgrade the database incrementally:

```
neutron-db-manage upgrade --delta <# of revs>
```

NOTE: Database downgrade is not supported.

Migration Branches

Neutron makes use of alembic branches for two purposes.

1. Independent Sub-Project Tables

Various Sub-Projects can be installed with Neutron. Each sub-project registers its own alembic branch which is responsible for migrating the schemas of the tables owned by the sub-project.

The neutron-db-manage script detects which sub-projects have been installed by enumerating the `neutron.db.alembic_migrations` entrypoints. For more details see the Entry Points section of Contributing extensions to Neutron.

The neutron-db-manage script runs the given alembic command against all installed sub-projects. (An exception is the `revision` command, which is discussed in the *Developers* section below.)

2. Offline/Online Migrations

Since Liberty, Neutron maintains two parallel alembic migration branches.

The first one, called expand, is used to store expansion-only migration rules. Those rules are strictly additive and can be applied while neutron-server is running. Examples of additive database schema changes are: creating a new table, adding a new table column, adding a new index, etc.

The second branch, called contract, is used to store those migration rules that are not safe to apply while neutron-server is running. Those include: column or table removal, moving data from one part of the database into another (renaming a column, transforming single table into multiple, etc.), introducing or modifying constraints, etc.

The intent of the split is to allow invoking those safe migrations from expand branch while neutron-server is running, reducing downtime needed to upgrade the service.

For more details, see the *Expand and Contract Scripts* section below.

Developers

A database migration script is required when you submit a change to Neutron or a sub-project that alters the database model definition. The migration script is a special python file that includes code to upgrade the database to match the changes in the model definition. Alembic will execute these scripts in order to provide a linear migration path between revisions. The neutron-db-manage command can be used to generate migration scripts for you to complete. The operations in the template are those supported by the Alembic migration library.

Running neutron-db-manage without devstack

When, as a developer, you want to work with the Neutron DB schema and alembic migrations only, it can be rather tedious to rely on devstack just to get an up-to-date neutron-db-manage installed. This section describes how to work on the schema and migration scripts with just the unit test virtualenv and mysql. You can also operate on a separate test database so you dont mess up the installed Neutron database.

Setting up the environment

Install mysql service

This only needs to be done once since it is a system install. If you have run devstack on your system before, then the mysql service is already installed and you can skip this step.

Mysql must be configured as installed by devstack, and the following script accomplishes this without actually running devstack:

```
INSTALL_MYSQL_ONLY=True ./tools/configure_for_func_testing.sh ../devstack
```

Run this from the root of the neutron repo. It assumes an up-to-date clone of the devstack repo is in `../devstack`.

Note that you must know the mysql root password. It is derived from (in order of precedence):

- `$MYSQL_PASSWORD` in your environment
- `$MYSQL_PASSWORD` in `../devstack/local.conf`
- `$MYSQL_PASSWORD` in `../devstack/localrc`
- default of secretmysql from `tools/configure_for_func_testing.sh`

Work on a test database

Rather than using the neutron database when working on schema and alembic migration script changes, we can work on a test database. In the examples below, we use a database named `testdb`.

To create the database:

```
mysql -e "create database testdb;"
```

You will often need to clear it to re-run operations from a blank database:

```
mysql -e "drop database testdb; create database testdb;"
```

To work on the test database instead of the neutron database, point to it with the `--database-connection` option:

```
neutron-db-manage --database-connection mysql+pymysql://
↪root:secretmysql@127.0.0.1/testdb?charset=utf8 <commands>
```

You may find it convenient to set up an alias (in your .bashrc) for this:

```
alias test-db-manage='neutron-db-manage --database-connection␣
↪mysql+pymysql://root:secretmysql@127.0.0.1/testdb?charset=utf8'
```

Create and activate the virtualenv

From the root of the neutron (or sub-project) repo directory, run:

```
tox --notest -r -e py38
source .tox/py38/bin/activate
```

Now you can use the `test-db-manage` alias in place of `neutron-db-manage` in the script auto-generation instructions below.

When you are done, exit the virtualenv:

```
deactivate
```

Script Auto-generation

This section describes how to auto-generate an alembic migration script for a model change. You may either use the system installed devstack environment, or a virtualenv + testdb environment as described in *Running neutron-db-manage without devstack*.

Stop the neutron service. Work from the base directory of the neutron (or sub-project) repo. Check out the master branch and do `git pull` to ensure it is fully up to date. Check out your development branch and rebase to master.

NOTE: Make sure you have not updated the `CONTRACT_HEAD` or `EXPAND_HEAD` yet at this point.

Start with an empty database and upgrade to heads:

```
mysql -e "drop database neutron; create database neutron;"
neutron-db-manage upgrade heads
```

The database schema is now created without your model changes. The alembic `revision --autogenerate` command will look for differences between the schema generated by the upgrade command and the schema defined by the models, including your model updates:

```
neutron-db-manage revision -m "description of revision" --autogenerate
```

This generates a prepopulated template with the changes needed to match the database state with the models. You should inspect the autogenerated template to ensure that the proper models have been altered. When running the above command you will probably get the following error message:

```
Multiple heads are present; please specify the head revision on which the
new revision should be based, or perform a merge.
```

This is alembic telling you that it does not know which branch (contract or expand) to generate the revision for. You must decide, based on whether you are doing contracting or expanding changes to the schema, and provide either the `--contract` or `--expand` option. If you have both types of changes, you must run the command twice, once with each option, and then manually edit the generated revision scripts to separate the migration operations.

In rare circumstances, you may want to start with an empty migration template and manually author the changes necessary for an upgrade. You can create a blank file for a branch via:

```
neutron-db-manage revision -m "description of revision" --expand
neutron-db-manage revision -m "description of revision" --contract
```

NOTE: If you use above command you should check that migration is created in a directory that is named as current release. If not, please raise the issue with the development team (IRC, mailing list, launchpad bug).

NOTE: The description of revision text should be a simple English sentence. The first 30 characters of the description will be used in the file name for the script, with underscores substituted for spaces. If the truncation occurs at an awkward point in the description, you can modify the script file name manually before committing.

The timeline on each alembic branch should remain linear and not interleave with other branches, so that there is a clear path when upgrading. To verify that alembic branches maintain linear timelines, you can run this command:

```
neutron-db-manage check_migration
```

If this command reports an error, you can troubleshoot by showing the migration timelines using the `history` command:

```
neutron-db-manage history
```

Expand and Contract Scripts

The obsolete branchless design of a migration script included that it indicates a specific version of the schema, and includes directives that apply all necessary changes to the database at once. If we look for example at the script `2d2a8a565438_hierarchical_binding.py`, we will see:

```
# .../alembic_migrations/versions/2d2a8a565438_hierarchical_binding.py

def upgrade():

    # .. inspection code ...

    op.create_table(
        'ml2_port_binding_levels',
        sa.Column('port_id', sa.String(length=36), nullable=False),
        sa.Column('host', sa.String(length=255), nullable=False),
        # ... more columns ...
    )

    for table in port_binding_tables:
        op.execute((
            "INSERT INTO ml2_port_binding_levels "
            "SELECT port_id, host, 0 AS level, driver, segment AS segment_
↪id "
            "FROM %s "
            "WHERE host <> '' "
            "AND driver <> '';"
        ) % table)
```

(continues on next page)

(continued from previous page)

```
    op.drop_constraint(fk_name_dvr[0], 'ml2_dvr_port_bindings', 'foreignkey
↪')
    op.drop_column('ml2_dvr_port_bindings', 'cap_port_filter')
    op.drop_column('ml2_dvr_port_bindings', 'segment')
    op.drop_column('ml2_dvr_port_bindings', 'driver')

    # ... more DROP instructions ...
```

The above script contains directives that are both under the expand and contract categories, as well as some data migrations. the `op.create_table` directive is an expand; it may be run safely while the old version of the application still runs, as the old code simply doesnt look for this table. The `op.drop_constraint` and `op.drop_column` directives are contract directives (the drop column more so than the drop constraint); running at least the `op.drop_column` directives means that the old version of the application will fail, as it will attempt to access these columns which no longer exist.

The data migrations in this script are adding new rows to the newly added `ml2_port_binding_levels` table.

Under the new migration script directory structure, the above script would be stated as two scripts; an expand and a contract script:

```
# expansion operations
# .../alembic_migrations/versions/liberty/expand/2bde560fc638_hierarchical_
↪binding.py

def upgrade():

    op.create_table(
        'ml2_port_binding_levels',
        sa.Column('port_id', sa.String(length=36), nullable=False),
        sa.Column('host', sa.String(length=255), nullable=False),
        # ... more columns ...
    )


# contraction operations
# .../alembic_migrations/versions/liberty/contract/4405aedc050e_
↪hierarchical_binding.py

def upgrade():

    for table in port_binding_tables:
        op.execute((
            "INSERT INTO ml2_port_binding_levels "
            "SELECT port_id, host, 0 AS level, driver, segment AS segment_
↪id "
            "FROM %s "
            "WHERE host <> '' "
            "AND driver <> '';"
        ) % table)

    op.drop_constraint(fk_name_dvr[0], 'ml2_dvr_port_bindings', 'foreignkey
↪')
    op.drop_column('ml2_dvr_port_bindings', 'cap_port_filter')
```

(continues on next page)

(continued from previous page)

```
    op.drop_column('ml2_dvr_port_bindings', 'segment')
    op.drop_column('ml2_dvr_port_bindings', 'driver')

    # ... more DROP instructions ...
```

The two scripts would be present in different subdirectories and also part of entirely separate versioning streams. The expand operations are in the expand script, and the contract operations are in the contract script.

For the time being, data migration rules also belong to contract branch. There is expectation that eventually live data migrations move into middleware that will be aware about different database schema elements to converge on, but Neutron is still not there.

Scripts that contain only expansion or contraction rules do not require a split into two parts.

If a contraction script depends on a script from expansion stream, the following directive should be added in the contraction script:

```
depends_on = ('<expansion-revision>',)
```

Expand and Contract Branch Exceptions

In some cases, we have to have expand operations in contract migrations. For example, table networksegments was renamed in contract migration, so all operations with this table are required to be in contract branch as well. For such cases, we use the `contract_creation_exceptions` that should be implemented as part of such migrations. This is needed to get functional tests pass.

Usage:

```
def contract_creation_exceptions():
    """Docstring should explain why we allow such exception for contract
    branch.
    """
    return {
        sqlalchemy_obj_type: ['name']
        # For example: sa.Column: ['subnets.segment_id']
    }
```

HEAD files for conflict management

In directory `neutron/db/migration/alembic_migrations/versions` there are two files, `CONTRACT_HEAD` and `EXPAND_HEAD`. These files contain the ID of the head revision in each branch. The purpose of these files is to validate the revision timelines and prevent non-linear changes from entering the merge queue.

When you create a new migration script by neutron-db-manage these files will be updated automatically. But if another migration script is merged while your change is under review, you will need to resolve the conflict manually by changing the `down_revision` in your migration script.

Applying database migration rules

To apply just expansion rules, execute:

```
neutron-db-manage upgrade --expand
```

After the first step is done, you can stop neutron-server, apply remaining non-expansive migration rules, if any:

```
neutron-db-manage upgrade --contract
```

and finally, start your neutron-server again.

If you have multiple neutron-server instances in your cloud, and there are pending contract scripts not applied to the database, full shutdown of all those services is required before upgrade contract is executed. You can determine whether there are any pending contract scripts by checking the message returned from the following command:

```
neutron-db-manage has_offline_migrations
```

If you are not interested in applying safe migration rules while the service is running, you can still upgrade database the old way, by stopping the service, and then applying all available rules:

```
neutron-db-manage upgrade head[s]
```

It will apply all the rules from both the expand and the contract branches, in proper order.

Tagging milestone revisions

When named release (liberty, mitaka, etc.) is done for neutron or a sub-project, the alembic revision scripts at the head of each branch for that release must be tagged. This is referred to as a milestone revision tag.

For example, here is a patch that tags the liberty milestone revisions for the neutron-fwaas sub-project. Note that each branch (expand and contract) is tagged.

Tagging milestones allows neutron-db-manage to upgrade the schema to a milestone release, e.g.:

```
neutron-db-manage upgrade liberty
```

Generation of comparable metadata with current database schema

Directory `neutron/db/migration/models` contains module `head.py`, which provides all database models at current HEAD. Its purpose is to create comparable metadata with the current database schema. The database schema is generated by alembic migration scripts. The models must match, and this is verified by a model-migration sync test in Neutrons functional test suite. That test requires all modules containing DB models to be imported by head.py in order to make a complete comparison.

When adding new database models, developers must update this module, otherwise the change will fail to merge.

14.4.8 Upgrade checks

Introduction

CLI tool `neutron-status upgrade check` contains checks which perform a release-specific readiness check before restarting services with new code. For more details see neutron-status command-line client page.

3rd party plugins checks

Neutron upgrade checks script allows to add checks by stadium and 3rd party projects. The `neutron-status` script detects which sub-projects have been installed by enumerating the `neutron.status.upgrade.checks` entrypoints. For more details see the Entry Points section of Contributing extensions to Neutron. Checks can be run in random order and should be independent from each other.

The recommended entry point name is a repository name: For example, neutron-fwaas for FWaaS and networking-sfc for SFC:

```
neutron.status.upgrade.checks =
    neutron-fwaas = neutron_fwaas.upgrade.checks:Checks
```

Entrypoint should be class which inherits from `neutron.cmd.upgrade_checks.base.BaseChecks`.

An example of a checks class can be found in `neutron.cmd.upgrade_checks.checks.CoreChecks`.

14.4.9 Testing

Testing Neutron

Why Should You Care

Theres two ways to approach testing:

1) Write unit tests because theyre required to get your patch merged. This typically involves mock heavy tests that assert that your code is as written.

2) Putting as much thought into your testing strategy as you do to the rest of your code. Use different layers of testing as appropriate to provide high *quality* coverage. Are you touching an agent? Test it against an actual system! Are you adding a new API? Test it for race conditions against a real database! Are you adding a new cross-cutting feature? Test that it does what its supposed to do when run on a real cloud!

Do you feel the need to verify your change manually? If so, the next few sections attempt to guide you through Neutrons different test infrastructures to help you make intelligent decisions and best exploit Neutrons test offerings.

Definitions

We will talk about three classes of tests: unit, functional and integration. Each respective category typically targets a larger scope of code. Other than that broad categorization, here are a few more characteristic:

- Unit tests - Should be able to run on your laptop, directly following a git clone of the project. The underlying system must not be mutated, mocks can be used to achieve this. A unit test typically targets a function or class.
- Functional tests - Run against a pre-configured environment (tools/configure_for_func_testing.sh). Typically test a component such as an agent using no mocks.
- Integration tests - Run against a running cloud, often target the API level, but also scenarios or user stories. You may find such tests under tests/fullstack, and in the Tempest, Rally and neutron-tempest-plugin(neutron_tempest_plugin/api|scenario) projects.

Tests in the Neutron tree are typically organized by the testing infrastructure used, and not by the scope of the test. For example, many tests under the unit directory invoke an API call and assert that the expected output was received. The scope of such a test is the entire Neutron server stack, and clearly not a specific function such as in a typical unit test.

Testing Frameworks

The different frameworks are listed below. The intent is to list the capabilities of each testing framework as to help the reader understand when should each tool be used. Remember that when adding code that touches many areas of Neutron, each area should be tested with the appropriate framework. Overlap between different test layers is often desirable and encouraged.

Unit Tests

Unit tests (neutron/tests/unit/) are meant to cover as much code as possible. They are designed to test the various pieces of the Neutron tree to make sure any new changes dont break existing functionality. Unit tests have no requirements nor make changes to the system they are running on. They use an in-memory sqlite database to test DB interaction.

At the start of each test run:

- RPC listeners are mocked away.
- The fake Oslo messaging driver is used.

At the end of each test run:

- Mocks are automatically reverted.
- The in-memory database is cleared of content, but its schema is maintained.
- The global Oslo configuration object is reset.

The unit testing framework can be used to effectively test database interaction, for example, distributed routers allocate a MAC address for every host running an OVS agent. One of DVRs DB mixins implements a method that lists all host MAC addresses. Its test looks like this:

```
def test_get_dvr_mac_address_list(self):
    self._create_dvr_mac_entry('host_1', 'mac_1')
    self._create_dvr_mac_entry('host_2', 'mac_2')
    mac_list = self.mixin.get_dvr_mac_address_list(self.ctx)
    self.assertEqual(2, len(mac_list))
```

It inserts two new host MAC address, invokes the method under test and asserts its output. The test has many things going for it:

- It targets the method under test correctly, not taking on a larger scope than is necessary.
- It does not use mocks to assert that methods were called, it simply invokes the method and asserts its output (In this case, that the list method returns two records).

This is allowed by the fact that the method was built to be testable - The method has clear input and output with no side effects.

You can get oslo.db to generate a file-based sqlite database by setting OS_TEST_DBAPI_ADMIN_CONNECTION to a file based URL as described in this mailing list post. This file will be created but (confusingly) wont be the actual file used for the database. To find the actual file, set a break point in your test method and inspect self.engine.url.

```
$ OS_TEST_DBAPI_ADMIN_CONNECTION=sqlite:///sqlite.db .tox/py38/bin/python -
↪m \
    testtools.run neutron.tests.unit...
...
(Pdb) self.engine.url
sqlite:////tmp/iwbgvhbshp.db
```

Now, you can inspect this file using sqlite3.

```
$ sqlite3 /tmp/iwbgvhbshp.db
```

Functional Tests

Functional tests (neutron/tests/functional/) are intended to validate actual system interaction. Mocks should be used sparingly, if at all. Care should be taken to ensure that existing system resources are not modified and that resources created in tests are properly cleaned up both on test success and failure.

Lets examine the benefits of the functional testing framework. Neutron offers a library called ip_lib that wraps around the ip binary. One of its methods is called device_exists which accepts a device name and a namespace and returns True if the device exists in the given namespace. Its easy building a test that targets the method directly, and such a test would be considered a unit test. However, what framework should such a test use? A test using the unit tests framework could not mutate state on the system, and so could not actually create a device and assert that it now exists. Such a test would look roughly like this:

- It would mock execute, a method that executes shell commands against the system to return an IP device named foo.
- It would then assert that when device_exists is called with foo, it returns True, but when called with a different device name it returns False.
- It would most likely assert that execute was called using something like: ip link show foo.

The value of such a test is arguable. Remember that new tests are not free, they need to be maintained. Code is often refactored, reimplemented and optimized.

- There are other ways to find out if a device exists (Such as by looking at /sys/class/net), and in such a case the test would have to be updated.
- Methods are mocked using their name. When methods are renamed, moved or removed, their mocks must be updated. This slows down development for avoidable reasons.
- Most importantly, the test does not assert the behavior of the method. It merely asserts that the code is as written.

When adding a functional test for device_exists, several framework level methods were added. These methods may now be used by other tests as well. One such method creates a virtual device in a namespace, and ensures that both the namespace and the device are cleaned up at the end of the test run regardless of success or failure using the addCleanup method. The test generates details for a temporary device, asserts that a device by that name does not exist, creates that device, asserts that it now exists, deletes it, and asserts that it no longer exists. Such a test avoids all three issues mentioned above if it were written using the unit testing framework.

Functional tests are also used to target larger scope, such as agents. Many good examples exist: See the OVS, L3 and DHCP agents functional tests. Such tests target a top level agent method and assert that the system interaction that was supposed to be performed was indeed performed. For example, to test the DHCP agents top level method that accepts network attributes and configures dnsmasq for that network, the test:

- Instantiates an instance of the DHCP agent class (But does not start its process).
- Calls its top level function with prepared data.
- Creates a temporary namespace and device, and calls dhclient from that namespace.
- Assert that the device successfully obtained the expected IP address.

Test exceptions

Test neutron.tests.functional.agent.test_ovs_flows.OVSFlowTestCase.test_install_flood_to_tun is currently skipped if openvswitch version is less than 2.5.1. This version contains bug where appctl command prints wrong output for Final flow. Its been fixed in openvswitch 2.5.1 in this commit. If openvswitch version meets the test requirement then the test is triggered normally.

Fullstack Tests

Why?

The idea behind fullstack testing is to fill a gap between unit + functional tests and Tempest. Tempest tests are expensive to run, and target black box API tests exclusively. Tempest requires an OpenStack deployment to be run against, which can be difficult to configure and setup. Full stack testing addresses these issues by taking care of the deployment itself, according to the topology that the test requires. Developers further benefit from full stack testing as it can sufficiently simulate a real environment and provide a rapidly reproducible way to verify code while youre still writing it.

How?

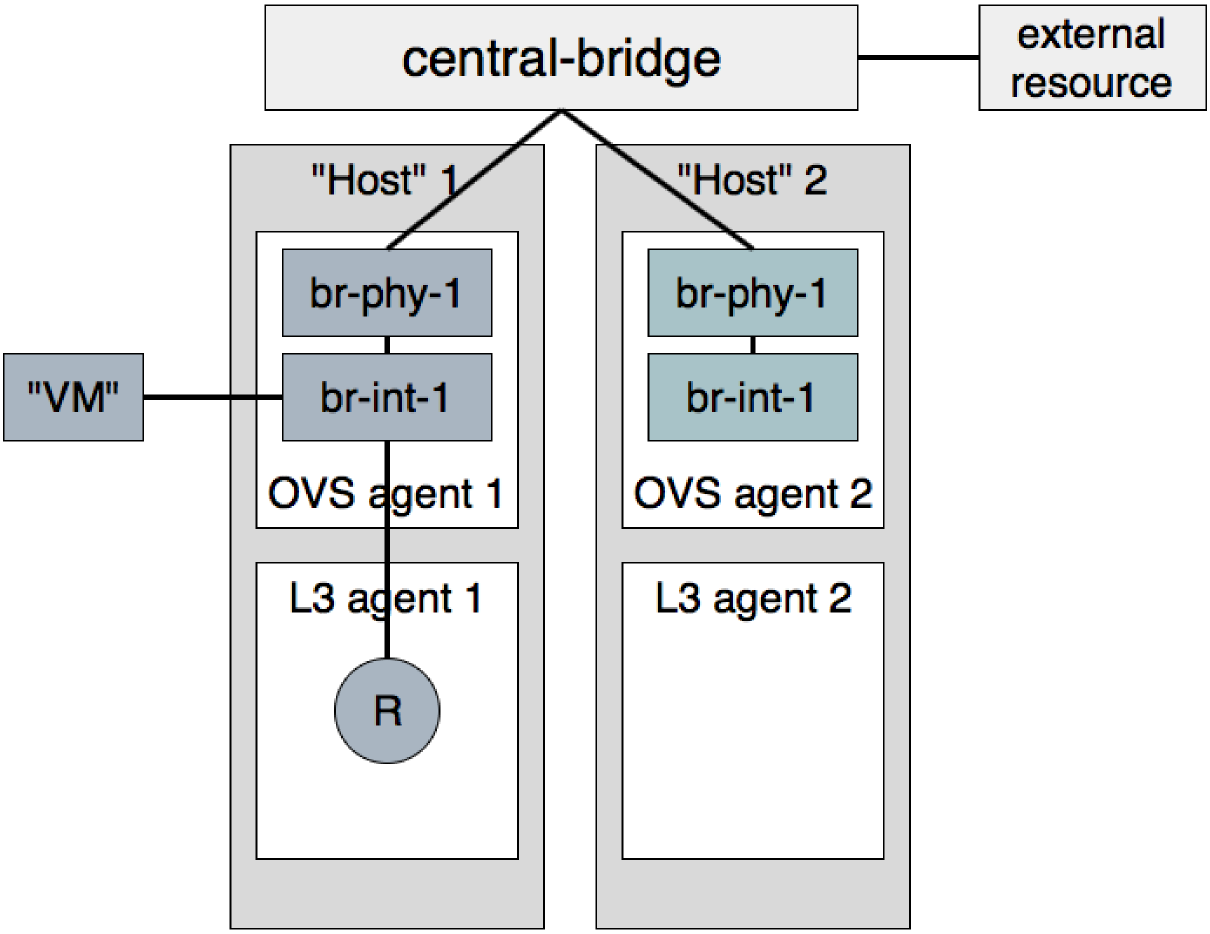
Full stack tests set up their own Neutron processes (Server & agents). They assume a working Rabbit and MySQL server before the run starts. Instructions on how to run fullstack tests on a VM are available below.

Each test defines its own topology (What and how many servers and agents should be running).

Since the test runs on the machine itself, full stack testing enables white box testing. This means that you can, for example, create a router through the API and then assert that a namespace was created for it.

Full stack tests run in the Neutron tree with Neutron resources alone. You may use the Neutron API (The Neutron server is set to NOAUTH so that Keystone is out of the picture). VMs may be simulated with a container-like class: neutron.tests.fullstack.resources.machine.FakeFullstackMachine. An example of its usage may be found at: neutron/tests/fullstack/test_connectivity.py.

Full stack testing can simulate multi node testing by starting an agent multiple times. Specifically, each node would have its own copy of the OVS/LinuxBridge/DHCP/L3 agents, all configured with the same host value. Each OVS agent is connected to its own pair of br-int/br-ex, and those bridges are then interconnected. For LinuxBridge agent each agent is started in its own namespace, called host-<some_random_value>. Such namespaces are connected with OVS central bridge to each other.



Segmentation at the database layer is guaranteed by creating a database per test. The messaging layer achieves segmentation by utilizing a RabbitMQ feature called vhosts. In short, just like a MySQL server serve multiple databases, so can a RabbitMQ server serve multiple messaging domains. Exchanges and queues in one vhost are segmented from those in another vhost.

Please note that if the change you would like to test using fullstack tests involves a change to python-neutronclient as well as neutron, then you should make sure your fullstack tests are in a separate third change that depends on the python-neutronclient change using the Depends-On tag in the commit message. You will need to wait for the next release of python-neutronclient, and a minimum version bump for python-neutronclient in the global requirements, before your fullstack tests will work in the gate. This is because tox uses the version of python-neutronclient listed in the upper-constraints.txt file in the openstack/requirements repository.

When?

1) Youd like to test the interaction between Neutron components (Server and agents) and have already tested each component in isolation via unit or functional tests. You should have many unit tests, fewer tests to test a component and even fewer to test their interaction. Edge cases should not be tested with full stack testing.
2) Youd like to increase coverage by testing features that require multi node testing such as l2pop, L3 HA and DVR.
3) Youd like to test agent restarts. Weve found bugs in the OVS, DHCP and L3 agents and havent found an effective way to test these scenarios. Full stack testing can help here as the full stack infrastructure can restart an agent during the test.

Example

Neutron offers a Quality of Service API, initially offering bandwidth capping at the port level. In the reference implementation, it does this by utilizing an OVS feature. neutron.tests.fullstack.test_qos.TestBwLimitQoSOvs.test_bw_limit_qos_policy_rule_lifecycle is a positive example of how the fullstack testing infrastructure should be used. It creates a network, subnet, QoS policy & rule and a port utilizing that policy. It then asserts that the expected bandwidth limitation is present on the OVS bridge connected to that port. The test is a true integration test, in the sense that it invokes the API and then asserts that Neutron interacted with the hypervisor appropriately.

Gate exceptions

Currently we compile openvswitch kernel module from source for fullstack job on the gate. The reason is to fix bug related to local VXLAN tunneling which is present in current Ubuntu Xenial 16.04 kernel. Kernel was fixed with this commit and backported with this openvswitch commit.

API Tests

API tests (neutron-tempest-plugin/neutron_tempest_plugin/api/) are intended to ensure the function and stability of the Neutron API. As much as possible, changes to this path should not be made at the same time as changes to the code to limit the potential for introducing backwards-incompatible changes, although the same patch that introduces a new API should include an API test.

Since API tests target a deployed Neutron daemon that is not test-managed, they should not depend on controlling the runtime configuration of the target daemon. API tests should be black-box - no assumptions should be made about implementation. Only the contract defined by Neutrons REST API should be validated, and all interaction with the daemon should be via a REST client.

The neutron-tempest-plugin/neutron_tempest_plugin directory was copied from the Tempest project around the Kilo timeframe. At the time, there was an overlap of tests between the Tempest and Neutron repositories. This overlap was then eliminated by carving out a subset of resources that belong to Tempest, with the rest in Neutron.

API tests that belong to Tempest deal with a subset of Neutrons resources:

- Port
- Network
- Subnet
- Security Group
- Router
- Floating IP

These resources were chosen for their ubiquity. They are found in most Neutron deployments regardless of plugin, and are directly involved in the networking and security of an instance. Together, they form the bare minimum needed by Neutron.

This is excluding extensions to these resources (For example: Extra DHCP options to subnets, or snat_gateway mode to routers) that are not mandatory in the majority of cases.

Tests for other resources should be contributed to the Neutron repository. Scenario tests should be similarly split up between Tempest and Neutron according to the API theyre targeting.

To create an API test, the testing class must at least inherit from neutron_tempest_plugin.api.base.BaseNetworkTest base class. As some of tests may require certain extensions to be enabled, the base class provides `required_extensions` class attribute which can be used by subclasses to define a list of required extensions for particular test class.

Scenario Tests

Scenario tests (neutron-tempest-plugin/neutron_tempest_plugin/scenario), like API tests, use the Tempest test infrastructure and have the same requirements. Guidelines for writing a good scenario test may be found at the Tempest developer guide: https://docs.openstack.org/tempest/latest/field_guide/scenario.html

Scenario tests, like API tests, are split between the Tempest and Neutron repositories according to the Neutron API the test is targeting.

Some scenario tests require advanced `Glance` images (for example, `Ubuntu` or `CentOS`) in order to pass. Those tests are skipped by default. To enable them, include the following in `tempest.conf`:

```
[compute]
image_ref = <uuid of advanced image>
[neutron_plugin_options]
image_is_advanced = True
```

Specific test requirements for advanced images are:

1. `test_trunk` requires `802.11q` kernel module loaded.

Rally Tests

Rally tests (rally-jobs/plugins) use the rally infrastructure to exercise a neutron deployment. Guidelines for writing a good rally test can be found in the rally plugin documentation. There are also some examples in tree; the process for adding rally plugins to neutron requires three steps: 1) write a plugin and place it under rally-jobs/plugins/. This is your rally scenario; 2) (optional) add a setup file under rally-jobs/extra/. This is any devstack configuration required to make sure your environment can successfully process your scenario requests; 3) edit neutron-neutron.yaml. This is your scenario contract or SLA.

Development Process

It is expected that any new changes that are proposed for merge come with tests for that feature or code area. Any bugs fixes that are submitted must also have tests to prove that they stay fixed! In addition, before proposing for merge, all of the current tests should be passing.

Structure of the Unit Test Tree

The structure of the unit test tree should match the structure of the code tree, e.g.

```
- target module: neutron.agent.utils

- test module: neutron.tests.unit.agent.test_utils
```

Unit test modules should have the same path under neutron/tests/unit/ as the module they target has under neutron/, and their name should be the name of the target module prefixed by *test_*. This requirement is intended to make it easier for developers to find the unit tests for a given module.

Similarly, when a test module targets a package, that modules name should be the name of the package prefixed by *test_* with the same path as when a test targets a module, e.g.

```
- target package: neutron.ipam

- test module: neutron.tests.unit.test_ipam
```

The following command can be used to validate whether the unit test tree is structured according to the above requirements:

```
./tools/check_unit_test_structure.sh
```

Where appropriate, exceptions can be added to the above script. If code is not part of the Neutron namespace, for example, its probably reasonable to exclude their unit tests from the check.

Note: At no time should the production code import anything from testing subtree (neutron.tests). There are distributions that split out neutron.tests modules in a separate package that is not installed by default, making any code that relies on presence of the modules to fail. For example, RDO is one of those distributions.

Running Tests

Before submitting a patch for review you should always ensure all tests pass; a tox run is triggered by the jenkins gate executed on gerrit for each patch pushed for review.

Neutron, like other OpenStack projects, uses tox for managing the virtual environments for running test cases. It uses Testr for managing the running of the test cases.

Tox handles the creation of a series of virtualenvs that target specific versions of Python.

Testr handles the parallel execution of series of test cases as well as the tracking of long-running tests and other things.

For more information on the standard Tox-based test infrastructure used by OpenStack and how to do some common test/debugging procedures with Testr, see this wiki page: https://wiki.openstack.org/wiki/Testr

PEP8 and Unit Tests

Running pep8 and unit tests is as easy as executing this in the root directory of the Neutron source code:

```
tox
```

To run only pep8:

```
tox -e pep8
```

Since pep8 includes running pylint on all files, it can take quite some time to run. To restrict the pylint check to only the files altered by the latest patch changes:

```
tox -e pep8 HEAD~1
```

To run only the unit tests:

```
tox -e py38
```

Many changes span across both the neutron and neutron-lib repos, and tox will always build the test environment using the published module versions specified in requirements.txt. To run tox tests against a different version of neutron-lib, use the TOX_ENV_SRC_MODULES environment variable to point at a local package repo.

For example, to run against the master branch of neutron-lib:

```
cd $SRC
git clone https://opendev.org/openstack/neutron-lib
cd $NEUTRON_DIR
env TOX_ENV_SRC_MODULES=$SRC/neutron-lib tox -r -e py38
```

To run against a change of your own, repeat the same steps, but use the directory with your changes, not a fresh clone.

To run against a particular gerrit change of the lib (substituting the desired gerrit refs for this example):

```
cd $SRC
git clone https://opendev.org/openstack/neutron-lib
cd neutron-lib
git fetch https://opendev.org/openstack/neutron-lib refs/changes/13/635313/
↪6 && git checkout FETCH_HEAD
cd $NEUTRON_DIR
env TOX_ENV_SRC_MODULES=$SRC/neutron-lib tox -r -e py38
```

Note that the -r is needed to re-create the tox virtual envs, and will also be needed to restore them to standard when not using this method.

Any pip installable package can be overriden with this environment variable, not just neutron-lib. To specify multiple packages to override, specify them as a space separated list to TOX_ENV_SRC_MODULES. Example:

```
env TOX_ENV_SRC_MODULES="$SRC/neutron-lib $SRC/oslo.db" tox -r -e py38
```

Functional Tests

To run functional tests that do not require sudo privileges or specific-system dependencies:

```
tox -e functional
```

To run all the functional tests, including those requiring sudo privileges and system-specific dependencies, the procedure defined by tools/configure_for_func_testing.sh should be followed.

IMPORTANT: configure_for_func_testing.sh relies on DevStack to perform extensive modification to the underlying host. Execution of the script requires sudo privileges and it is recommended that the following commands be invoked only on a clean and disposable VM. A VM that has had DevStack previously installed on it is also fine.

```
git clone https://opendev.org/openstack/devstack ../devstack
./tools/configure_for_func_testing.sh ../devstack -i
tox -e dsvm-functional
```

The -i option is optional and instructs the script to use DevStack to install and configure all of Neutrons package dependencies. It is not necessary to provide this option if DevStack has already been used to deploy Neutron to the target host.

Fullstack Tests

To run all the fullstack tests, you may use:

```
tox -e dsvm-fullstack
```

Since fullstack tests often require the same resources and dependencies as the functional tests, using the configuration script tools/configure_for_func_testing.sh is advised (as described above). Before running the script, you must first set the following environment variable so things are setup correctly

```
export VENV=dsvm-fullstack
```

When running fullstack tests on a clean VM for the first time, it is important to make sure all of Neutrons package dependencies have been met. As mentioned in the functional test section above, this can be done by running the configure script with the -i argument

```
./tools/configure_for_func_testing.sh ../devstack -i
```

You can also run ./stack.sh, and if successful, it will have also verified the package dependencies have been met. When running on a new VM it is suggested to set the following environment variable as well, to make sure that all requirements (including database and message bus) are installed and set

```
export IS_GATE=False
```

Fullstack-based Neutron daemons produce logs to a sub-folder in the $OS_LOG_PATH directory (default: /opt/stack/logs, note: if running fullstack tests on a newly created VM, make sure that $OS_LOG_PATH exists with the correct permissions) called dsvm-fullstack-logs. For example, a test named test_example will produce logs in $OS_LOG_PATH/dsvm-fullstack-logs/test_example/, as well as create $OS_LOG_PATH/dsvm-fullstack-logs/test_example.txt, so that is a good place to look if your test is failing.

The fullstack test suite assumes 240.0.0.0/4 (Class E) range in the root namespace of the test machine is available for its usage.

Fullstack tests execute a custom dhclient-script. From kernel version 4.14 onward, apparmor on certain distros could deny the execution of this script. To be sure, check journalctl

```
sudo journalctl | grep DENIED | grep fullstack-dhclient-script
```

To execute these tests, the easiest workaround is to disable apparmor

```
sudo systemctl stop apparmor
sudo systemctl disable apparmor
```

A more granular solution could be to disable apparmor only for dhclient

```
sudo ln -s /etc/apparmor.d/sbin.dhclient /etc/apparmor.d/disable/
```

API & Scenario Tests

To run the api or scenario tests, deploy Tempest, neutron-tempest-plugin and Neutron with DevStack and then run the following command, from the tempest directory:

```
$ export DEVSTACK_GATE_TEMPEST_REGEX="neutron"
$ tox -e all-plugin $DEVSTACK_GATE_TEMPEST_REGEX
```

If you want to limit the amount of tests, or run an individual test, you can do, for instance:

```
$ tox -e all-plugin neutron_tempest_plugin.api.admin.test_routers_ha
$ tox -e all-plugin neutron_tempest_plugin.api.test_qos.QosTestJSON.test_
↪create_policy
```

If you want to use special config for Neutron, like use advanced images (Ubuntu or CentOS) testing advanced features, you may need to add config in tempest/etc/tempest.conf:

```
[neutron_plugin_options]
image_is_advanced = True
```

The Neutron tempest plugin configs are under `neutron_plugin_options` scope of `tempest.conf`.

Running Individual Tests

For running individual test modules, cases or tests, you just need to pass the dot-separated path you want as an argument to it.

For example, the following would run only a single test or test case:

```
$ tox -e py38 neutron.tests.unit.test_manager
$ tox -e py38 neutron.tests.unit.test_manager.NeutronManagerTestCase
$ tox -e py38 neutron.tests.unit.test_manager.NeutronManagerTestCase.test_
↪service_plugin_is_loaded
```

If you want to pass other arguments to stestr, you can do the following:

```
$ tox -e py38 -- neutron.tests.unit.test_manager --serial
```

Coverage

Neutron has a fast growing code base and there are plenty of areas that need better coverage.

To get a grasp of the areas where tests are needed, you can check current unit tests coverage by running:

```
$ tox -ecover
```

Since the coverage command can only show unit test coverage, a coverage document is maintained that shows test coverage per area of code in: doc/source/devref/testing_coverage.rst. You could also rely on Zuul logs, that are generated post-merge (not every project builds coverage results). To access them, do the following:

- Check out the latest merge commit
- Go to: http://logs.openstack.org/<first-2-digits-of-sha1>/<sha1>/post/neutron-coverage/.
- Spec is a work in progress to provide a better landing page.

Debugging

By default, calls to pdb.set_trace() will be ignored when tests are run. For pdb statements to work, invoke tox as follows:

```
$ tox -e venv -- python -m testtools.run [test module path]
```

Tox-created virtual environments (venvs) can also be activated after a tox run and reused for debugging:

```
$ tox -e venv
$ . .tox/venv/bin/activate
$ python -m testtools.run [test module path]
```

Tox packages and installs the Neutron source tree in a given venv on every invocation, but if modifications need to be made between invocation (e.g. adding more pdb statements), it is recommended that the source tree be installed in the venv in editable mode:

```
# run this only after activating the venv
$ pip install --editable .
```

Editable mode ensures that changes made to the source tree are automatically reflected in the venv, and that such changes are not overwritten during the next tox run.

Post-mortem Debugging

TBD: how to do this with tox.

References

Full Stack Testing

Goals

- **Stabilize the job:**
  - Fix L3 HA failure
  - Look in to non-deterministic failures when adding a large amount of tests (Possibly bug 1486199).
  - Switch to kill signal 15 to terminate agents (Bug 1487548).
- Convert the L3 HA failover functional test to a full stack test
- Write DVR tests
- Write additional L3 HA tests
- Write a test that validates DVR + L3 HA integration after https://bugs.launchpad.net/neutron/+bug/1365473 is fixed.

Test Coverage

The intention is to track merged features or areas of code that lack certain types of tests. This document may be used both by developers that want to contribute tests, and operators that are considering adopting a feature.

Coverage

Note that while both API and scenario tests target a deployed OpenStack cloud, API tests are under the Neutron tree and scenario tests are under the Tempest tree.

It is the expectation that API changes involve API tests, agent features or modifications involve functional tests, and Neutron-wide features involve fullstack or scenario tests as appropriate.

The table references tests that explicitly target a feature, and not a job that is configured to run against a specific backend (Thereby testing it implicitly). So, for example, while the Linux bridge agent has a job that runs the API and scenario tests with the Linux bridge agent configured, it does not have functional tests that target the agent explicitly. The gate column is about running API/scenario tests with Neutron configured in a certain way, such as what L2 agent to use or what type of routers to create.

- V - Merged
- Blank - Not applicable
- X - Absent or lacking
- Patch number - Currently in review
- A name - That person has committed to work on an item
- Implicit - The code is executed, yet no assertions are made

Area	Unit	Functional	API	Fullstack	Scenario	Gate
DVR	V	L3-V OVS-X	V	X	X	V
L3 HA	V	V	X	286087	X	X
L2pop	V	X		Implicit		
DHCP HA	V			amuller		
OVS ARP responder	V	X*		Implicit		
OVS agent	V	V		V		V
Linux Bridge agent	V	X		V		V
Metering	V	X	V	X		
DHCP agent	V	V		amuller		V
rpc_workers						X
Reference ipam driver	V					X
MTU advertisement	V			X		
VLAN transparency	V		X	X		
Prefix delegation	V	X		X		

- Prefix delegation doesnt have functional tests for the dibbler and pd layers, nor for the L3 agent changes. This has been an area of repeated regressions.
- The functional job now compiles OVS 2.5 from source, enabling testing features that we previously could not.

Missing Infrastructure

The following section details missing test *types*. If you want to pick up an action item, please contact amuller for more context and guidance.

- The Neutron team would like Rally to persist results over a window of time, graph and visualize this data, so that reviewers could compare average runs against a proposed patch.
- Its possible to test RPC methods via the unit tests infrastructure. This was proposed in patch 162811. The goal is provide developers a light weight way to rapidly run tests that target the RPC layer, so that a patch that modifies an RPC methods signature could be verified quickly and locally.
- Neutron currently runs a partial-grenade job that verifies that an OVS version from the latest stable release works with neutron-server from master. We would like to expand this to DHCP and L3 agents as well.

Template for ModelMigrationSync for external repos

This section contains a template for a test which checks that the Python models for database tables are synchronized with the alembic migrations that create the database schema. This test should be implemented in all driver/plugin repositories that were split out from Neutron.

What does the test do?

This test compares models with the result of existing migrations. It is based on ModelsMigrationsSync which is provided by oslo.db and was adapted for Neutron. It compares core Neutron models and vendor specific models with migrations from Neutron core and migrations from the driver/plugin repo. This test is functional - it runs against MySQL and PostgreSQL dialects. The detailed description of this test can be found in Neutron Database Layer section - *Tests to verify that database migrations and models are in sync*.

Steps for implementing the test

1. Import all models in one place

Create a module `networking_foo/db/models/head.py` with the following content:

```
from neutron_lib.db import model_base

from networking_foo import models  # noqa
# Alternatively, import separate modules here if the models are not in one
# models.py file


def get_metadata():
    return model_base.BASEV2.metadata
```

2. Implement the test module

The test uses external.py from Neutron. This file contains lists of table names, which were moved out of Neutron:

```
VPNAAS_TABLES = [...]

FWAAS_TABLES = [...]

# Arista ML2 driver Models moved to openstack/networking-arista
REPO_ARISTA_TABLES = [...]

# Models moved to openstack/networking-cisco
REPO_CISCO_TABLES = [...]

...

TABLES = (FWAAS_TABLES + VPNAAS_TABLES + ...
          + REPO_ARISTA_TABLES + REPO_CISCO_TABLES)
```

Also the test uses **VERSION_TABLE**, it is the name of table in database which contains revision id of head migration. It is preferred to keep this variable in `networking_foo/db/migration/alembic_migrations/__init__.py` so it will be easy to use in test.

Create a module `networking_foo/tests/functional/db/test_migrations.py` with the following content:

```
from oslo_config import cfg

from neutron.db.migration.alembic_migrations import external
from neutron.db.migration import cli as migration
from neutron.tests.functional.db import test_migrations
from neutron.tests.unit import testlib_api

from networking_foo.db.migration import alembic_migrations
from networking_foo.db.models import head

# EXTERNAL_TABLES should contain all names of tables that are not related␣
↪to
# current repo.
EXTERNAL_TABLES = set(external.TABLES) - set(external.REPO_FOO_TABLES)


class _TestModelsMigrationsFoo(test_migrations._TestModelsMigrations):

    def db_sync(self, engine):
        cfg.CONF.set_override('connection', engine.url, group='database')
        for conf in migration.get_alembic_configs():
            self.alembic_config = conf
            self.alembic_config.neutron_config = cfg.CONF
            migration.do_alembic_command(conf, 'upgrade', 'heads')

    def get_metadata(self):
        return head.get_metadata()

    def include_object(self, self, object_, name, type_, reflected, compare_to):
```

(continues on next page)

(continued from previous page)

```
        if type_ == 'table' and (name == 'alembic' or
                                 name == alembic_migrations.VERSION_TABLE or
                                 name in EXTERNAL_TABLES):
            return False
        else:
            return True


class TestModelsMigrationsMysql(testlib_api.MySQLTestCaseMixin,
                                _TestModelsMigrationsFoo,
                                testlib_api.SqlTestCaseLight):
    pass


class TestModelsMigrationsPsql(testlib_api.PostgreSQLTestCaseMixin,
                               _TestModelsMigrationsFoo,
                               testlib_api.SqlTestCaseLight):
    pass
```

3. Add functional requirements

A separate file `networking_foo/tests/functional/requirements.txt` should be created containing the following requirements that are needed for successful test execution.

```
psutil>=3.2.2 # BSD
psycopg2
PyMySQL>=0.6.2  # MIT License
```

Example implementation in VPNaaS

Transient DB Failure Injection

Neutron has a service plugin to inject random delays and Deadlock exceptions into normal Neutron operations. The service plugin is called Loki and is located under neutron.services.loki.loki_plugin.

To enable the plugin, just add loki to the list of service_plugins in your neutron-server neutron.conf file.

The plugin will inject a Deadlock exception on database flushes with a 1/50 probability and a delay of 1 second with a 1/200 probability when SQLAlchemy objects are loaded into the persistent state from the DB. The goal is to ensure the code is tolerant of these transient delays/failures that will be experienced in busy production (and Galera) systems.

Neutron jobs running in Zuul CI

Tempest jobs running in Neutron CI

In upstream Neutron CI there are various tempest and neutron-tempest-plugin jobs running. Each of those jobs runs on slightly different configuration of Neutron services. Below is a summary of those jobs.

```
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
| Job name                                     | Run tests                  ␣
↪       | python   | nodes | L2 agent     | firewall        | L3 agent | L3␣
↪HA  | L3 DVR | enable_dvr | Run in gate |
|                                              |                            ␣
↪       | version  |       |              | driver          | mode     |   ␣
↪   |        |            | queue       |
+==============================================+======================================+======
|neutron-tempest-plugin-api                    |neutron_tempest_plugin.api␣
↪       |   3.6    |   1   | openvswitch | openvswitch     | legacy   |␣
↪False | False   | True          | Yes         |
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-plugin-designate-scenario     |neutron_tempest_plugin.
↪scenario.\ |    3.6    |   1    | openvswitch | openvswitch     | legacy   ␣
↪| False | False   | True          | No          |
|                                              |test_dns_integration        ␣
↪       |          |       |              |                 |          |   ␣
↪   |        |            |             |
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-plugin-dvr-multinode-scenario |neutron_tempest_plugin.
↪scenario    |    3.6    |   2    | openvswitch | openvswitch     | dvr_snat␣
↪| False | True    | True          | No          |
|(non-voting)                                  |                            ␣
↪       |          |       |              |                 | dvr_snat |   ␣
↪   |        |            |             |
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-plugin-scenario-linuxbridge   |neutron_tempest_plugin.
↪scenario    |    3.6    |   1    | linuxbridge | iptables        | legacy   ␣
↪| False | False   | False         | Yes         |
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-plugin-scenario-openvswitch   |neutron_tempest_plugin.
↪scenario    |    3.6    |   1    | openvswitch | openvswitch     | legacy   ␣
↪| False | False   | False         | Yes         |
+----------------------------------------------+----------------------------
↪-------+----------+-------+--------------+-----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-plugin-scenario-openvswitch-\ |neutron_tempest_plugin.
↪scenario    |    3.6    |   1    | openvswitch | iptables_hybrid | legacy   ␣
↪| False | False   | False         | Yes         |
```

(continues on next page)

(continued from previous page)

```
|  iptables_hybrid                                |                                                                    ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
+-----------------------------------------------+-------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+-----------+-------------+
|neutron-tempest-plugin-scenario-ovn             | neutron_tempest_plugin.
↪scenario  |    3.6    |    1    | ovn          | ovn              | ---      |␣
↪False | False   | False        | Yes          |
+-----------------------------------------------+-------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+-----------+-------------+
|neutron-tempest-linuxbridge                    |tempest.api (without slow␣
↪tests)  |    3.6    |    1    | linuxbridge | iptables         | legacy   |␣
↪False | False   | True         | Yes          |
|                                               |tempest.scenario                ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
|                                               |(only tests related to          ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
|                                               |Neutron and Nova)               ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
+-----------------------------------------------+-------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+-----------+-------------+
|neutron-tempest-multinode-full-py3             |tempest.api (without slow␣
↪tests)  |    3.6    |    2    | openvswitch | openvswitch      | legacy   |␣
↪False | False   | True         | Yes          |
|                                               |tempest.scenario                ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
|                                               |(only tests related to          ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
|                                               |Neutron and Nova)               ␣
↪        |         |       |             |                |          |    ␣
↪   |        |           |             |
+-----------------------------------------------+-------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+-----------+-------------+
|neutron-tempest-dvr-ha-multinode-full          |tempest.api (without slow␣
↪tests)  |    3.6    |    3    | openvswitch | openvswitch      | dvr      |␣
↪True  | True    | True         | No           |
|(non-voting)                                   |tempest.scenario                ␣
↪        |         |       |             |                | dvr_snat |    ␣
↪   |        |           |             |
|                                               |                                ␣
↪        |         |       |             |                | dvr_snat |    ␣
↪   |        |           |             |
+-----------------------------------------------+-------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+-----------+-------------+
|neutron-tempest-slow-py3                       |tempest slow tests              ␣
↪        |   3.6   |    2    | openvswitch | openvswitch      | legacy   |␣
↪False | False   | True         | Yes          |
```

(continues on next page)

(continued from previous page)

```
+----------------------------------------------+---------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-with-uwsgi                  |tempest.api (without slow␣
↪tests)  |    3.6    |    1    | openvswitch | openvswitch      | legacy    |␣
↪False | False   | True          | Yes          |
|                                            |tempest.scenario          ␣
↪       |          |        |             |                |          |  ␣
↪   |        |           |              |
|                                            |(only tests related to    ␣
↪       |          |        |             |                |          |  ␣
↪   |        |           |              |
|                                            |Neutron and Nova)         ␣
↪       |          |        |             |                |          |  ␣
↪   |        |           |              |
+----------------------------------------------+---------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+------------+-------------+
|neutron-tempest-ipv6-only                   |tempest smoke + IPv6 tests␣
↪       |    3.6    |    1    | openvswitch | openvswitch      | legacy    |␣
↪False | False   | True          | Yes          |
|                                            |(only tests related to    ␣
↪       |          |        |             |                |          |  ␣
↪   |        |           |              |
|                                            |Neutron and Nova)         ␣
↪       |          |        |             |                |          |  ␣
↪   |        |           |              |
+----------------------------------------------+---------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+------------+-------------+
|neutron-ovn-tempest-ovs-release             |Various tempest api,␣
↪scenario      |    3.6    |    1    | ovn          | ovn              | ---      ␣
↪ | False | False   | True          | Yes          |
+----------------------------------------------+---------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+------------+-------------+
|neutron-ovn-tempest-slow                    |tempest slow tests        ␣
↪       |    3.6    |    2    | ovn          | ovn              | ---       |␣
↪False | False   | True          | No           |
+----------------------------------------------+---------------------------
↪-------+---------+-------+-------------+----------------+----------+----
↪---+--------+------------+-------------+
```

Grenade jobs running in Neutron CI

In upstream Neutron CI there are various Grenade jobs running. Each of those jobs runs on slightly different configuration of Neutron services. Below is summary of those jobs.

```
+-----------------------------------+----------+--------+-------------+---------
↪----+----------+-------+--------+-------------+-------------+
| Job name                          | python   | nodes | L2 agent    |␣
↪firewall    | L3 agent | L3 HA | L3 DVR | enable_dvr | Run in gate |
|                                   | version |        |             | driver ␣
↪    | mode     |       |        |             | queue        |
```

(continues on next page)

(continued from previous page)

```
+================================+=========+=======+=============+=============+========
| neutron-grenade-multinode      |   3.6   |   2   | openvswitch |␣
↪openvswitch | legacy    | False | False   | True          | Yes           |
+--------------------------------+---------+-------+-------------+---------
↪----+----------+-------+--------+------------+-------------+
| neutron-grenade-dvr-multinode  |   3.6   |   2   | openvswitch |␣
↪openvswitch | dvr       | False | False   | True          | Yes           |
|                                |         |       |             |            ␣
↪    | dvr_snat |       |        |            |             |
+--------------------------------+---------+-------+-------------+---------
↪----+----------+-------+--------+------------+-------------+
| neutron-grenade-ovn            |   3.6   |   2   | OVN         | ---        ␣
↪    | ---      | ---   | ---    | False      | no          |
| (non-voting)                   |         |       |             |            ␣
↪    |          |       |        |            |             |
+--------------------------------+---------+-------+-------------+---------
↪----+----------+-------+--------+------------+-------------+
```

Columns description

- L2 agent - agent used on nodes in test job,
- firewall driver - driver configured in L2 agents config,
- L3 agent mode - mode(s) configured for L3 agent(s) on test nodes,
- L3 HA - value of `l3_ha` option set in `neutron.conf`,
- L3 DVR - value of `router_distributed` option set in `neutron.conf`,
- enable_dvr - value of `enable_dvr` option set in `neutron.conf`

Testing OVN with DevStack

This document describes how to test OpenStack with OVN using DevStack. We will start by describing how to test on a single host.

Single Node Test Environment

1. Create a test system.

Its best to use a throwaway dev system for running DevStack. Your best bet is to use either CentOS 8 or the latest Ubuntu LTS (18.04, Bionic).

2. Create the `stack` user.

```
$ git clone https://opendev.org/openstack/devstack.git
$ sudo ./devstack/tools/create-stack-user.sh
```

3. Switch to the `stack` user and clone DevStack and Neutron.

```
$ sudo su - stack
$ git clone https://opendev.org/openstack/devstack.git
$ git clone https://opendev.org/openstack/neutron.git
```

4. Configure DevStack to use the OVN driver.

OVN driver comes with a sample DevStack configuration file you can start with. For example, you may want to set some values for the various PASSWORD variables in that file so DevStack doesnt have to prompt you for them. Feel free to edit it if youd like, but it should work as-is.

```
$ cd devstack
$ cp ../neutron/devstack/ovn-local.conf.sample local.conf
```

5. Run DevStack.

This is going to take a while. It installs a bunch of packages, clones a bunch of git repos, and installs everything from these git repos.

```
$ ./stack.sh
```

Once DevStack completes successfully, you should see output that looks something like this:

```
This is your host IP address: 172.16.189.6
This is your host IPv6 address: ::1
Horizon is now available at http://172.16.189.6/dashboard
Keystone is serving at http://172.16.189.6/identity/
The default users are: admin and demo
The password: password
2017-03-09 15:10:54.117 | stack.sh completed in 2110 seconds.
```

Environment Variables

Once DevStack finishes successfully, were ready to start interacting with OpenStack APIs. OpenStack provides a set of command line tools for interacting with these APIs. DevStack provides a file you can source to set up the right environment variables to make the OpenStack command line tools work.

```
$ . openrc
```

If youre curious what environment variables are set, they generally start with an OS prefix:

```
$ env | grep OS
OS_REGION_NAME=RegionOne
OS_IDENTITY_API_VERSION=2.0
OS_PASSWORD=password
OS_AUTH_URL=http://192.168.122.8:5000/v2.0
OS_USERNAME=demo
OS_TENANT_NAME=demo
OS_VOLUME_API_VERSION=2
OS_CACERT=/opt/stack/data/CA/int-ca/ca-chain.pem
OS_NO_CACHE=1
```

Default Network Configuration

By default, DevStack creates networks called `private` and `public`. Run the following command to see the existing networks:

```
$ openstack network list
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+
| ID                                   | Name    | Subnets                ␣
↪                                                    |
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+
| 40080dad-0064-480a-b1b0-592ae51c1471 | private | 5ff81545-7939-4ae0-8365-
↪1658d45fa85c, da34f952-3bfc-45bb-b062-d2d973c1a751 |
| 7ec986dd-aae4-40b5-86cf-8668feeeab67 | public  | 60d0c146-a29b-4cd3-bd90-
↪3745603b1a4b, f010c309-09be-4af2-80d6-e6af9c78bae7 |
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+
```

A Neutron network is implemented as an OVN logical switch. OVN driver creates logical switches with a name in the format neutron-<network UUID>. We can use `ovn-nbctl` to list the configured logical switches and see that their names correlate with the output from `openstack network list`:

```
$ ovn-nbctl ls-list
71206f5c-b0e6-49ce-b572-eb2e964b2c4e (neutron-40080dad-0064-480a-b1b0-
↪592ae51c1471)
8d8270e7-fd51-416f-ae85-16565200b8a4 (neutron-7ec986dd-aae4-40b5-86cf-
↪8668feeeab67)

$ ovn-nbctl get Logical_Switch neutron-40080dad-0064-480a-b1b0-
↪592ae51c1471 external_ids
{"neutron:network_name"=private}
```

Booting VMs

In this section well go through the steps to create two VMs that have a virtual NIC attached to the `private` Neutron network.

DevStack uses libvirt as the Nova backend by default. If KVM is available, it will be used. Otherwise, it will just run qemu emulated guests. This is perfectly fine for our testing, as we only need these VMs to be able to send and receive a small amount of traffic so performance is not very important.

1. Get the Network UUID.

Start by getting the UUID for the `private` network from the output of `openstack network list` from earlier and save it off:

```
$ PRIVATE_NET_ID=$(openstack network show private -c id -f value)
```

2. Create an SSH keypair.

Next create an SSH keypair in Nova. Later, when we boot a VM, well ask that the public key be put in the VM so we can SSH into it.

```
$ openstack keypair create demo > id_rsa_demo
$ chmod 600 id_rsa_demo
```

3. Choose a flavor.

We need minimal resources for these test VMs, so the `m1.nano` flavor is sufficient.

```
$ openstack flavor list
+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 1  | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3  | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4  | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 42 | m1.nano   |    64 |    0 |         0 |     1 | True      |
| 5  | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
| 84 | m1.micro  |   128 |    0 |         0 |     1 | True      |
| c1 | cirros256 |   256 |    0 |         0 |     1 | True      |
| d1 | ds512M    |   512 |    5 |         0 |     1 | True      |
| d2 | ds1G      |  1024 |   10 |         0 |     1 | True      |
| d3 | ds2G      |  2048 |   10 |         0 |     2 | True      |
| d4 | ds4G      |  4096 |   20 |         0 |     4 | True      |
+----+-----------+-------+------+-----------+-------+-----------+

$ FLAVOR_ID=$(openstack flavor show m1.nano -c id -f value)
```

4. Choose an image.

DevStack imports the CirrOS image by default, which is perfect for our testing. Its a very small test image.

```
$ openstack image list
+--------------------------------------+--------------------------+--------
↪+
| ID                                   | Name                     | Status␣
↪|
+--------------------------------------+--------------------------+--------
↪+
| 849a8db2-3754-4cf6-9271-491fa4ff7195 | cirros-0.3.5-x86_64-disk | active␣
↪|
+--------------------------------------+--------------------------+--------
↪+

$ IMAGE_ID=$(openstack image list -c ID -f value)
```

5. Setup a security rule so that we can access the VMs we will boot up next.

By default, DevStack does not allow users to access VMs, to enable that, we will need to add a rule. We will allow both ICMP and SSH.

```
$ openstack security group rule create --ingress --ethertype IPv4 --dst-
↪port 22 --protocol tcp default
$ openstack security group rule create --ingress --ethertype IPv4 --
↪protocol ICMP default
$ openstack security group rule list
+--------------------------------------+-------------+-----------+---------
↪---+--------------------------------------+---------------
↪--------+
```

(continues on next page)

(continued from previous page)

```
| ID                                   | IP Protocol | IP Range  | Port␣
↪Range | Remote Security Group                   | Security Group                   ␣
↪             |
+--------------------------------------+-------------+-----------+---------
↪---+--------------------------------------+------------------------------
↪--------+
...
| ade97198-db44-429e-9b30-24693d86d9b1 | tcp         | 0.0.0.0/0 | 22:22     ␣
↪    | None                                | a47b14da-5607-404a-8de4-
↪3a0f1ad3649c |
| d0861a98-f90e-4d1a-abfb-827b416bc2f6 | icmp        | 0.0.0.0/0 |           ␣
↪    | None                                | a47b14da-5607-404a-8de4-
↪3a0f1ad3649c |
...
+--------------------------------------+-------------+-----------+---------
↪---+--------------------------------------+------------------------------
↪--------+
```

6. Boot some VMs.

Now we will boot two VMs. Well name them `test1` and `test2`.

```
$ openstack server create --nic net-id=$PRIVATE_NET_ID --flavor $FLAVOR_ID␣
↪--image $IMAGE_ID --key-name demo test1
+------------------------------+--------------------------------------------
↪---------------------+
| Field                        | Value                                      ␣
↪                     |
+------------------------------+--------------------------------------------
↪---------------------+
| OS-DCF:diskConfig            | MANUAL                                     ␣
↪                     |
| OS-EXT-AZ:availability_zone  |                                            ␣
↪                     |
| OS-EXT-STS:power_state       | NOSTATE                                    ␣
↪                     |
| OS-EXT-STS:task_state        | scheduling                                 ␣
↪                     |
| OS-EXT-STS:vm_state          | building                                   ␣
↪                     |
| OS-SRV-USG:launched_at       | None                                       ␣
↪                     |
| OS-SRV-USG:terminated_at     | None                                       ␣
↪                     |
| accessIPv4                   |                                            ␣
↪                     |
| accessIPv6                   |                                            ␣
↪                     |
| addresses                    |                                            ␣
↪                     |
| adminPass                    | BzAWWA6byGP6                               ␣
↪                     |
| config_drive                 |                                            ␣
↪                     |
| created                      | 2017-03-09T16:56:08Z                       ␣
↪                     |
```

(continues on next page)

(continued from previous page)

```
| flavor                      | m1.nano (42)                                ␣
↪                    |
| hostId                      |                                             ␣
↪                    |
| id                          | d8b8084e-58ff-44f4-b029-a57e7ef6ba61        ␣
↪                    |
| image                       | cirros-0.3.5-x86_64-disk (849a8db2-3754-
↪4cf6-9271-491fa4ff7195) |
| key_name                    | demo                                        ␣
↪                    |
| name                        | test1                                       ␣
↪                    |
| progress                    | 0                                           ␣
↪                    |
| project_id                  | b6522570f7344c06b1f24303abf3c479            ␣
↪                    |
| properties                  |                                             ␣
↪                    |
| security_groups             | name='default'                              ␣
↪                    |
| status                      | BUILD                                       ␣
↪                    |
| updated                     | 2017-03-09T16:56:08Z                        ␣
↪                    |
| user_id                     | c68f77f1d85e43eb9e5176380a68ac1f            ␣
↪                    |
| volumes_attached            |                                             ␣
↪                    |
+-----------------------------+---------------------------------------------
↪---------------------+

$ openstack server create --nic net-id=$PRIVATE_NET_ID --flavor $FLAVOR_ID␣
↪--image $IMAGE_ID --key-name demo test2
+-----------------------------+---------------------------------------------
↪---------------------+
| Field                       | Value                                       ␣
↪                    |
+-----------------------------+---------------------------------------------
↪---------------------+
| OS-DCF:diskConfig           | MANUAL                                      ␣
↪                    |
| OS-EXT-AZ:availability_zone |                                             ␣
↪                    |
| OS-EXT-STS:power_state      | NOSTATE                                     ␣
↪                    |
| OS-EXT-STS:task_state       | scheduling                                  ␣
↪                    |
| OS-EXT-STS:vm_state         | building                                    ␣
↪                    |
| OS-SRV-USG:launched_at      | None                                        ␣
↪                    |
| OS-SRV-USG:terminated_at    | None                                        ␣
↪                    |
| accessIPv4                  |                                             ␣
↪                    |
| accessIPv6                  |                                             ␣
↪                    |
```

(continues on next page)

(continued from previous page)

```
| addresses                    |                                            ␣
↪                      |
| adminPass                    | YB8dmt5v88JV                               ␣
↪                      |
| config_drive                 |                                            ␣
↪                      |
| created                      | 2017-03-09T16:56:50Z                       ␣
↪                      |
| flavor                       | m1.nano (42)                               ␣
↪                      |
| hostId                       |                                            ␣
↪                      |
| id                           | 170d4f37-9299-4a08-b48b-2b90fce8e09b       ␣
↪                      |
| image                        | cirros-0.3.5-x86_64-disk (849a8db2-3754-
↪4cf6-9271-491fa4ff7195) |
| key_name                     | demo                                       ␣
↪                      |
| name                         | test2                                      ␣
↪                      |
| progress                     | 0                                          ␣
↪                      |
| project_id                   | b6522570f7344c06b1f24303abf3c479           ␣
↪                      |
| properties                   |                                            ␣
↪                      |
| security_groups              | name='default'                             ␣
↪                      |
| status                       | BUILD                                      ␣
↪                      |
| updated                      | 2017-03-09T16:56:51Z                       ␣
↪                      |
| user_id                      | c68f77f1d85e43eb9e5176380a68ac1f           ␣
↪                      |
| volumes_attached             |                                            ␣
↪                      |
+------------------------------+--------------------------------------------
↪----------------------+
```

Once both VMs have been started, they will have a status of `ACTIVE`:

```
$ openstack server list
+--------------------------------------+-------+--------+-------------------
↪---------------------------------------+-------------------------+
| ID                                   | Name  | Status | Networks          ␣
↪                                         | Image Name              |
+--------------------------------------+-------+--------+-------------------
↪---------------------------------------+-------------------------+
| 170d4f37-9299-4a08-b48b-2b90fce8e09b | test2 | ACTIVE |␣
↪private=fd5d:9d1b:457c:0:f816:3eff:fe24:49df, 10.0.0.3  | cirros-0.3.5-
↪x86_64-disk |
| d8b8084e-58ff-44f4-b029-a57e7ef6ba61 | test1 | ACTIVE |␣
↪private=fd5d:9d1b:457c:0:f816:3eff:fe3f:953d, 10.0.0.10 | cirros-0.3.5-
↪x86_64-disk |
+--------------------------------------+-------+--------+-------------------
↪---------------------------------------+-------------------------+
```

Our two VMs have addresses of `10.0.0.3` and `10.0.0.10`. If we list Neutron ports, there are two new ports with these addresses associated with them:

```
$ openstack port list
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                             ␣
↪                    | Status |
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+
...
| 97c970b0-485d-47ec-868d-783c2f7acde3 |      | fa:16:3e:3f:95:3d | ip_
↪address='10.0.0.10', subnet_id='da34f952-3bfc-45bb-b062-d2d973c1a751'    ␣
↪                       | ACTIVE |
|                                      |      |                   | ip_
↪address='fd5d:9d1b:457c:0:f816:3eff:fe3f:953d', subnet_id='5ff81545-7939-
↪4ae0-8365-1658d45fa85c'  |        |
| e003044d-334a-4de3-96d9-35b2d2280454 |      | fa:16:3e:24:49:df | ip_
↪address='10.0.0.3', subnet_id='da34f952-3bfc-45bb-b062-d2d973c1a751'     ␣
↪                       | ACTIVE |
|                                      |      |                   | ip_
↪address='fd5d:9d1b:457c:0:f816:3eff:fe24:49df', subnet_id='5ff81545-7939-
↪4ae0-8365-1658d45fa85c'  |        |
...
+--------------------------------------+------+-------------------+--------
↪-------------------------------------------------------------------------
↪--------------------+--------+

$ TEST1_PORT_ID=97c970b0-485d-47ec-868d-783c2f7acde3
$ TEST2_PORT_ID=e003044d-334a-4de3-96d9-35b2d2280454
```

Now we can look at OVN using `ovn-nbctl` to see the logical switch ports that were created for these two Neutron ports. The first part of the output is the OVN logical switch port UUID. The second part in parentheses is the logical switch port name. Neutron sets the logical switch port name equal to the Neutron port ID.

```
$ ovn-nbctl lsp-list neutron-$PRIVATE_NET_ID
...
fde1744b-e03b-46b7-b181-abddcbe60bf2 (97c970b0-485d-47ec-868d-783c2f7acde3)
7ce284a8-a48a-42f5-bf84-b2bca62cd0fe (e003044d-334a-4de3-96d9-35b2d2280454)
...
```

These two ports correspond to the two VMs we created.

VM Connectivity

We can connect to our VMs by associating a floating IP address from the public network.

```
$ openstack floating ip create --port $TEST1_PORT_ID public
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-03-09T18:58:12Z                 |
| description         |                                      |
| fixed_ip_address    | 10.0.0.10                            |
| floating_ip_address | 172.24.4.8                           |
| floating_network_id | 7ec986dd-aae4-40b5-86cf-8668feeeab67 |
| id                  | 24ff0799-5a72-4a5b-abc0-58b301c9aee5 |
| name                | None                                 |
| port_id             | 97c970b0-485d-47ec-868d-783c2f7acde3 |
| project_id          | b6522570f7344c06b1f24303abf3c479     |
| revision_number     | 1                                    |
| router_id           | ee51adeb-0dd8-4da0-ab6f-7ce60e00e7b0 |
| status              | DOWN                                 |
| updated_at          | 2017-03-09T18:58:12Z                 |
+---------------------+--------------------------------------+
```

Devstack does not wire up the public network by default so we must do that before connecting to this floating IP address.

```
$ sudo ip link set br-ex up
$ sudo ip route add 172.24.4.0/24 dev br-ex
$ sudo ip addr add 172.24.4.1/24 dev br-ex
```

Now you should be able to connect to the VM via its floating IP address. First, ping the address.

```
$ ping -c 1 172.24.4.8
PING 172.24.4.8 (172.24.4.8) 56(84) bytes of data.
64 bytes from 172.24.4.8: icmp_seq=1 ttl=63 time=0.823 ms

--- 172.24.4.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.823/0.823/0.823/0.000 ms
```

Now SSH to the VM:

```
$ ssh -i id_rsa_demo cirros@172.24.4.8 hostname
test1
```

Adding Another Compute Node

After completing the earlier instructions for setting up devstack, you can use a second VM to emulate an additional compute node. This is important for OVN testing as it exercises the tunnels created by OVN between the hypervisors.

Just as before, create a throwaway VM but make sure that this VM has a different host name. Having same host name for both VMs will confuse Nova and will not produce two hypervisors when you query nova hypervisor list later. Once the VM is setup, create the `stack` user:

```
$ git clone https://opendev.org/openstack/devstack.git
$ sudo ./devstack/tools/create-stack-user.sh
```

Switch to the `stack` user and clone DevStack and neutron:

```
$ sudo su - stack
$ git clone https://opendev.org/openstack/devstack.git
$ git clone https://opendev.org/openstack/neutron.git
```

OVN comes with another sample configuration file that can be used for this:

```
$ cd devstack
$ cp ../neutron/devstack/ovn-compute-local.conf.sample local.conf
```

You must set SERVICE_HOST in local.conf. The value should be the IP address of the main DevStack host. You must also set HOST_IP to the IP address of this new host. See the text in the sample configuration file for more information. Once that is complete, run DevStack:

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

This should complete in less time than before, as its only running a single OpenStack service (nova-compute) along with OVN (ovn-controller, ovs-vswitchd, ovsdb-server). The final output will look something like this:

```
This is your host IP address: 172.16.189.30
This is your host IPv6 address: ::1
2017-03-09 18:39:27.058 | stack.sh completed in 1149 seconds.
```

Now go back to your main DevStack host. You can use admin credentials to verify that the additional hypervisor has been added to the deployment:

```
$ cd devstack
$ . openrc admin
$ ./tools/discover_hosts.sh
$ openstack hypervisor list
+----+------------------------+-----------------+----------------+-------+
| ID | Hypervisor Hostname    | Hypervisor Type | Host IP        | State |
+----+------------------------+-----------------+----------------+-------+
|  1 | centos7-ovn-devstack   | QEMU            | 172.16.189.6   | up    |
|  2 | centos7-ovn-devstack-2 | QEMU            | 172.16.189.30  | up    |
+----+------------------------+-----------------+----------------+-------+
```

You can also look at OVN and OVS to see that the second host has shown up. For example, there will be a second entry in the Chassis table of the OVN_Southbound database. You can use the `ovn-sbctl` utility to list chassis, their configuration, and the ports bound to each of them:

```
$ ovn-sbctl show

Chassis "ddc8991a-d838-4758-8d15-71032da9d062"
    hostname: "centos7-ovn-devstack"
    Encap vxlan
        ip: "172.16.189.6"
        options: {csum="true"}
    Encap geneve
```

(continues on next page)

(continued from previous page)

```
        ip: "172.16.189.6"
        options: {csum="true"}
    Port_Binding "97c970b0-485d-47ec-868d-783c2f7acde3"
    Port_Binding "e003044d-334a-4de3-96d9-35b2d2280454"
    Port_Binding "cr-lrp-08d1f28d-cc39-4397-b12b-7124080899a1"
Chassis "b194d07e-0733-4405-b795-63b172b722fd"
    hostname: "centos7-ovn-devstack-2.os1.phx2.redhat.com"
    Encap geneve
        ip: "172.16.189.30"
        options: {csum="true"}
    Encap vxlan
        ip: "172.16.189.30"
        options: {csum="true"}
```

You can also see a tunnel created to the other compute node:

```
$ ovs-vsctl show
...
Bridge br-int
    fail_mode: secure
    ...
    Port "ovn-b194d0-0"
        Interface "ovn-b194d0-0"
            type: geneve
            options: {csum="true", key=flow, remote_ip="172.16.189.30"}
    ...
...
```

Provider Networks

Neutron has a provider networks API extension that lets you specify some additional attributes on a network. These attributes let you map a Neutron network to a physical network in your environment. The OVN ML2 driver is adding support for this API extension. It currently supports flat and vlan networks.

Here is how you can test it:

First you must create an OVS bridge that provides connectivity to the provider network on every host running ovn-controller. For trivial testing this could just be a dummy bridge. In a real environment, you would want to add a local network interface to the bridge, as well.

```
$ ovs-vsctl add-br br-provider
```

ovn-controller on each host must be configured with a mapping between a network name and the bridge that provides connectivity to that network. In this case well create a mapping from the network name providernet to the bridge br-provider.

```
$ ovs-vsctl set open . \
external-ids:ovn-bridge-mappings=providernet:br-provider
```

If you want to enable this chassis to host a gateway router for external connectivity, then set ovn-cms-options to enable-chassis-as-gw.

```
$ ovs-vsctl set open . \
external-ids:ovn-cms-options="enable-chassis-as-gw"
```

Now create a Neutron provider network.

```
$ openstack network create provider --share \
--provider-physical-network providernet \
--provider-network-type flat
```

Alternatively, you can define connectivity to a VLAN instead of a flat network:

```
$ openstack network create provider-101 --share \
--provider-physical-network providernet \
--provider-network-type vlan
--provider-segment 101
```

Observe that the OVN ML2 driver created a special logical switch port of type localnet on the logical switch to model the connection to the physical network.

```
$ ovn-nbctl show
...
 switch 5bbccbbd-f5ca-411b-bad9-01095d6f1316 (neutron-729dbbee-db84-4a3d-
↪afc3-82c0b3701074)
     port provnet-729dbbee-db84-4a3d-afc3-82c0b3701074
         addresses: ["unknown"]
...

$ ovn-nbctl lsp-get-type provnet-729dbbee-db84-4a3d-afc3-82c0b3701074
localnet

$ ovn-nbctl lsp-get-options provnet-729dbbee-db84-4a3d-afc3-82c0b3701074
network_name=providernet
```

If VLAN is used, there will be a VLAN tag shown on the localnet port as well.

Finally, create a Neutron port on the provider network.

```
$ openstack port create --network provider myport
```

or if you followed the VLAN example, it would be:

```
$ openstack port create --network provider-101 myport
```

Skydive

Skydive is an open source real-time network topology and protocols analyzer. It aims to provide a comprehensive way of understanding what is happening in the network infrastructure. Skydive works by utilizing agents to collect host-local information, and sending this information to a central agent for further analysis. It utilizes elasticsearch to store the data.

To enable Skydive support with OVN and devstack, enable it on the control and compute nodes.

On the control node, enable it as follows:

```
enable_plugin skydive https://github.com/skydive-project/skydive.git
enable_service skydive-analyzer
```

On the compute nodes, enable it as follows:

```
enable_plugin skydive https://github.com/skydive-project/skydive.git
enable_service skydive-agent
```

Troubleshooting

If you run into any problems, take a look at our *Troubleshooting* page.

Additional Resources

See the documentation and other references linked from the *OVN information* page.

14.5 Neutron Internals

14.5.1 Neutron Internals

Subnet Pools and Address Scopes

This page discusses subnet pools and address scopes

Subnet Pools

Learn about subnet pools by watching the summit talk given in Vancouver[1].

Subnet pools were added in Kilo. They are relatively simple. A SubnetPool has any number of SubnetPoolPrefix objects associated to it. These prefixes are in CIDR format. Each CIDR is a piece of the address space that is available for allocation.

Subnet Pools support IPv6 just as well as IPv4.

The Subnet model object now has a subnetpool_id attribute whose default is null for backward compatibility. The subnetpool_id attribute stores the UUID of the subnet pool that acted as the source for the address range of a particular subnet.

When creating a subnet, the subnetpool_id can be optionally specified. If it is, the cidr field is not required. If cidr is specified, it will be allocated from the pool assuming the pool includes it and hasnt already allocated any part of it. If cidr is left out, then the prefixlen attribute can be specified. If it is not, the default prefix length will be taken from the subnet pool. Think of it this way, the allocation logic always needs to know the size of the subnet desired. It can pull it from a specific CIDR, prefixlen, or default. A specific CIDR is optional and the allocation will try to honor it if provided. The request will fail if it cant honor it.

Subnet pools do not allow overlap of subnets.

[1] http://www.youtube.com/watch?v=QqP8yBUUXBM&t=6m12s

Subnet Pool Quotas

A quota mechanism was provided for subnet pools. It is different than other quota mechanisms in Neutron because it doesnt count instances of first class objects. Instead it counts how much of the address space is used.

For IPv4, it made reasonable sense to count quota in terms of individual addresses. So, if youre allowed exactly one /24, your quota should be set to 256. Three /26s would be 192. This mechanism encourages more efficient use of the IPv4 space which will be increasingly important when working with globally routable addresses.

For IPv6, the smallest viable subnet in Neutron is a /64. There is no reason to allocate a subnet of any other size for use on a Neutron network. It would look pretty funny to set a quota of 4611686018427387904 to allow one /64 subnet. To avoid this, we count IPv6 quota in terms of /64s. So, a quota of 3 allows three /64 subnets. When we need to allocate something smaller in the future, we will need to ensure that the code can handle non-integer quota consumption.

Allocation

Allocation is done in a way that aims to minimize fragmentation of the pool. The relevant code is here[2]. First, the available prefixes are computed using a set difference: pool - allocations. The result is compacted[3] and then sorted by size. The subnet is then allocated from the smallest available prefix that is large enough to accommodate the request.

Address Scopes

Before subnet pools or address scopes, it was impossible to tell if a network address was routable in a certain context because the address was given explicitly on subnet create and wasnt validated against any other addresses. Address scopes are meant to solve this by putting control over the address space in the hands of an authority: the address scope owner. It makes use of the already existing SubnetPool concept for allocation.

Address scopes are the thing within which address overlap is not allowed and thus provide more flexible control as well as decoupling of address overlap from tenancy.

Prior to the Mitaka release, there was implicitly only a single shared address scope. Arbitrary address overlap was allowed making it pretty much a free for all. To make things seem somewhat sane, normal users are not able to use routers to cross-plug networks from different projects and NAT was used between internal networks and external networks. It was almost as if each project had a private address scope.

The problem is that this model cannot support use cases where NAT is not desired or supported (e.g. IPv6) or we want to allow different projects to cross-plug their networks.

An AddressScope covers only one address family. But, they work equally well for IPv4 and IPv6.

[2] neutron/ipam/subnet_alloc.py (_allocate_any_subnet)

[3] http://pythonhosted.org/netaddr/api.html#netaddr.IPSet.compact

Routing

The reference implementation honors address scopes. Within an address scope, addresses route freely (barring any FW rules or other external restrictions). Between scopes, routing is prevented unless address translation is used.

For now, floating IPs are the only place where traffic crosses scope boundaries. When a floating IP is associated to a fixed IP, the fixed IP is allowed to access the address scope of the floating IP by way of a 1:1 NAT rule. That means the fixed IP can access not only the external network, but also any internal networks that are in the same address scope as the external network. This is diagrammed as follows:

```
+----------------------+      +---------------------------+
|    address scope 1   |      |       address scope 2     |
|                      |      |                           |
| +------------------+ |      |   +------------------+    |
| | internal network | |      |   | external network |    |
| +-------------+----+ |      |   +--------+---------+    |
|               |      |      |            |              |
|       +-------+--+   |      |     +------+------+       |
|       | fixed ip +-----------------+ floating IP |       |
|       +----------+   |      |     +--+--------+-+       |
+----------------------+      |        |        |         |
                              | +------+---+ +--+-------+ |
                              | | internal | | internal | |
                              | +----------+ +----------+ |
                              +---------------------------+
```

Due to the asymmetric route in DVR, and the fact that DVR local routers do not know the information of the floating IPs that reside in other hosts, there is a limitation in the DVR multiple hosts scenario. With DVR in multiple hosts, when the destination of traffic is an internal fixed IP in a different host, the fixed IP with a floating IP associated cant cross the scope boundary to access the internal networks that are in the same address scope of the external network. See https://bugs.launchpad.net/neutron/+bug/1682228

RPC

The L3 agent in the reference implementation needs to know the address scope for each port on each router in order to map ingress traffic correctly.

Each subnet from the same address family on a network is required to be from the same subnet pool. Therefore, the address scope will also be the same. If this were not the case, it would be more difficult to match ingress traffic on a port with the appropriate scope. It may be counter-intuitive but L3 address scopes need to be anchored to some sort of non-L3 thing (e.g. an L2 interface) in the topology in order to determine the scope of ingress traffic. For now, we use ports/networks. In the future, we may be able to distinguish by something else like the remote MAC address or something.

The address scope id is set on each port in a dict under the address_scopes attribute. The scope is distinct per address family. If the attribute does not appear, it is assumed to be null for both families. A value of null means that the addresses are in the implicit address scope which holds all addresses that dont have an explicit one. All subnets that existed in Neutron before address scopes existed fall here.

Here is an example of how the json will look in the context of a router port:

```
"address_scopes": {
    "4": "d010a0ea-660e-4df4-86ca-ae2ed96da5c1",
```

(continues on next page)

(continued from previous page)

```
    "6": null
},
```

To implement floating IPs crossing scope boundaries, the L3 agent needs to know the target scope of the floating ip. The fixed address is not enough to disambiguate because, theoretically, there could be overlapping addresses from different scopes. The scope is computed[4] from the floating ip fixed port and attached to the floating ip dict under the fixed_ip_address_scope attribute. Heres what the json looks like (trimmed):

```
{
     ...
     "floating_ip_address": "172.24.4.4",
     "fixed_ip_address": "172.16.0.3",
     "fixed_ip_address_scope": "d010a0ea-660e-4df4-86ca-ae2ed96da5c1",
     ...
}
```

Model

The model for subnet pools and address scopes can be found in neutron/db/models_v2.py and neutron/db/address_scope_db.py. This document wont go over all of the details. It is worth noting how they relate to existing Neutron objects. The existing Neutron subnet now optionally references a single subnet pool:

```
+----------------+          +------------------+           +--------------+
| Subnet         |          | SubnetPool       |           | AddressScope |
+----------------+          +------------------+           +--------------+
| subnet_pool_id +------> | address_scope_id +------> |              |
|                |          |                  |           |              |
|                |          |                  |           |              |
|                |          |                  |           |              |
+----------------+          +------------------+           +--------------+
```

L3 Agent

The L3 agent is limited in its support for multiple address scopes. Within a router in the reference implementation, traffic is marked on ingress with the address scope corresponding to the network it is coming from. If that traffic would route to an interface in a different address scope, the traffic is blocked unless an exception is made.

One exception is made for floating IP traffic. When traffic is headed to a floating IP, DNAT is applied and the traffic is allowed to route to the private IP address potentially crossing the address scope boundary. When traffic flows from an internal port to the external network and a floating IP is assigned, that traffic is also allowed.

Another exception is made for traffic from an internal network to the external network when SNAT is enabled. In this case, SNAT to the routers fixed IP address is applied to the traffic. However, SNAT is not used if the external network has an explicit address scope assigned and it matches the internal networks.

[4] neutron/db/l3_db.py (_get_sync_floating_ips)

In that case, traffic routes straight through without NAT. The internal networks addresses are viable on the external network in this case.

The reference implementation has limitations. Even with multiple address scopes, a router implementation is unable to connect to two networks with overlapping IP addresses. There are two reasons for this.

First, a single routing table is used inside the namespace. An implementation using multiple routing tables has been in the works but there are some unresolved issues with it.

Second, the default SNAT feature cannot be supported with the current Linux conntrack implementation unless a double NAT is used (one NAT to get from the address scope to an intermediate address specific to the scope and a second NAT to get from that intermediate address to an external address). Single NAT wont work if there are duplicate addresses across the scopes.

Due to these complications the router will still refuse to connect to overlapping subnets. We can look in to an implementation that overcomes these limitations in the future.

Agent extensions

All reference agents utilize a common extension mechanism that allows for the introduction and enabling of a core resource extension without needing to change agent code. This mechanism allows multiple agent extensions to be run by a single agent simultaneously. The mechanism may be especially interesting to third parties whose extensions lie outside the neutron tree.

Under this framework, an agent may expose its API to each of its extensions thereby allowing an extension to access resources internal to the agent. At layer 2, for instance, upon each port event the agent is then able to trigger a handle_port method in its extensions.

Interactions with the agent API object are in the following order:

1. The agent initializes the agent API object.
2. The agent passes the agent API object into the extension manager.
3. The manager passes the agent API object into each extension.
4. An extension calls the new agent API object method to receive, for instance, bridge wrappers with cookies allocated.

```
+-----------+
| Agent API +--------------------------------------------------+
+-----+-----+                                                  |
      |                                   +-----------+        |
      |1                              +---+ Extension +---+    |
      |                               |   +-----------+   |    |
+---+-+-+---+  2  +--------------+  3 |                   |  4 |
|   Agent   +-----+ Ext. manager +-----+---+   ....    +---+-----+
+-----------+     +--------------+    |                   |
                                      |   +-----------+   |
                                      +---+ Extension +---+
                                          +-----------+
```

Each extension is referenced through a stevedore entry point defined within a specific namespace. For example, L2 extensions are referenced through the neutron.agent.l2.extensions namespace.

The relevant modules are:

- neutron_lib.agent.extension: This module defines an abstract extension interface for all agent extensions across L2 and L3.
- neutron_lib.agent.l2_extension:
- neutron_lib.agent.l3_extension: These modules subclass neutron_lib.agent.extension.AgentExtension and define a layer-specific abstract extension interface.
- neutron.agent.agent_extensions_manager: This module contains a manager that allows extensions to load themselves at runtime.
- neutron.agent.l2.l2_agent_extensions_manager:
- neutron.agent.l3.l3_agent_extensions_manager: Each of these modules passes core resource events to loaded extensions.

Agent API object

Every agent can pass an agent API object into its extensions in order to expose its internals to them in a controlled way. To accommodate different agents, each extension may define a consume_api() method that will receive this object.

This agent API object is part of neutrons public interface for third parties. All changes to the interface will be managed in a backwards-compatible way.

At this time, on the L2 side, only the L2 Open vSwitch agent provides an agent API object to extensions. See *L2 agent extensions*. For L3, see *L3 agent extensions*.

The relevant modules are:

- neutron_lib.agent.extension
- neutron_lib.agent.l2_extension
- neutron_lib.agent.l3_extension
- neutron.agent.agent_extensions_manager
- neutron.agent.l2.l2_agent_extensions_manager
- neutron.agent.l3.l3_agent_extensions_manager

API Extensions

API extensions is the standard way of introducing new functionality to the Neutron project, it allows plugins to determine if they wish to support the functionality or not.

Examples

The easiest way to demonstrate how an API extension is written, is by studying an existing API extension and explaining the different layers.

Guided Tour: The Neutron Security Group API

https://wiki.openstack.org/wiki/Neutron/SecurityGroups

API Extension

The API extension is the front end portion of the code, which handles defining a REST-ful API, which is used by projects.

Database API

The Security Group API extension adds a number of methods to the database layer of Neutron

Agent RPC

This portion of the code handles processing requests from projects, after they have been stored in the database. It involves messaging all the L2 agents running on the compute nodes, and modifying the IPTables rules on each hypervisor.

- Plugin RPC classes
  - SecurityGroupServerRpcMixin - defines the RPC API that the plugin uses to communicate with the agents running on the compute nodes
  - SecurityGroupServerRpcMixin - Defines the API methods used to fetch data from the database, in order to return responses to agents via the RPC API
- Agent RPC classes
  - The SecurityGroupServerRpcApi defines the API methods that can be called by agents, back to the plugin that runs on the Neutron controller
  - The SecurityGroupAgentRpcCallbackMixin defines methods that a plugin uses to call back to an agent after performing an action called by an agent.

IPTables Driver

- `prepare_port_filter` takes a `port` argument, which is a `dictionary` object that contains information about the port - including the `security_group_rules`
- `prepare_port_filter` appends the port to an internal dictionary, `filtered_ports` which is used to track the internal state.
- Each security group has a chain in Iptables.

- The `IptablesFirewallDriver` has a method to convert security group rules into iptables statements.

Extensions for Resources with standard attributes

Resources that inherit from the HasStandardAttributes DB class can automatically have the extensions written for standard attributes (e.g. timestamps, revision number, etc) extend their resources by defining the api_collections on their model. These are used by extensions for standard attr resources to generate the extended resources map.

Any new addition of a resource to the standard attributes collection must be accompanied with a new extension to ensure that it is discoverable via the API. If its a completely new resource, the extension describing that resource will suffice. If its an existing resource that was released in a previous cycle having the standard attributes added for the first time, then a dummy extension needs to be added indicating that the resource now has standard attributes. This ensures that an API caller can always discover if an attribute will be available.

For example, if Flavors were migrated to include standard attributes, we need a new flavor-standardattr extension. Then as an API caller, I will know that flavors will have timestamps by checking for flavor-standardattr and timestamps.

Current API resources extended by standard attr extensions:

- subnets: neutron.db.models_v2.Subnet
- trunks: neutron.services.trunk.models.Trunk
- routers: neutron.db.l3_db.Router
- segments: neutron.db.segments_db.NetworkSegment
- security_group_rules: neutron.db.models.securitygroup.SecurityGroupRule
- networks: neutron.db.models_v2.Network
- policies: neutron.db.qos.models.QosPolicy
- subnetpools: neutron.db.models_v2.SubnetPool
- ports: neutron.db.models_v2.Port
- security_groups: neutron.db.models.securitygroup.SecurityGroup
- floatingips: neutron.db.l3_db.FloatingIP
- network_segment_ranges: neutron.db.models.network_segment_range.NetworkSegmentRange

Neutron WSGI/HTTP API layer

This section will cover the internals of Neutrons HTTP API, and the classes in Neutron that can be used to create Extensions to the Neutron API.

Python web applications interface with webservers through the Python Web Server Gateway Interface (WSGI) - defined in PEP 333

Startup

Neutrons WSGI server is started from the server module and the entry point *serve_wsgi* is called to build an instance of the NeutronApiService, which is then returned to the server module, which spawns a Eventlet GreenPool that will run the WSGI application and respond to requests from clients.

WSGI Application

During the building of the NeutronApiService, the *_run_wsgi* function creates a WSGI application using the *load_paste_app* function inside config.py - which parses api-paste.ini - in order to create a WSGI app using Pastes deploy.

The api-paste.ini file defines the WSGI applications and routes - using the Paste INI file format.

The INI file directs paste to instantiate the APIRouter class of Neutron, which contains several methods that map Neutron resources (such as Ports, Networks, Subnets) to URLs, and the controller for each resource.

Further reading

Yong Sheng Gong: Deep Dive into Neutron

Calling the ML2 Plugin

When writing code for an extension, service plugin, or any other part of Neutron you must not call core plugin methods that mutate state while you have a transaction open on the session that you pass into the core plugin method.

The create and update methods for ports, networks, and subnets in ML2 all have a precommit phase and postcommit phase. During the postcommit phase, the data is expected to be fully persisted to the database and ML2 drivers will use this time to relay information to a backend outside of Neutron. Calling the ML2 plugin within a transaction would violate this semantic because the data would not be persisted to the DB; and, were a failure to occur that caused the whole transaction to be rolled back, the backend would become inconsistent with the state in Neutrons DB.

To prevent this, these methods are protected with a decorator that will raise a RuntimeError if they are called with context that has a session in an active transaction. The decorator can be found at neutron.common.utils.transaction_guard and may be used in other places in Neutron to protect functions that are expected to be called outside of a transaction.

Profiling Neutron Code

As more functionality is added to Neutron over time, efforts to improve performance become more difficult, given the rising complexity of the code. Identifying performance bottlenecks is frequently not straightforward, because they arise as a result of complex interactions of different code components.

To help community developers to improve Neutron performance, a Python decorator has been implemented. Decorating a method or a function with it will result in profiling data being added to the corresponding Neutron component log file. These data are generated using cProfile which is part of the Python standard library.

Once a method or function has been decorated, every one of its executions will add to the corresponding log file data grouped in 3 sections:

1. The top calls (sorted by CPU cumulative time) made by the decorated method or function. The number of calls included in this section can be controlled by a configuration option, as explained in *Setting up Neutron for code profiling*. Following is a summary example of this section:

```
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: DEBUG neutron.profiling.profiled_decorator [None req-
↪dc2d428f-4531-4f07-a12d-56843b5f9374 c_rally_8af8f2b4_YbhFJ6Ge c_
↪rally_8af8f2b4_fqvy1XJp] os-profiler parent trace-id c5b30c7f-100b-
↪4e1c-8f07-b2c38f41ad65 trace-id 6324fa85-ea5f-4ae2-9d89-
↪2aabff0dddfc   16928 millisecs elapsed for neutron.plugins.ml2.
↪plugin.create_port((<neutron.plugins.ml2.plugin.Ml2Plugin object at␣
↪0x7f0b4e6ca978>, <neutron_lib.context.Context object at␣
↪0x7f0b4bcee240>, {'port': {'tenant_id':
↪'421ab52e126e45af81a3eb1962613e18', 'network_id': 'dc59577a-9589-
↪4617-82b5-6ee31dbdb15d', 'fixed_ips': [{'ip_address': '1.1.5.177',
↪'subnet_id': 'e15ec947-9edd-4793-bf0f-c463c7ff2f62'}], 'admin_state_
↪up': True, 'device_id': 'f33db890-7958-440e-b07b-432e40bb4049',
↪'device_owner': 'network:router_interface', 'name': '', 'project_id
↪': '421ab52e126e45af81a3eb1962613e18', 'mac_address': <neutron_lib.
↪constants.Sentinel object at 0x7f0b4fc69860>, 'allowed_address_pairs
↪': <neutron_lib.constants.Sentinel object at 0x7f0b4fc69860>,
↪'extra_dhcp_opts': None, 'binding:vnic_type': 'normal',
↪'binding:host_id': <neutron_lib.constants.Sentinel object at␣
↪0x7f0b4fc69860>, 'binding:profile': <neutron_lib.constants.Sentinel␣
↪object at 0x7f0b4fc69860>, 'port_security_enabled': <neutron_lib.
↪constants.Sentinel object at 0x7f0b4fc69860>, 'description': '',
↪'security_groups': <neutron_lib.constants.Sentinel object at␣
↪0x7f0b4fc69860>}}), {}):
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          247612 function calls (238220 primitive␣
↪calls) in 16.943 seconds
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    Ordered by: cumulative time
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    List reduced from 1861 to 100 due to restriction
↪<100>
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    ncalls  tottime  percall  cumtime  percall␣
↪filename:lineno(function)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:       4/2    0.000    0.000   16.932    8.466 /usr/
↪local/lib/python3.6/dist-packages/neutron_lib/db/api.py:132(wrapped)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:         1    0.000    0.000   16.928   16.928 /opt/
↪stack/neutron/neutron/common/utils.py:678(inner)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      20/9    0.000    0.000   16.884    1.876 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/orm/strategies.py:1317(
↪<genexpr>)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:     37/17    0.000    0.000   16.867    0.992 /opt/
↪stack/osprofiler/osprofiler/sqlalchemy.py:84(handler)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:     37/17    0.000    0.000   16.860    0.992 /opt/
↪stack/osprofiler/osprofiler/profiler.py:86(stop)
```

(continues on next page)

(continued from previous page)

```
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:        8/3    0.005    0.001   16.844    5.615 /usr/
↪local/lib/python3.6/dist-packages/neutron_lib/db/api.py:224(wrapped)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          1    0.000    0.000   16.836   16.836 /opt/
↪stack/neutron/neutron/plugins/ml2/plugin.py:1395(_create_port_db)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          1    0.000    0.000   16.836   16.836 /opt/
↪stack/neutron/neutron/db/db_base_plugin_v2.py:1413(create_port_db)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          1    0.000    0.000   16.836   16.836 /opt/
↪stack/neutron/neutron/db/db_base_plugin_v2.py:1586(_enforce_device_
↪owner_not_router_intf_or_device_id)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          1    0.000    0.000   16.836   16.836 /opt/
↪stack/neutron/neutron/db/l3_db.py:522(get_router)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:          1    0.000    0.000   16.836   16.836 /opt/
↪stack/neutron/neutron/db/l3_db.py:186(_get_router)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      34/22    0.000    0.000   16.745    0.761 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/orm/loading.
↪py:35(instances)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:       39/8    0.000    0.000   16.727    2.091 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/sql/elements.py:285(_
↪execute_on_connection)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:       39/8    0.001    0.000   16.727    2.091 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/engine/base.py:1056(_
↪execute_clauseelement)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      17/13    0.000    0.000   16.704    1.285 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/orm/strategies.
↪py:1310(get)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      20/14    0.001    0.000   16.704    1.193 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/orm/strategies.
↪py:1315(_load)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      19/14    0.000    0.000   16.703    1.193 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/orm/loading.py:88(
↪<listcomp>)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      76/23    0.001    0.000   16.699    0.726 /opt/
↪stack/osprofiler/osprofiler/profiler.py:426(_notify)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:       39/8    0.001    0.000   16.696    2.087 /usr/
↪local/lib/python3.6/dist-packages/sqlalchemy/engine/base.py:1163(_
↪execute_context)
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:      75/23    0.000    0.000   16.686    0.725 /opt/
↪stack/osprofiler/osprofiler/notifier.py:28(notify)
```

2. Callers section: all functions or methods that called each function or method in the resulting profiling data. This is restricted by the configured number of top calls to log, as explained in

Setting up Neutron for code profiling. Following is a summary example of this section:

```
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    Ordered by: cumulative time
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    List reduced from 1861 to 100 due to restriction
↪<100>
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: Function                                              ␣
↪                                                          was called␣
↪by...
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                              ncalls␣
↪ tottime  cumtime
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /usr/local/lib/python3.6/dist-packages/neutron_lib/
↪db/api.py:132(wrapped)                                     <-     2/
↪0    0.000    0.000  /usr/local/lib/python3.6/dist-packages/neutron_
↪lib/db/api.py:224(wrapped)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /opt/stack/neutron/neutron/common/utils.
↪py:678(inner)                                                        ␣
↪  <-
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /usr/local/lib/python3.6/dist-packages/sqlalchemy/
↪orm/strategies.py:1317(<genexpr>)                             <-     ␣
↪ 3    0.000    0.000  /opt/stack/osprofiler/osprofiler/profiler.
↪py:426(_notify)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   1␣
↪   0.000   16.883  /usr/local/lib/python3.6/dist-packages/neutron_
↪lib/db/api.py:132(wrapped)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   2␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/engine/base.py:69(__init__)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/engine/base.py:1056(_execute_clauseelement)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   1␣
↪   0.000   16.704  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/query.py:3281(one)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   0␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/query.py:3337(__iter__)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                       ␣
↪                                                                   1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/query.py:3362(_execute_and_instances)
```

(continues on next page)

(continued from previous page)

```
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                          1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/session.py:1127(_connection_for_bind)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                          1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:1310(get)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                          1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:1315(_load)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                          1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:2033(load_scalar_from_joined_new_row)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                        1/0␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/pool/base.py:840(_checkin)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                        1/0␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/webob/
↪request.py:1294(send)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /opt/stack/osprofiler/osprofiler/sqlalchemy.
↪py:84(handler)                                                           <-
↪    16/0    0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/event/attr.py:316(__call__)
Oct 20 01:52:40.767003 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /opt/stack/osprofiler/osprofiler/profiler.
↪py:86(stop)
↪<-     16/0    0.000    0.000  /opt/stack/osprofiler/osprofiler/
↪sqlalchemy.py:84(handler)
```

3. Callees section: a list of all functions or methods that were called by the indicated function or method. Again, this is restricted by the configured number of top calls to log. Following is a summary example of this section:

```
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    Ordered by: cumulative time
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:    List reduced from 1861 to 100 due to restriction
↪<100>
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: Function                                                     ␣
↪                                                               called...
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                              ␣
↪                                                                   ncalls␣
↪ tottime  cumtime
```

(continues on next page)

(continued from previous page)

```
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /usr/local/lib/python3.6/dist-packages/neutron_lib/
↪db/api.py:132(wrapped)                                         ->     1/
↪0    0.000    0.000  /usr/local/lib/python3.6/dist-packages/oslo_db/
↪api.py:135(wrapper)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     1␣
↪   0.000   16.883  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:1317(<genexpr>)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /opt/stack/neutron/neutron/common/utils.
↪py:678(inner)                                                         ␣
↪  ->        1    0.000    0.000  /usr/local/lib/python3.6/dist-
↪packages/neutron_lib/context.py:145(session)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     1␣
↪   0.000   16.928  /usr/local/lib/python3.6/dist-packages/neutron_
↪lib/db/api.py:224(wrapped)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/session.py:2986(is_active)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: /usr/local/lib/python3.6/dist-packages/sqlalchemy/
↪orm/strategies.py:1317(<genexpr>)                              ->     ␣
↪ 1    0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/engine/default.py:579(do_execute)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     2␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/engine/default.py:1078(post_exec)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     2␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/engine/default.py:1122(get_result_proxy)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     0␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/event/attr.py:316(__call__)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/event/base.py:266(__getattr__)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                  15/3␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/loading.py:35(instances)
Oct 20 01:52:40.788842 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:
↪                                                                     1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:1317(<listcomp>)
```

(continues on next page)

(continued from previous page)

```
Oct 20 01:52:40.791161 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     1␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/orm/strategies.py:1318(<lambda>)
Oct 20 01:52:40.791161 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]:                                                        ␣
↪                                                                     3␣
↪   0.000    0.000  /usr/local/lib/python3.6/dist-packages/
↪sqlalchemy/util/langhelpers.py:852(__get__)
```

Setting up Neutron for code profiling

To start profiling Neutron code, the following steps have to be taken:

1. Add he following line to the `[default]` section of `/etc/neutron/neutron.conf` (code profiling is disabled by default):

```
enable_code_profiling = True
```

2. Add the following import line to each module to be profiled:

```
from neutron.profiling import profiled_decorator
```

3. Decorate each mehtod or function to be profiled as follows:

```
@profiled_decorator.profile
def create_subnet(self, context, subnet):
```

4. For each decorated method or function execution, only the top 50 calls by cumulative CPU time are logged. This can be changed adding the following line to the `[default]` section of `/etc/neutron/neutron.conf`:

```
code_profiling_calls_to_log = 100
```

Profiling code with the Neutron Rally job

Code profiling is enabled for the `neutron-rally-task` job in Neutrons check queue in Zuul. Taking advantage of the fact that `os-profiler` is enabled for this job, the data logged by the `profiled_decorator.profile` decorator includes the `os-profiler` `parent trace-id` and `trace-id` as can be seen here:

```
Oct 20 01:52:40.759379 ubuntu-bionic-vexxhost-sjc1-0012393267 neutron-
↪server[19578]: DEBUG neutron.profiling.profiled_decorator [None req-
↪dc2d428f-4531-4f07-a12d-56843b5f9374 c_rally_8af8f2b4_YbhFJ6Ge c_rally_
↪8af8f2b4_fqvy1XJp] os-profiler parent trace-id c5b30c7f-100b-4e1c-8f07-
↪b2c38f41ad65 trace-id 6324fa85-ea5f-4ae2-9d89-2aabff0dddfc   16928␣
↪millisecs elapsed for neutron.plugins.ml2.plugin.create_port((<neutron.
↪plugins.ml2.plugin.Ml2Plugin object at 0x7f0b4e6ca978>, <neutron_lib.
↪context.Context object at 0x7f0b4bcee240>, {'port': {'tenant_id':
↪'421ab52e126e45af81a3eb1962613e18', 'network_id': 'dc59577a-9589-4617-
↪82b5-6ee31dbdb15d', 'fixed_ips': [{'ip_address': '1.1.5.177', 'subnet_id
↪': 'e15ec947-9edd-4793-bf0f-c463c7ff2f62'}], 'admin_state_up': True,
↪'device_id': 'f33db890-7958-440e-b07b-432e40bb4049', 'device_owner':
↪network:router_interface', 'name': '', 'projec
↪'421ab52e126e45af81a3eb1962613e18', 'mac_address': <neutron_lib.
↪constants.Sentinel object at 0x7f0b4fc69860>, 'allowed_address_pairs':
↪<neutron_lib.constants.Sentinel object at 0x7f0b4fc69860>, 'extra_dhcp_
```

(continues on next page)

(continued from previous page)

Community developers wanting to use this to correlate data from `os-profiler` and the `profiled_decorator.profile` decorator can submit a `DNM` (Do Not Merge) patch, decorating the functions and methods they want to profile and optionally:

1. Configure the number of calls to be logged in the `neutron-rally-task` job definition, as described in *Setting up Neutron for code profiling*.
2. Increase the `timeout` parameter value of the `neutron-rally-task` job in the .zuul yaml file. The value used for the Neutron gate might be too short when logging large quantities of profiling data.

The `profiled_decorator.profile` and `os-profiler` data will be found in the `neutron-rally-task` log files and `HTML report` respectively.

Neutron Database Layer

This section contains some common information that will be useful for developers that need to do some db changes.

Difference between default and server_default parameters for columns

For columns it is possible to set default or server_default. What is the difference between them and why should they be used?

The explanation is quite simple:

- default - the default value that SQLAlchemy will specify in queries for creating instances of a given model;
- server_default - the default value for a column that SQLAlchemy will specify in DDL.

Summarizing, default is useless in migrations and only server_default should be used. For synchronizing migrations with models server_default parameter should also be added in model. If default value in database is not needed, server_default should not be used. The declarative approach can be bypassed (i.e. default may be omitted in the model) if default is enforced through business logic.

Database migrations

For details on the neutron-db-manage wrapper and alembic migrations, see Alembic Migrations.

Tests to verify that database migrations and models are in sync

class neutron.tests.functional.db.test_migrations.**_TestModelsMigrations**

Test for checking of equality models state and migrations.

For the opportunistic testing you need to set up a db named openstack_citest with user openstack_citest and password openstack_citest on localhost. The test will then use that db and user/password combo to run the tests.

For PostgreSQL on Ubuntu this can be done with the following commands:

```
sudo -u postgres psql
postgres=# create user openstack_citest with createdb login password
          'openstack_citest';
postgres=# create database openstack_citest with owner
           openstack_citest;
```

For MySQL on Ubuntu this can be done with the following commands:

```
mysql -u root
>create database openstack_citest;
>grant all privileges on openstack_citest.* to
 openstack_citest@localhost identified by 'openstack_citest';
```

Output is a list that contains information about differences between db and models. Output example:

```
[('add_table',
  Table('bat', MetaData(bind=None),
        Column('info', String(), table=<bat>), schema=None)),
 ('remove_table',
  Table(u'bar', MetaData(bind=None),
        Column(u'data', VARCHAR(), table=<bar>), schema=None)),
 ('add_column',
  None,
  'foo',
  Column('data', Integer(), table=<foo>)),
 ('remove_column',
  None,
  'foo',
  Column(u'old_data', VARCHAR(), table=None)),
 [('modify_nullable',
   None,
   'foo',
   u'x',
   {'existing_server_default': None,
   'existing_type': INTEGER()},
   True,
   False)]]
```

- `remove_*` means that there is extra table/column/constraint in db;
- `add_*` means that it is missing in db;
- `modify_*` means that on column in db is set wrong type/nullable/server_default. Element contains information:

- what should be modified,
- schema,
- table,
- column,
- existing correct column parameters,
- right value,
- wrong value.

This class also contains tests for branches, like that correct operations are used in contract and expand branches.

db_sync(*engine*)
Run migration scripts with the given engine instance.

This method must be implemented in subclasses and run migration scripts for a DB the given engine is connected to.

filter_metadata_diff(*diff*)
Filter changes before assert in test_models_sync().

Allow subclasses to whitelist/blacklist changes. By default, no filtering is performed, changes are returned as is.

Parameters **diff** a list of differences (see *compare_metadata()* docs for details on format)

Returns a list of differences

get_engine()
Return the engine instance to be used when running tests.

This method must be implemented in subclasses and return an engine instance to be used when running tests.

get_metadata()
Return the metadata instance to be used for schema comparison.

This method must be implemented in subclasses and return the metadata instance attached to the BASE model.

include_object(*object_*, *name*, *type_*, *reflected*, *compare_to*)
Return True for objects that should be compared.

Parameters

- **object** a SchemaItem object such as a Table or Column object
- **name** the name of the object
- **type** a string describing the type of object (e.g. table)
- **reflected** True if the given object was produced based on table reflection, False if its from a local MetaData object
- **compare_to** the object being compared against, if available, else None

The Standard Attribute Table

There are many attributes that we would like to store in the database which are common across many Neutron objects (e.g. tags, timestamps, rbac entries). We have previously been handling this by duplicating the schema to every table via model mixins. This means that a DB migration is required for each object that wants to adopt one of these common attributes. This becomes even more cumbersome when the relationship between the attribute and the object is many-to-one because each object then needs its own table for the attributes (assuming referential integrity is a concern).

To address this issue, the standardattribute table is available. Any model can add support for this table by inheriting the HasStandardAttributes mixin in neutron.db.standard_attr. This mixin will add a standard_attr_id BigInteger column to the model with a foreign key relationship to the standardattribute table. The model will then be able to access any columns of the standardattribute table and any tables related to it.

A model that inherits HasStandardAttributes must implement the property api_collections, which is a list of API resources that the new object may appear under. In most cases, this will only be one (e.g. ports for the Port model). This is used by all of the service plugins that add standard attribute fields to determine which API responses need to be populated.

A model that supports tag mechanism must implement the property collection_resource_map which is a dict of collection_name and resource_name for API resources. And also the model must implement tag_support with a value True.

The introduction of a new standard attribute only requires one column addition to the standardattribute table for one-to-one relationships or a new table for one-to-many or one-to-zero relationships. Then all of the models using the HasStandardAttribute mixin will automatically gain access to the new attribute.

Any attributes that will apply to every neutron resource (e.g. timestamps) can be added directly to the standardattribute table. For things that will frequently be NULL for most entries (e.g. a column to store an error reason), a new table should be added and joined to in a query to prevent a bunch of NULL entries in the database.

Relocation of Database Models

This document is intended to track and notify developers that db models in neutron will be centralized and moved to a new tree under neutron/db/models. This was discussed in [1]. The reason for relocating db models is to solve the cyclic import issue while implementing oslo versioned objects for resources in neutron.

The reason behind this relocation is Mixin class and db models for some resources in neutron are in same module. In Mixin classes, there are methods which provide functionality of fetching, adding, updating and deleting data via queries. These queries will be replaced with use of versioned objects and definition of versioned object will be using db models. So object files will be importing models and Mixin need to import those objects which will end up in cyclic import.

Structure of Model Definitions

We have decided to move all models definitions to neutron/db/models/ with no further nesting after that point. The deprecation method to move models has already been added to avoid breakage of third party plugins using those models. All relocated models need to use deprecate method that will generate a warning and return new class for use of old class. Some examples of relocated models [2] and [3]. In future if you define new models please make sure they are separated from mixins and are under tree neutron/db/models/ .

References

[1]. https://www.mail-archive.com/openstack-dev@lists.openstack.org/msg88910.html [2]. https://review.opendev.org/#/c/348562/ [3]. https://review.opendev.org/#/c/348757/

Keep DNS Nameserver Order Consistency In Neutron

In Neutron subnets, DNS nameservers are given priority when created or updated. This means if you create a subnet with multiple DNS servers, the order will be retained and guests will receive the DNS servers in the order you created them in when the subnet was created. The same thing applies for update operations on subnets to add, remove, or update DNS servers.

Get Subnet Details Info

```
changzhi@stack:~/devstack$ neutron subnet-list
+--------------------------------------+------+-------------+--------------
↪-------------------------------+
| id                                   | name | cidr        | allocation_
↪pools                          |
+--------------------------------------+------+-------------+--------------
↪-------------------------------+
| 1a2d261b-b233-3ab9-902e-88576a82afa6 |      | 10.0.0.0/24 | {"start":
↪"10.0.0.2", "end": "10.0.0.254"} |
+--------------------------------------+------+-------------+--------------
↪-------------------------------+

changzhi@stack:~/devstack$ neutron subnet-show 1a2d261b-b233-3ab9-902e-
↪88576a82afa6
+------------------+--------------------------------------------+
| Field            | Value                                      |
+------------------+--------------------------------------------+
| allocation_pools | {"start": "10.0.0.2", "end": "10.0.0.254"} |
| cidr             | 10.0.0.0/24                                |
| dns_nameservers  | 1.1.1.1                                    |
|                  | 2.2.2.2                                    |
|                  | 3.3.3.3                                    |
| enable_dhcp      | True                                       |
| gateway_ip       | 10.0.0.1                                   |
| host_routes      |                                            |
| id               | 1a2d26fb-b733-4ab3-992e-88554a87afa6       |
| ip_version       | 4                                          |
| name             |                                            |
```

(continues on next page)

(continued from previous page)

```
| network_id        | a404518c-800d-2353-9193-57dbb42ac5ee         |
| tenant_id         | 3868290ab10f417390acbb754160dbb2             |
+-------------------+----------------------------------------------+
```

Update Subnet DNS Nameservers

```
neutron subnet-update 1a2d261b-b233-3ab9-902e-88576a82afa6 \
--dns_nameservers list=true 3.3.3.3 2.2.2.2 1.1.1.1

changzhi@stack:~/devstack$ neutron subnet-show 1a2d261b-b233-3ab9-902e-
↪88576a82afa6
+-------------------+----------------------------------------------+
| Field             | Value                                        |
+-------------------+----------------------------------------------+
| allocation_pools  | {"start": "10.0.0.2", "end": "10.0.0.254"}  |
| cidr              | 10.0.0.0/24                                  |
| dns_nameservers   | 3.3.3.3                                      |
|                   | 2.2.2.2                                      |
|                   | 1.1.1.1                                      |
| enable_dhcp       | True                                         |
| gateway_ip        | 10.0.0.1                                     |
| host_routes       |                                              |
| id                | 1a2d26fb-b733-4ab3-992e-88554a87afa6         |
| ip_version        | 4                                            |
| name              |                                              |
| network_id        | a404518c-800d-2353-9193-57dbb42ac5ee         |
| tenant_id         | 3868290ab10f417390acbb754160dbb2             |
+-------------------+----------------------------------------------+
```

As shown in above output, the order of the DNS nameservers has been updated. New virtual machines deployed to this subnet will receive the DNS nameservers in this new priority order. Existing virtual machines that have already been deployed will not be immediately affected by changing the DNS nameserver order on the neutron subnet. Virtual machines that are configured to get their IP address via DHCP will detect the DNS nameserver order change when their DHCP lease expires or when the virtual machine is restarted. Existing virtual machines configured with a static IP address will never detect the updated DNS nameserver order.

Integration with external DNS services

Since the Mitaka release, neutron has an interface defined to interact with an external DNS service. This interface is based on an abstract driver that can be used as the base class to implement concrete drivers to interact with various DNS services. The reference implementation of such a driver integrates neutron with OpenStack Designate.

This integration allows users to publish *dns_name* and *dns_domain* attributes associated with floating IP addresses, ports, and networks in an external DNS service.

Changes to the neutron API

To support integration with an external DNS service, the *dns_name* and *dns_domain* attributes were added to floating ips, ports and networks. The *dns_name* specifies the name to be associated with a corresponding IP address, both of which will be published to an existing domain with the name *dns_domain* in the external DNS service.

Specifically, floating ips, ports and networks are extended as follows:

- Floating ips have a *dns_name* and a *dns_domain* attribute.
- Ports have a *dns_name* attribute.
- Networks have a *dns_domain* attributes.

Neutron Stadium i18n

- Refer to oslo_i18n documentation for the general mechanisms that should be used: https://docs.openstack.org/oslo.i18n/latest/user/usage.html
- Each stadium project should NOT consume _i18n module from neutron-lib or neutron.
- It is recommended that you create a {package_name}/_i18n.py file in your repo, and use that. Your localization strings will also live in your repo.

L2 agent extensions

L2 agent extensions are part of a generalized L2/L3 extension framework. See *agent extensions*.

Open vSwitch agent API

- neutron.plugins.ml2.drivers.openvswitch.agent.ovs_agent_extension_api

Open vSwitch agent API object includes two methods that return wrapped and hardened bridge objects with cookie values allocated for calling extensions:

```
#. request_int_br
#. request_tun_br
```

Bridge objects returned by those methods already have new default cookie values allocated for extension flows. All flow management methods (add_flow, mod_flow,) enforce those allocated cookies.

Linuxbridge agent API

- neutron.plugins.ml2.drivers.linuxbridge.agent.linuxbridge_agent_extension_api

The Linux bridge agent extension API object includes a method that returns an instance of the Iptables-Manager class, which is used by the L2 agent to manage security group rules:

```
#. get_iptables_manager
```

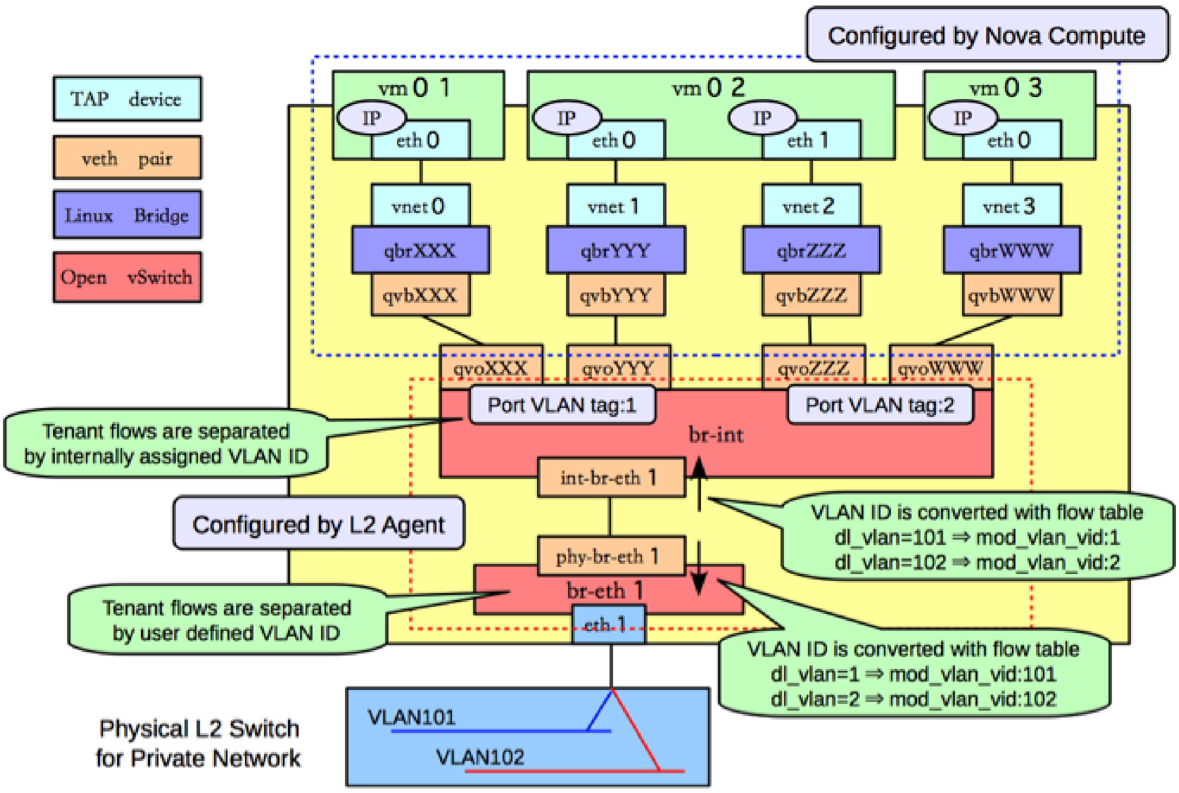
L2 Agent Networking

Open vSwitch L2 Agent

This Agent uses the Open vSwitch virtual switch to create L2 connectivity for instances, along with bridges created in conjunction with OpenStack Nova for filtering.

ovs-neutron-agent can be configured to use different networking technologies to create project isolation. These technologies are implemented as ML2 type drivers which are used in conjunction with the Open vSwitch mechanism driver.

VLAN Tags



GRE Tunnels

GRE Tunneling is documented in depth in the Networking in too much detail by RedHat.

VXLAN Tunnels

VXLAN is an overlay technology which encapsulates MAC frames at layer 2 into a UDP header. More information can be found in The VXLAN wiki page.

Geneve Tunnels

Geneve uses UDP as its transport protocol and is dynamic in size using extensible option headers. It is important to note that currently it is only supported in newer kernels. (kernel >= 3.18, OVS version >=2.4) More information can be found in the Geneve RFC document.

Bridge Management

In order to make the agent capable of handling more than one tunneling technology, to decouple the requirements of segmentation technology from project isolation, and to preserve backward compatibility for OVS agents working without tunneling, the agent relies on a tunneling bridge, or br-tun, and the well known integration bridge, or br-int.

All VM VIFs are plugged into the integration bridge. VM VIFs on a given virtual network share a common local VLAN (i.e. not propagated externally). The VLAN id of this local VLAN is mapped to the physical networking details realizing that virtual network.

For virtual networks realized as VXLAN/GRE tunnels, a Logical Switch (LS) identifier is used to differentiate project traffic on inter-HV tunnels. A mesh of tunnels is created to other Hypervisors in the cloud. These tunnels originate and terminate on the tunneling bridge of each hypervisor, leaving br-int unaffected. Port patching is done to connect local VLANs on the integration bridge to inter-hypervisor tunnels on the tunnel bridge.

For each virtual network realized as a VLAN or flat network, a veth or a pair of patch ports is used to connect the local VLAN on the integration bridge with the physical network bridge, with flow rules adding, modifying, or stripping VLAN tags as necessary, thus preserving backward compatibility with the way the OVS agent used to work prior to the tunneling capability (for more details, please look at https://review.opendev.org/#/c/4367).

Bear in mind, that this design decision may be overhauled in the future to support existing VLAN-tagged traffic (coming from NFV VMs for instance) and/or to deal with potential QinQ support natively available in the Open vSwitch.

Tackling the Network Trunking use case

Rationale

At the time the first design for the OVS agent came up, trunking in OpenStack was merely a pipe dream. Since then, lots has happened in the OpenStack platform, and many deployments have gone into production since early 2012.

In order to address the vlan-aware-vms use case on top of Open vSwitch, the following aspects must be taken into account:

- Design complexity: starting afresh is always an option, but a complete rearchitecture is only desirable under some circumstances. After all, customers want solutionsyesterday. It is noteworthy that the OVS agent design is already relatively complex, as it accommodates a number of deployment options, especially in relation to security rules and/or acceleration.
- Upgrade complexity: being able to retrofit the existing design means that an existing deployment does not need to go through a forklift upgrade in order to expose new functionality; alternatively, the desire of avoiding a migration requires a more complex solution that is able to support multiple modes of operations;
- Design reusability: ideally, a proposed design can easily apply to the various technology backends that the Neutron L2 agent supports: Open vSwitch and Linux Bridge.
- Performance penalty: no solution is appealing enough if it is unable to satisfy the stringent requirement of high packet throughput, at least in the long term.
- Feature compatibility: VLAN transparency is for better or for worse intertwined with vlan awareness. The former is about making the platform not interfere with the tag associated to the packets sent by the VM, and let the underlay figure out where the packet needs to be sent out; the latter is about making the platform use the vlan tag associated to packet to determine where the packet needs to go. Ideally, a design choice to satisfy the awareness use case will not have a negative impact for solving the transparency use case. Having said that, the two features are still meant to be mutually exclusive in their application, and plugging subports into networks whose vlan-transparency flag is set to True might have unexpected results. In fact, it would be impossible from the platforms point of view discerning which tagged packets are meant to be treated transparently and which ones are meant to be used for demultiplexing (in order to reach the right destination). The outcome might only be predictable if two layers of vlan tags are stacked up together, making guest support even more crucial for the combined use case.

It is clear by now that an acceptable solution must be assessed with these issues in mind. The potential solutions worth enumerating are:

- VLAN interfaces: in laymans terms, these interfaces allow to demux the traffic before it hits the integration bridge where the traffic will get isolated and sent off to the right destination. This solution is proven to work for both iptables-based and native ovs security rules (credit to Rawlin Peters). This solution has the following design implications:
  - Design complexity: this requires relative small changes to the existing OVS design, and it can work with both iptables and native ovs security rules.
  - Upgrade complexity: in order to employ this solution no major upgrade is necessary and thus no potential dataplane disruption is involved.
  - Design reusability: VLAN interfaces can easily be employed for both Open vSwitch and Linux Bridge.

  - Performance penalty: using VLAN interfaces means that the kernel must be involved. For Open vSwitch, being able to use a fast path like DPDK would be an unresolved issue (Kernel NIC interfaces are not on the roadmap for distros and OVS, and most likely will never be). Even in the absence of an extra bridge, i.e. when using native ovs firewall, and with the advent of userspace connection tracking that would allow the stateful firewall driver to work with DPDK, the performance gap between a pure userspace DPDK capable solution and a kernel based solution will be substantial, at least under certain traffic conditions.
  - Feature compatibility: in order to keep the design simple once VLAN interfaces are adopted, and yet enable VLAN transparency, Open vSwitch needs to support QinQ, which is currently lacking as of 2.5 and with no ongoing plan for integration.
- Going full openflow: in laymans terms, this means programming the dataplane using OpenFlow in order to provide tenant isolation, and packet processing. This solution has the following design implications:
  - Design complexity: this requires a big rearchitecture of the current Neutron L2 agent solution.
  - Upgrade complexity: existing deployments will be unable to work correctly unless one of the actions take place: a) the agent can handle both the old and new way of wiring the data path; b) a dataplane migration is forced during a release upgrade and thus it may cause (potentially unrecoverable) dataplane disruption.
  - Design reusability: a solution for Linux Bridge will still be required to avoid widening the gap between Open vSwitch (e.g. OVS has DVR but LB does not).
  - Performance penalty: using Open Flow will allow to leverage the user space and fast processing given by DPDK, but at a considerable engineering cost nonetheless. Security rules will have to be provided by a learn based firewall to fully exploit the capabilities of DPDK, at least until user space connection tracking becomes available in OVS.
  - Feature compatibility: with the adoption of Open Flow, tenant isolation will no longer be provided by means of local vlan provisioning, thus making the requirement of QinQ support no longer strictly necessary for Open vSwitch.
- Per trunk port OVS bridge: in laymans terms, this is similar to the first option, in that an extra layer of mux/demux is introduced between the VM and the integration bridge (br-int) but instead of using vlan interfaces, a combination of a new per port OVS bridge and patch ports to wire this new bridge with br-int will be used. This solution has the following design implications:
  - Design complexity: the complexity of this solution can be considered in between the above mentioned options in that some work is already available since Mitaka and the data path wiring logic can be partially reused.
  - Upgrade complexity: if two separate code paths are assumed to be maintained in the OVS agent to handle regular ports and ports participating a trunk with no ability to convert from one to the other (and vice versa), no migration is required. This is done at a cost of some loss of flexibility and maintenance complexity.
  - Design reusability: a solution to support vlan trunking for the Linux Bridge mech driver will still be required to avoid widening the gap with Open vSwitch (e.g. OVS has DVR but LB does not).
  - Performance penalty: from a performance standpoint, the adoption of a trunk bridge relieves the agent from employing kernel interfaces, thus unlocking the full potential of fast packet processing. That said, this is only doable in combination with a native ovs firewall. At the

time of writing the only DPDK enabled firewall driver is the learn based one available in the networking-ovs-dpdk repo;

– Feature compatibility: the existing local provisioning logic will not be affected by the introduction of a trunk bridge, therefore use cases where VMs are connected to a vlan transparent network via a regular port will still require QinQ support from OVS.

To summarize:

- VLAN interfaces (A) are compelling because will lead to a relatively contained engineering cost at the expense of performance. The Open vSwitch community will need to be involved in order to deliver vlan transparency. Irrespective of whether this strategy is chosen for Open vSwitch or not, this is still the only viable approach for Linux Bridge and thus pursued to address Linux Bridge support for VLAN trunking. To some extent, this option can also be considered a fallback strategy for OVS deployments that are unable to adopt DPDK.
- Open Flow (B) is compelling because it will allow Neutron to unlock the full potential of Open vSwitch, at the expense of development and operations effort. The development is confined within the boundaries of the Neutron community in order to address vlan awareness and transparency (as two distinct use cases, ie. to be adopted separately). Stateful firewall (based on ovs conntrack) limits the adoption for DPDK at the time of writing, but a learn-based firewall can be a suitable alternative. Obviously this solution is not compliant with iptables firewall.
- Trunk Bridges (C) tries to bring the best of option A and B together as far as OVS development and performance are concerned, but it comes at the expense of maintenance complexity and loss of flexibility. A Linux Bridge solution would still be required and, QinQ support will still be needed to address vlan transparency.

All things considered, as far as OVS is concerned, option (C) is the most promising in the medium term. Management of trunks and ports within trunks will have to be managed differently and, to start with, it is sensible to restrict the ability to update ports (i.e. convert) once they are bound to a particular bridge (integration vs trunk). Security rules via iptables rules is obviously not supported, and never will be.

Option (A) for OVS could be pursued in conjunction with Linux Bridge support, if the effort is seen particularly low hanging fruit. However, a working solution based on this option positions the OVS agent as a sub-optminal platform for performance sensitive applications in comparison to other accelerated or SDN-controller based solutions. Since further data plane performance improvement is hindered by the extra use of kernel resources, this option is not at all appealing in the long term.

Embracing option (B) in the long run may be complicated by the adoption of option (C). The development and maintenance complexity involved in Option (C) and (B) respectively poses the existential question as to whether investing in the agent-based architecture is an effective strategy, especially if the end result would look a lot like other maturing alternatives.

Implementation VLAN Interfaces (Option A)

This implementation doesnt require any modification of the vif-drivers since Nova will plug the vif of the VM the same way as it does for traditional ports.

Trunk port creation

A VM is spawned passing to Nova the port-id of a parent port associated with a trunk. Nova/libvirt will create the tap interface and will plug it into br-int or into the firewall bridge if using iptables firewall. In the external-ids of the port Nova will store the port ID of the parent port. The OVS agent detects that a new vif has been plugged. It gets the details of the new port and wires it. The agent configures it in the same way as a traditional port: packets coming out from the VM will be tagged using the internal VLAN ID associated to the network, packets going to the VM will be stripped of the VLAN ID. After wiring it successfully the OVS agent will send a message notifying Neutron server that the parent port is up. Neutron will send back to Nova an event to signal that the wiring was successful. If the parent port is associated with one or more subports the agent will process them as described in the next paragraph.

Subport creation

If a subport is added to a parent port but no VM was booted using that parent port yet, no L2 agent will process it (because at that point the parent port is not bound to any host). When a subport is created for a parent port and a VM that uses that parent port is already running, the OVS agent will create a VLAN interface on the VM tap using the VLAN ID specified in the subport segmentation id. Theres a small possibility that a race might occur: the firewall bridge might be created and plugged while the vif is not there yet. The OVS agent needs to check if the vif exists before trying to create a subinterface. Lets see how the models differ when using the iptables firewall or the ovs native firewall.

Iptables Firewall

```
+----------------------------+
|             VM             |
|   eth0             eth0.100 |
+-----+-----------------+----+
      |
      |
  +---+---+        +-----+-----+
  | tap1  |--------|  tap1.100 |
  +---+---+        +-----+-----+
      |                  |
      |                  |
  +---+---+          +---+---+
  | qbr1  |          | qbr2  |
  +---+---+          +---+---+
      |                  |
      |                  |
+-----+-----------------+----+
|     port 1           port 2 |
|    (tag 3)          (tag 5) |
|            br-int           |
+----------------------------+
```

Lets assume the subport is on network2 and uses segmentation ID 100. In the case of hybrid plugging the OVS agent will have to create the firewall bridge (qbr2), create tap1.100 and plug it into qbr2. It will connect qbr2 to br-int and set the subport ID in the external-ids of port 2.

Inbound traffic from the VM point of view

The untagged traffic will flow from port 1 to eth0 through qbr1. For the traffic coming out of port 2, the internal VLAN ID of network2 will be stripped. The packet will then go untagged through qbr2 where iptables rules will filter the traffic. The tag 100 will be pushed by tap1.100 and the packet will finally get to eth0.100.

Outbound traffic from the VM point of view

The untagged traffic will flow from eth0 to port1 going through qbr1 where firewall rules will be applied. Traffic tagged with VLAN 100 will leave eth0.100, go through tap1.100 where the VLAN 100 is stripped. It will reach qbr2 where iptables rules will be applied and go to port 2. The internal VLAN of network2 will be pushed by br-int when the packet enters port2 because its a tagged port.

OVS Firewall case

```
+----------------------------+
|             VM             |
|   eth0            eth0.100 |
+-----+-----------------+----+
      |
      |
  +---+---+       +-----+-----+
  | tap1  |-------|  tap1.100 |
  +---+---+       +-----+-----+
      |                 |
      |                 |
      |                 |
+-----+-----------------+----+
|    port 1          port 2  |
|   (tag 3)         (tag 5)  |
|           br-int           |
+----------------------------+
```

When a subport is created the OVS agent will create the VLAN interface tap1.100 and plug it into br-int. Lets assume the subport is on network2.

Inbound traffic from the VM point of view

The traffic will flow untagged from port 1 to eth0. The traffic going out from port 2 will be stripped of the VLAN ID assigned to network2. It will be filtered by the rules installed by the firewall and reach tap1.100. tap1.100 will tag the traffic using VLAN 100. It will then reach the VMs eth0.100.

Outbound traffic from the VM point of view

The untagged traffic will flow and reach port 1 where it will be tagged using the VLAN ID associated to the network. Traffic tagged with VLAN 100 will leave eth0.100 reach tap1.100 where VLAN 100 will be stripped. It will then reach port2. It will be filtered by the rules installed by the firewall on port 2. Then the packets will be tagged using the internal VLAN associated to network2 by br-int since port 2 is a tagged port.

Parent port deletion

Deleting a port that is an active parent in a trunk is forbidden. If the parent port has no trunk associated (its a normal port), it can be deleted. The OVS agent doesnt need to perform any action, the deletion will result in a removal of the port data from the DB.

Trunk deletion

When Nova deletes a VM, it deletes the VMs corresponding Neutron ports only if they were created by Nova when booting the VM. In the vlan-aware-vm case the parent port is passed to Nova, so the port data will remain in the DB after the VM deletion. Nova will delete the VIF of the VM (in the example tap1) as part of the VM termination. The OVS agent will detect that deletion and notify the Neutron server that the parent port is down. The OVS agent will clean up the corresponding subports as explained in the next paragraph.

The deletion of a trunk that is used by a VM is not allowed. The trunk can be deleted (leaving the parent port intact) when the parent port is not used by any VM. After the trunk is deleted, the parent port can also be deleted.

Subport deletion

Removing a subport that is associated with a parent port that was not used to boot any VM is a no op from the OVS agent perspective. When a subport associated with a parent port that was used to boot a VM is deleted, the OVS agent will take care of removing the firewall bridge if using iptables firewall and the port on br-int.

Implementation Trunk Bridge (Option C)

This implementation is based on this etherpad. Credits to Bence Romsics. The option use_veth_interconnection=true wont be supported, it is deprecated since Victoria, see [1]. The IDs used for bridge and port names are truncated.

```
+--------------------------------+
|             VM                 |
|   eth0                eth0.100 |
+-----+------------------+-------+
      |
      |
+-----+--------------------------+
|    tap1                        |
|          tbr-trunk-id          |
|                                |
| tpt-parent-id    spt-subport-id |
|                    (tag 100)    |
+-----+-----------------+--------+
      |                 |
      |                 |
      |                 |
+-----+-----------------+---------+
| tpi-parent-id     spi-subport-id |
```

(continues on next page)

(continued from previous page)

```
|   (tag 3)             (tag 5)        |
|                                      |
|          br-int                      |
+--------------------------------------+
```

tpt-parent-id: trunk bridge side of the patch port that implements a trunk. tpi-parent-id: int bridge side of the patch port that implements a trunk. spt-subport-id: trunk bridge side of the patch port that implements a subport. spi-subport-id: int bridge side of the patch port that implements a subport.

[1] https://bugs.launchpad.net/neutron/+bug/1587296

Trunk creation

A VM is spawned passing to Nova the port-id of a parent port associated with a trunk. Neutron will pass to Nova the bridge where to plug the vif as part of the vif details. The os-vif driver creates the trunk bridge tbr-trunk-id if it does not exist in plug(). It will create the tap interface tap1 and plug it into tbr-trunk-id setting the parent port ID in the external-ids. The OVS agent will be monitoring the creation of ports on the trunk bridges. When it detects that a new port has been created on the trunk bridge, it will do the following:

```
ovs-vsctl add-port tbr-trunk-id tpt-parent-id -- set Interface tpt-parent-
↪id type=patch options:peer=tpi-parent-id
ovs-vsctl add-port br-int tpi-parent-id tag=3 -- set Interface tpi-parent-
↪id type=patch options:peer=tpt-parent-id
```

A patch port is created to connect the trunk bridge to the integration bridge. tpt-parent-id, the trunk bridge side of the patch is not associated to any tag. It will carry untagged traffic. tpi-parent-id, the br-int side the patch port is tagged with VLAN 3. We assume that the trunk is on network1 that on this host is associated with VLAN 3. The OVS agent will set the trunk ID in the external-ids of tpt-parent-id and tpi-parent-id. If the parent port is associated with one or more subports the agent will process them as described in the next paragraph.

Subport creation

If a subport is added to a parent port but no VM was booted using that parent port yet, the agent wont process the subport (because at this point theres no node associated with the parent port). When a subport is added to a parent port that is used by a VM the OVS agent will create a new patch port:

```
ovs-vsctl add-port tbr-trunk-id spt-subport-id tag=100 -- set Interface␣
↪spt-subport-id type=patch options:peer=spi-subport-id
ovs-vsctl add-port br-int spi-subport-id tag=5 -- set Interface spi-
↪subport-id type=patch options:peer=spt-subport-id
```

This patch port connects the trunk bridge to the integration bridge. spt-subport-id, the trunk bridge side of the patch is tagged using VLAN 100. We assume that the segmentation ID of the subport is 100. spi-subport-id, the br-int side of the patch port is tagged with VLAN 5. We assume that the subport is on network2 that on this host uses VLAN 5. The OVS agent will set the subport ID in the external-ids of spt-subport-id and spi-subport-id.

Inbound traffic from the VM point of view

The traffic coming out of tpi-parent-id will be stripped by br-int of VLAN 3. It will reach tpt-parent-id untagged and from there tap1. The traffic coming out of spi-subport-id will be stripped by br-int of VLAN 5. It will reach spt-subport-id where it will be tagged with VLAN 100 and it will then get to tap1 tagged.

Outbound traffic from the VM point of view

The untagged traffic coming from tap1 will reach tpt-parent-id and from there tpi-parent-id where it will be tagged using VLAN 3. The traffic tagged with VLAN 100 from tap1 will reach spt-subport-id. VLAN 100 will be stripped since spt-subport-id is a tagged port and the packet will reach spi-subport-id, where its tagged using VLAN 5.

Parent port deletion

Deleting a port that is an active parent in a trunk is forbidden. If the parent port has no trunk associated, it can be deleted. The OVS agent doesnt need to perform any action.

Trunk deletion

When Nova deletes a VM, it deletes the VMs corresponding Neutron ports only if they were created by Nova when booting the VM. In the vlan-aware-vm case the parent port is passed to Nova, so the port data will remain in the DB after the VM deletion. Nova will delete the port on the trunk bridge where the VM is plugged. The L2 agent will detect that and delete the trunk bridge. It will notify the Neutron server that the parent port is down.

The deletion of a trunk that is used by a VM is not allowed. The trunk can be deleted (leaving the parent port intact) when the parent port is not used by any VM. After the trunk is deleted, the parent port can also be deleted.

Subport deletion

The OVS agent will delete the patch port pair corresponding to the subport deleted.

Agent resync

During resync the agent should check that all the trunk and subports are still valid. It will delete the stale trunk and subports using the procedure specified in the previous paragraphs according to the implementation.

Further Reading

- Darragh OReilly - The Open vSwitch plugin with VLANs

L2 Networking with Linux Bridge

This Agent uses the Linux Bridge to provide L2 connectivity for VM instances running on the compute node to the public network. A graphical illustration of the deployment can be found in Networking Guide.

In most common deployments, there is a compute and a network node. On both the compute and the network node, the Linux Bridge Agent will manage virtual switches, connectivity among them, and interaction via virtual ports with other network components such as namespaces and underlying interfaces. Additionally, on the compute node, the Linux Bridge Agent will manage security groups.

Three use cases and their packet flow are documented as follows:

1. Linux Bridge: Provider networks
2. Linux Bridge: Self-service networks
3. Linux Bridge: High availability using VRRP

L2 Networking with SR-IOV enabled NICs

SR-IOV (Single Root I/O Virtualization) is a specification that allows a PCIe device to appear to be multiple separate physical PCIe devices. SR-IOV works by introducing the idea of physical functions (PFs) and virtual functions (VFs). Physical functions (PFs) are full-featured PCIe functions. Virtual functions (VFs) are lightweight functions that lack configuration resources.

SR-IOV supports VLANs for L2 network isolation, other networking technologies such as VXLAN/GRE may be supported in the future.

SR-IOV NIC agent manages configuration of SR-IOV Virtual Functions that connect VM instances running on the compute node to the public network.

In most common deployments, there are compute and a network nodes. Compute node can support VM connectivity via SR-IOV enabled NIC. SR-IOV NIC Agent manages Virtual Functions admin state. Quality of service is partially implemented with the bandwidth limit and minimum bandwidth rules. In the future it will manage additional settings, such as additional quality of service rules, rate limit settings, spoofcheck and more. Network node will be usually deployed with either Open vSwitch or Linux Bridge to support network node functionality.

Further Reading

Nir Yechiel - SR-IOV Networking Part I: Understanding the Basics

SR-IOV Passthrough For Networking

L3 agent extensions

L3 agent extensions are part of a generalized L2/L3 extension framework. See *agent extensions*.

L3 agent extension API

The L3 agent extension API object includes several methods that expose router information to L3 agent extensions:

```
#. get_routers_in_project
#. get_router_hosting_port
#. is_router_in_namespace
#. get_router_info
```

Layer 3 Networking in Neutron - via Layer 3 agent & OpenVSwitch

This page discusses the usage of Neutron with Layer 3 functionality enabled.

Neutron logical network setup

```
vagrant@bionic64:~/devstack$ openstack network list
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+
| ID                                   | Name    | Subnets                ␣
↪                                                    |
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+
| 6ece2847-971b-487a-9c7b-184651ebbc82 | public  | 0d9c4261-4046-462f-9d92-
↪64fb89bc3ae6, 9e90b059-da97-45b8-8cb8-f9370217e181 |
| 713bae25-8276-4e0a-a453-e59a1d65425a | private | 6fa3bab9-103e-45d5-872c-
↪91f21b52ceda, c5c9f5c2-145d-46d2-a513-cf675530eaed |
+--------------------------------------+---------+------------------------
↪----------------------------------------------------+

vagrant@bionic64:~/devstack$ openstack subnet list
+--------------------------------------+---------------------+------------
↪-------------------------+--------------------+
| ID                                   | Name                | Network    ␣
↪                         | Subnet             |
+--------------------------------------+---------------------+------------
↪-------------------------+--------------------+
| 0d9c4261-4046-462f-9d92-64fb89bc3ae6 | public-subnet       | 6ece2847-
↪971b-487a-9c7b-184651ebbc82 | 172.24.4.0/24      |
| 6fa3bab9-103e-45d5-872c-91f21b52ceda | ipv6-private-subnet | 713bae25-
↪8276-4e0a-a453-e59a1d65425a | 2001:db8:8000::/64 |
| 9e90b059-da97-45b8-8cb8-f9370217e181 | ipv6-public-subnet  | 6ece2847-
↪971b-487a-9c7b-184651ebbc82 | 2001:db8::/64      |
| c5c9f5c2-145d-46d2-a513-cf675530eaed | private-subnet      | 713bae25-
↪8276-4e0a-a453-e59a1d65425a | 10.0.0.0/24        |
+--------------------------------------+---------------------+------------
↪-------------------------+--------------------+
```

(continues on next page)

(continued from previous page)

```
vagrant@bionic64:~/devstack$ openstack port list
+--------------------------------------+------+-------------------+--------
↪------------------------------------------------------------------------
↪-------------------+--------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                         ␣
↪                   | Status |
+--------------------------------------+------+-------------------+--------
↪------------------------------------------------------------------------
↪-------------------+--------+
| 420abb60-2a5a-4e80-90a3-3ff47742dc53 |      | fa:16:3e:2d:5c:4e | ip_
↪address='172.24.4.7', subnet_id='0d9c4261-4046-462f-9d92-64fb89bc3ae6' ␣
↪                      | ACTIVE |
|                                      |      |                   | ip_
↪address='2001:db8::1', subnet_id='9e90b059-da97-45b8-8cb8-f9370217e181' ␣
↪                      |        |
| b42d789d-c9ed-48a1-8822-839c4599301e |      | fa:16:3e:0a:ff:24 | ip_
↪address='10.0.0.1', subnet_id='c5c9f5c2-145d-46d2-a513-cf675530eaed'   ␣
↪                      | ACTIVE |
| cfff6574-091c-4d16-a54b-5b7f3eab89ce |      | fa:16:3e:a0:a3:9e | ip_
↪address='10.0.0.2', subnet_id='c5c9f5c2-145d-46d2-a513-cf675530eaed'   ␣
↪                      | ACTIVE |
|                                      |      |                   | ip_
↪address='2001:db8:8000:0:f816:3eff:fea0:a39e', subnet_id='6fa3bab9-103e-
↪45d5-872c-91f21b52ceda' |        |
| e3b7fede-277e-4c72-b66c-418a582b61ca |      | fa:16:3e:13:dd:42 | ip_
↪address='2001:db8:8000::1', subnet_id='6fa3bab9-103e-45d5-872c-
↪91f21b52ceda'                       | ACTIVE |
+--------------------------------------+------+-------------------+--------
↪------------------------------------------------------------------------
↪-------------------+--------+


vagrant@bionic64:~/devstack$ openstack subnet show c5c9f5c2-145d-46d2-a513-
↪cf675530eaed
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 10.0.0.2-10.0.0.254                  |
| cidr              | 10.0.0.0/24                          |
| created_at        | 2016-11-08T21:55:22Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 10.0.0.1                             |
| host_routes       |                                      |
| id                | c5c9f5c2-145d-46d2-a513-cf675530eaed |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | private-subnet                       |
| network_id        | 713bae25-8276-4e0a-a453-e59a1d65425a |
| project_id        | 35e3820f7490493ca9e3a5e685393298     |
| revision_number   | 2                                    |
| service_types     |                                      |
| subnetpool_id     | b1f81d96-d51d-41f3-96b5-a0da16ad7f0d |
```

(continues on next page)

(continued from previous page)

```
| updated_at          | 2016-11-08T21:55:22Z                  |
+---------------------+---------------------------------------+
```

Neutron logical router setup

```
vagrant@bionic64:~/devstack$ openstack router list
+--------------------------------------+---------+--------+-------+--------
↪-----+-------+----------------------------------+
| ID                                   | Name    | Status | State |␣
↪Distributed | HA    | Project                          |
+--------------------------------------+---------+--------+-------+--------
↪-----+-------+----------------------------------+
| 82fa9a47-246e-4da8-a864-53ea8daaed42 | router1 | ACTIVE | UP    | False ␣
↪      | False | 35e3820f7490493ca9e3a5e685393298 |
+--------------------------------------+---------+--------+-------+--------
↪-----+-------+----------------------------------+
vagrant@bionic64:~/devstack$ openstack router show router1
+-------------------------+------------------------------------------------
↪--------------------------------------------------------------------------
↪----------------------------+
| Field                   | Value                                         ␣
↪                                                                         ␣
↪                            |
+-------------------------+------------------------------------------------
↪--------------------------------------------------------------------------
↪----------------------------+
| admin_state_up          | UP                                            ␣
↪                                                                         ␣
↪                            |
| availability_zone_hints |                                               ␣
↪                                                                         ␣
↪                            |
| availability_zones      | nova                                          ␣
↪                                                                         ␣
↪                            |
| created_at              | 2016-11-08T21:55:30Z                          ␣
↪                                                                         ␣
↪                            |
| description             |                                               ␣
↪                                                                         ␣
↪                            |
| distributed             | False                                         ␣
↪                                                                         ␣
↪                            |
| external_gateway_info   | {"network_id": "6ece2847-971b-487a-9c7b-
↪184651ebbc82", "enable_snat": true, "external_fixed_ips": [{"subnet_id":
↪"0d9c4261-4046-462f-                  |
|                         | 9d92-64fb89bc3ae6", "ip_address": "172.24.4.7"}
↪, {"subnet_id": "9e90b059-da97-45b8-8cb8-f9370217e181", "ip_address":
↪"2001:db8::1"}]}                 |
| flavor_id               | None                                          ␣
↪                                                                         ␣
↪                            |
| ha                      | False                                         ␣
↪                                                                         ␣
↪                            |
```

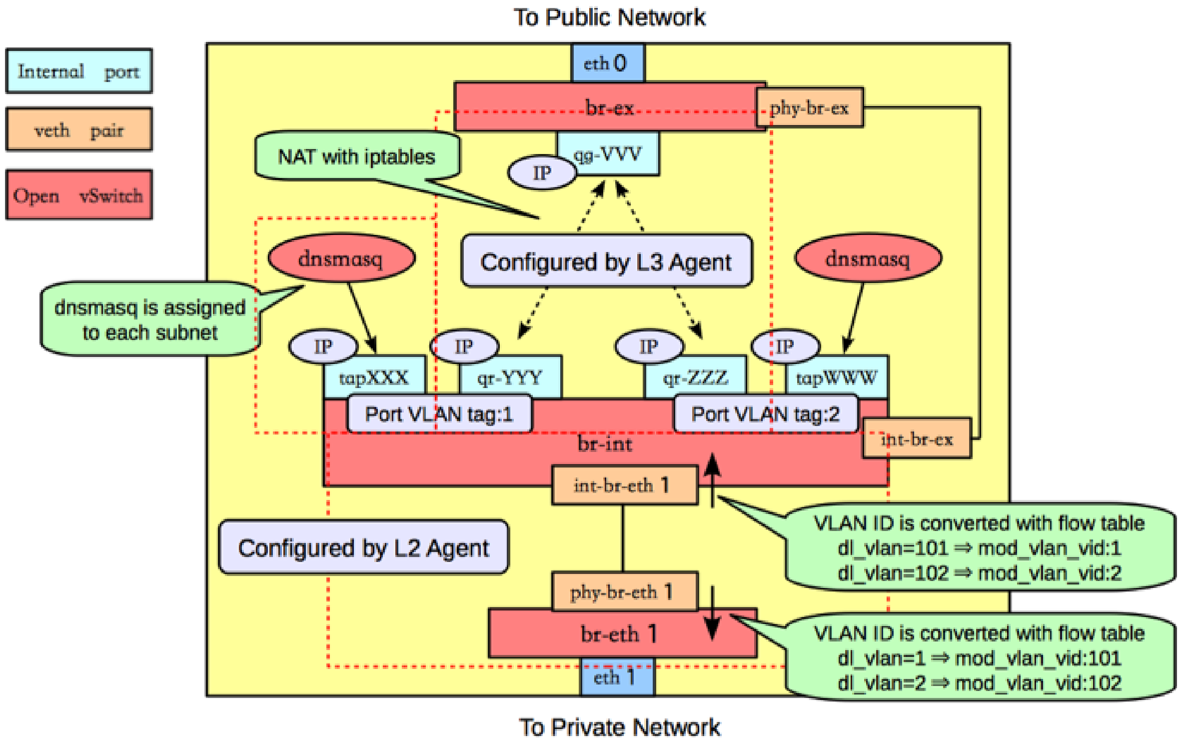
(continues on next page)

(continued from previous page)

```
| id                      | 82fa9a47-246e-4da8-a864-53ea8daaed42                                 ␣
↪                                                                                              ␣
↪                                   |
| name                    | router1                                                              ␣
↪                                                                                              ␣
↪                                   |
| project_id              | 35e3820f7490493ca9e3a5e685393298                                     ␣
↪                                                                                              ␣
↪                                   |
| revision_number         | 8                                                                    ␣
↪                                                                                              ␣
↪                                   |
| routes                  |                                                                      ␣
↪                                                                                              ␣
↪                                   |
| status                  | ACTIVE                                                               ␣
↪                                                                                              ␣
↪                                   |
| updated_at              | 2016-11-08T21:55:51Z                                                 ␣
↪                                                                                              ␣
↪                                   |
+-------------------------+---------------------------------------------------------------------------
↪---------------------------------------------------------------------------
↪------------------------------+
vagrant@bionic64:~/devstack$ openstack port list --router router1
+--------------------------------------+------+-------------------+--------
↪---------------------------------------------------------------------------
↪+--------+
| ID                                   | Name | MAC Address       | Fixed␣
↪IP Addresses                                                              ␣
↪ | Status |
+--------------------------------------+------+-------------------+--------
↪---------------------------------------------------------------------------
↪+--------+
| 420abb60-2a5a-4e80-90a3-3ff47742dc53 |      | fa:16:3e:2d:5c:4e | ip_
↪address='172.24.4.7', subnet_id='0d9c4261-4046-462f-9d92-64fb89bc3ae6'   ␣
↪    | ACTIVE |
|                                      |      |                   | ip_
↪address='2001:db8::1', subnet_id='9e90b059-da97-45b8-8cb8-f9370217e181'  ␣
↪    |        |
| b42d789d-c9ed-48a1-8822-839c4599301e |      | fa:16:3e:0a:ff:24 | ip_
↪address='10.0.0.1', subnet_id='c5c9f5c2-145d-46d2-a513-cf675530eaed'     ␣
↪    | ACTIVE |
| e3b7fede-277e-4c72-b66c-418a582b61ca |      | fa:16:3e:13:dd:42 | ip_
↪address='2001:db8:8000::1', subnet_id='6fa3bab9-103e-45d5-872c-
↪91f21b52ceda' | ACTIVE |
+--------------------------------------+------+-------------------+--------
↪---------------------------------------------------------------------------
↪+--------+
```

See the Networking Guide for more detail on the creation of networks, subnets, and routers.

Neutron Routers are realized in OpenVSwitch



router1 in the Neutron logical network is realized through a port (qr-0ba8700e-da) in OpenVSwitch - attached to br-int:

```
vagrant@bionic64:~/devstack$ sudo ovs-vsctl show
b9b27fc3-5057-47e7-ba64-0b6afe70a398
    Bridge br-int
        Port "qr-0ba8700e-da"
            tag: 1
            Interface "qr-0ba8700e-da"
                type: internal
        Port br-int
            Interface br-int
                type: internal
        Port int-br-ex
            Interface int-br-ex
        Port "tapbb60d1bb-0c"
            tag: 1
            Interface "tapbb60d1bb-0c"
                type: internal
        Port "qvob2044570-ad"
            tag: 1
            Interface "qvob2044570-ad"
        Port "int-br-eth1"
            Interface "int-br-eth1"
    Bridge "br-eth1"
        Port "phy-br-eth1"
            Interface "phy-br-eth1"
        Port "br-eth1"
            Interface "br-eth1"
```

(continues on next page)

(continued from previous page)

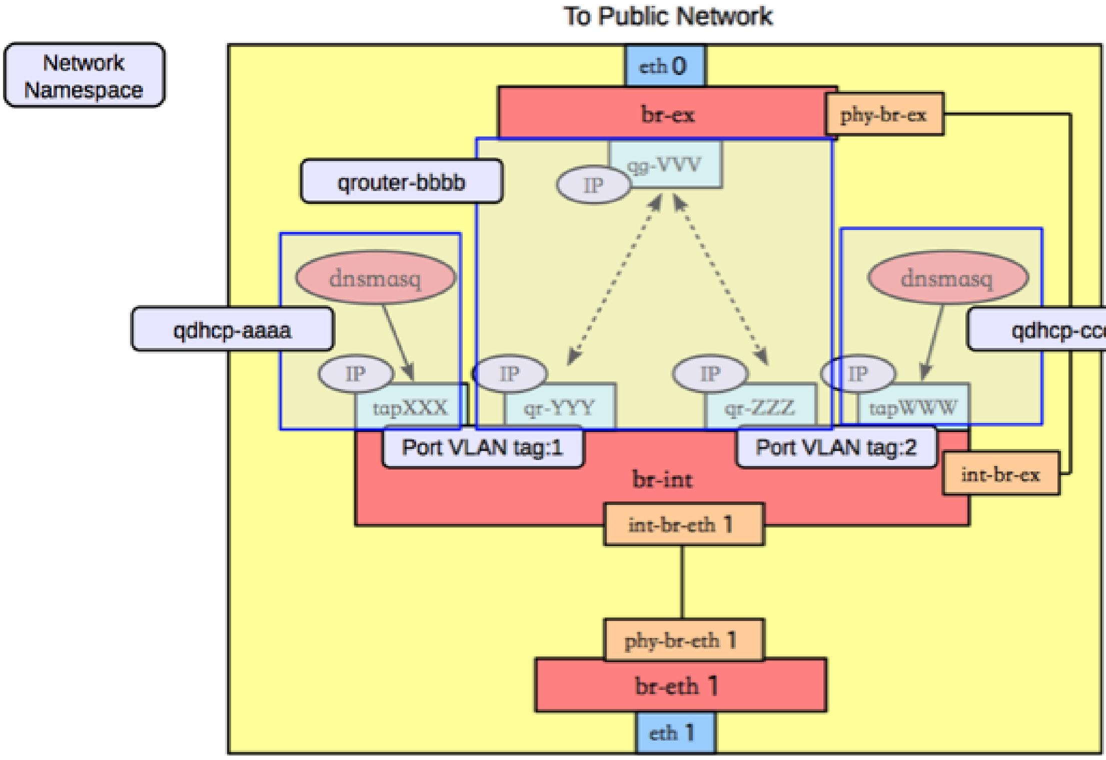
```
                type: internal
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
        Port "qg-0143bce1-08"
            Interface "qg-0143bce1-08"
                type: internal
        Port br-ex
            Interface br-ex
                type: internal
    ovs_version: "1.4.0+build0"


vagrant@bionic64:~/devstack$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-eth1         0000.e2e7fc5ccb4d       no
br-ex           0000.82ee46beaf4d       no              phy-br-ex
                                                        qg-39efb3f9-f0
                                                        qg-77e0666b-cd
br-int          0000.5e46cb509849       no              int-br-ex
                                                        qr-54c9cd83-43
                                                        qvo199abeb2-63
                                                        qvo1abbbb60-b8
                                                        tap74b45335-cc
qbr199abeb2-63          8000.ba06e5f8675c       no              ␣
↪qvb199abeb2-63
                                                        tap199abeb2-63
qbr1abbbb60-b8          8000.46a87ed4fb66       no              ␣
↪qvb1abbbb60-b8
                                                        tap1abbbb60-b8
virbr0          8000.000000000000       yes
```

Finding the router in ip/ipconfig

The neutron-l3-agent uses the Linux IP stack and iptables to perform L3 forwarding and NAT. In order to support multiple routers with potentially overlapping IP addresses, neutron-l3-agent defaults to using Linux network namespaces to provide isolated forwarding contexts. As a result, the IP addresses of routers will not be visible simply by running ip addr list or ifconfig on the node. Similarly, you will not be able to directly ping fixed IPs.

To do either of these things, you must run the command within a particular routers network namespace. The namespace will have the name qrouter-<UUID of the router>.



For example:

```
vagrant@bionic64:~$ openstack router list
+--------------------------------------+---------+------------------------
↪--------------------------------------------------+
| ID                                   | Name    | Status | State |␣
↪Distributed | HA    | Project                          |
+--------------------------------------+---------+------------------------
↪--------------------------------------------------+
| ad948c6e-afb6-422a-9a7b-0fc44cbb3910 | router1 | Active | UP    | True  ␣
↪      | False | 35e3820f7490493ca9e3a5e685393298 |
+--------------------------------------+---------+------------------------
↪--------------------------------------------------+
vagrant@bionic64:~/devstack$ sudo ip netns exec qrouter-ad948c6e-afb6-422a-
↪9a7b-0fc44cbb3910 ip addr list
18: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
19: qr-54c9cd83-43: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:dd:c1:8f brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global qr-54c9cd83-43
    inet6 fe80::f816:3eff:fedd:c18f/64 scope link
       valid_lft forever preferred_lft forever
20: qg-77e0666b-cd: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500␣
↪qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:1f:d3:ec brd ff:ff:ff:ff:ff:ff
    inet 192.168.27.130/28 brd 192.168.27.143 scope global qg-77e0666b-cd
    inet6 fe80::f816:3eff:fe1f:d3ec/64 scope link
       valid_lft forever preferred_lft forever
```

Provider Networking

Neutron can also be configured to create provider networks.

L3 agent extensions

See *L3 agent extensions*.

Further Reading

- Packet Pushers - Neutron Network Implementation on Linux
- OpenStack Networking Guide
- Neutron - Layer 3 API extension
- Darragh OReilly - The Quantum L3 router and floating IPs

Live-migration

Lets consider a VM with one port migrating from host1 with nova-compute1, neutron-l2-agent1 and neutron-l3-agent1 to host2 with nova-compute2 and neutron-l2-agent2 and neutron-l3agent2.

Since the VM that is about to migrate is hosted by nova-compute1, nova sends the live-migration order to nova-compute1 through RPC.

Nova Live Migration consists of the following stages:

- Pre-live-migration
- Live-migration-operation
- Post-live-migration

Pre-live-migration actions

Nova-compute1 will first ask nova-compute2 to perform pre-live-migration actions with a synchronous RPC call. Nova-compute2 will use neutron REST API to retrieve the list of VMs ports. Then, it calls its vif driver to create the VMs port (VIF) using plug_vifs().

In the case Open vSwitch Hybrid plug is used, Neutron-l2-agent2 will detect this new VIF, request the device details from the neutron server and configure it accordingly. However, ports status wont change, since this port is not bound to nova-compute2.

Nova-compute1 calls setup_networks_on_hosts. This updates the Neutron ports binding:profile with the information of the target host. The port update RPC message sent out by Neutron server will be received by neutron-l3-agent2, which proactively sets up the DVR router.

If pre-live-migration fails, nova rollbacks and port is removed from host2. If pre-live-migration succeeds, nova proceeds with live-migration-operation.

Potential error cases related to networking

- Plugging the VIFs on host2 fails

  As Live migration operation was not yet started, the instance resides active on host1.

Live-migration-operation

Once nova-compute2 has performed pre-live-migration actions, nova-compute1 can start the live-migration. This results in the creation of the VM and its corresponding tap interface on node 2.

In the case Open vSwitch normal plug, linux bridge or MacVTap is being used, Neutron-l2-agent2 will detect this new tap device and configure it accordingly. However, ports status wont change, since this port is not bound to nova-compute2.

As soon as the instance is active on host2, the original instance on host1 gets removed and with it the corresponding tap device. Assuming OVS-hybrid plug is NOT used, Neutron-l2-agent1 detects the removal and tells the neutron server to set the ports status to DOWN state with RPC messages.

There is no rollback if failure happens in live-migration-operation stage. TBD: Error are handled by the post-live-migration stage.

Potential error cases related to networking

- Some host devices that are specified in the instance definition are not present on the target host. Migration fails before it really started. This can happen with MacVTap agent. See bug https://bugs.launchpad.net/bugs/1550400

Post-live-migration actions

Once live-migration succeeded, both nova-compute1 and nova-compute2 perform post-live-migration actions. Nova-compute1 which is aware of the success will send a RPC cast to nova-compute2 to tell it to perform post-live-migration actions.

On host2, nova-compute2 sends a REST call update_port(binding=host2, profile={}) to the neutron server to tell it to update the ports binding. This will clear the port binding information and move the ports status to DOWN. The ML2 plugin will then try to rebind the port according to its new host. This update_port REST call always triggers a port-update RPC fanout message to every neutron-l2-agent. Since neutron-l2-agent2 is now hosting the port, it will take this message into account and re-synchronize the port by asking the neutron server details about it through RPC messages. This will move the port from DOWN status to BUILD, and then back to ACTIVE. This update also removes the migrating_to value from the portbindng dictionary. Its not clearing it totally, like indicated by {}, but just removing the migrating_to key and value.

On host1, nova-compute1 calls its vif driver to unplug the VMs port.

Assuming, Open vSwitch Hybrid plug is used, Neutron-l2-agent1 detects the removal and tells the neutron server to set the ports status to DOWN state with RPC messages. For all other cases this happens as soon as the instance and its tap device got destroyed on host1, like described in *Live-migration-operation*.

If neutron didnt already processed the REST call update_port(binding=host2), the port status will effectively move to BUILD and then to DOWN. Otherwise, the port is bound to host2, and neutron wont change the port status since its not bound the host that is sending RPC messages.

There is no rollback if failure happens in post-live-migration stage. In the case of an error, the instance is set into ERROR state.

Potential error cases related to networking

- Portbinding for host2 fails

  If this happens, the vif_type of the port is set to binding_failed. When Nova tries to recreated the domain.xml on the migration target it will stumble over this invalid vif_type and fail. The instance is put into ERROR state.

Post-Copy Migration

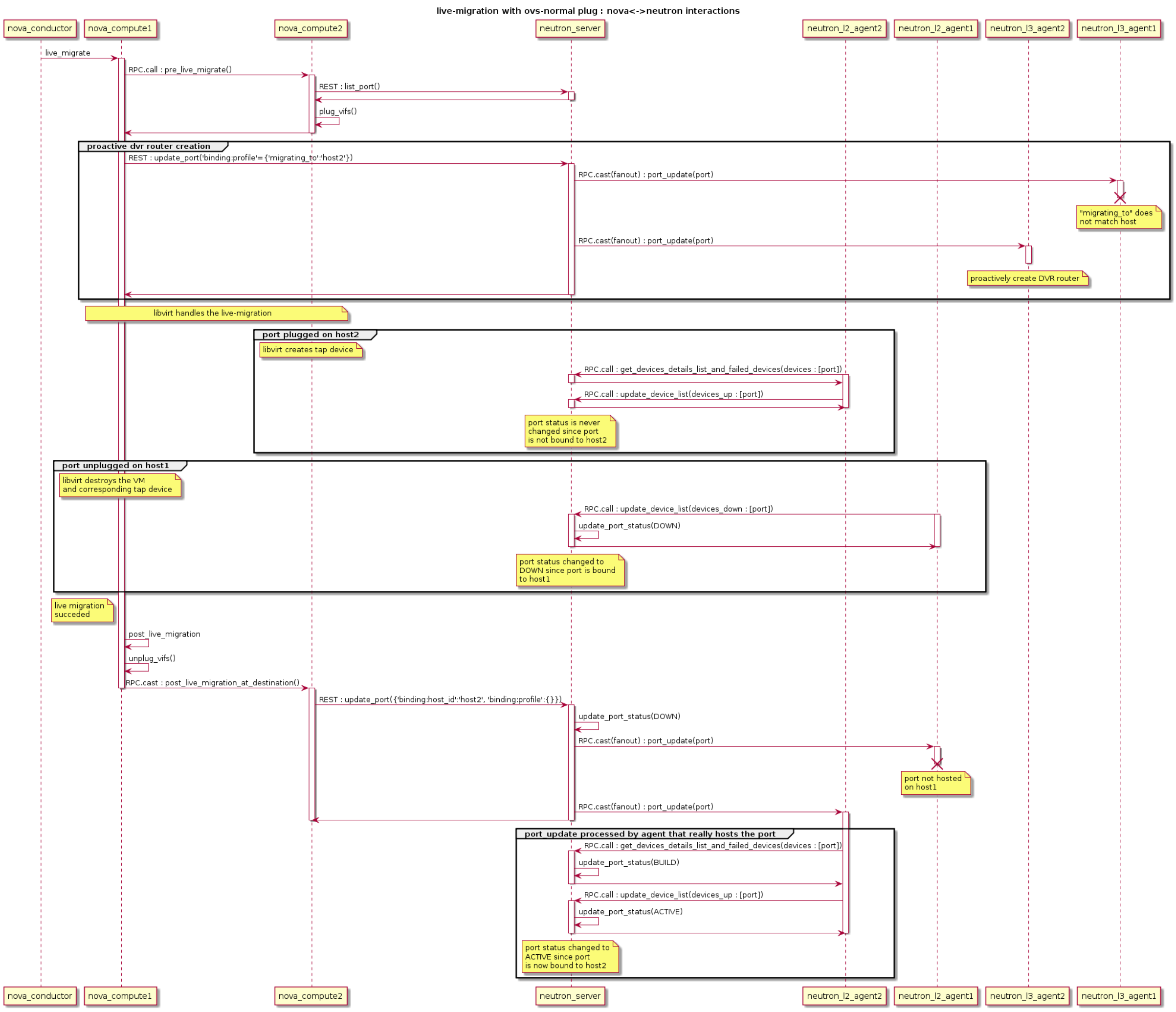
Usually, Live Migration is executed as pre-copy migration. The instance is active on host1 until nearly all memory has been copied to host2. If a certain threshold of copied memory is met, the instance on the source gets paused, the rest of the memory copied over and the instance started on the target. The challenge with this approach is, that migration might take a infinite amount of time, when the instance is heavily writing to memory.

This issue gets solved with post-copy migration. At some point in time, the instance on host2 will be set to active, although still a huge amount of memory pages reside only on host1. The phase that starts now is called the post_copy phase. If the instance tries to access a memory page that has not yet been transferred, libvirt/qemu takes care of moving this page to the target immediately. New pages will only be written to the source. With this approach the migration operation takes a finite amount of time.

Today, the rebinding of the port from host1 to host2 happens in the post_live_migration phase, after migration finished. This is fine for the pre-copy case, as the time windows between the activation of the instance on the target and the binding of the port to the target is pretty small. This becomes more problematic for the post-copy migration case. The instance becomes active on the target pretty early but the portbinding still happens after migration finished. During this time window, the instance might not be reachable via the network. This should be solved with bug https://bugs.launchpad.net/nova/+bug/1605016

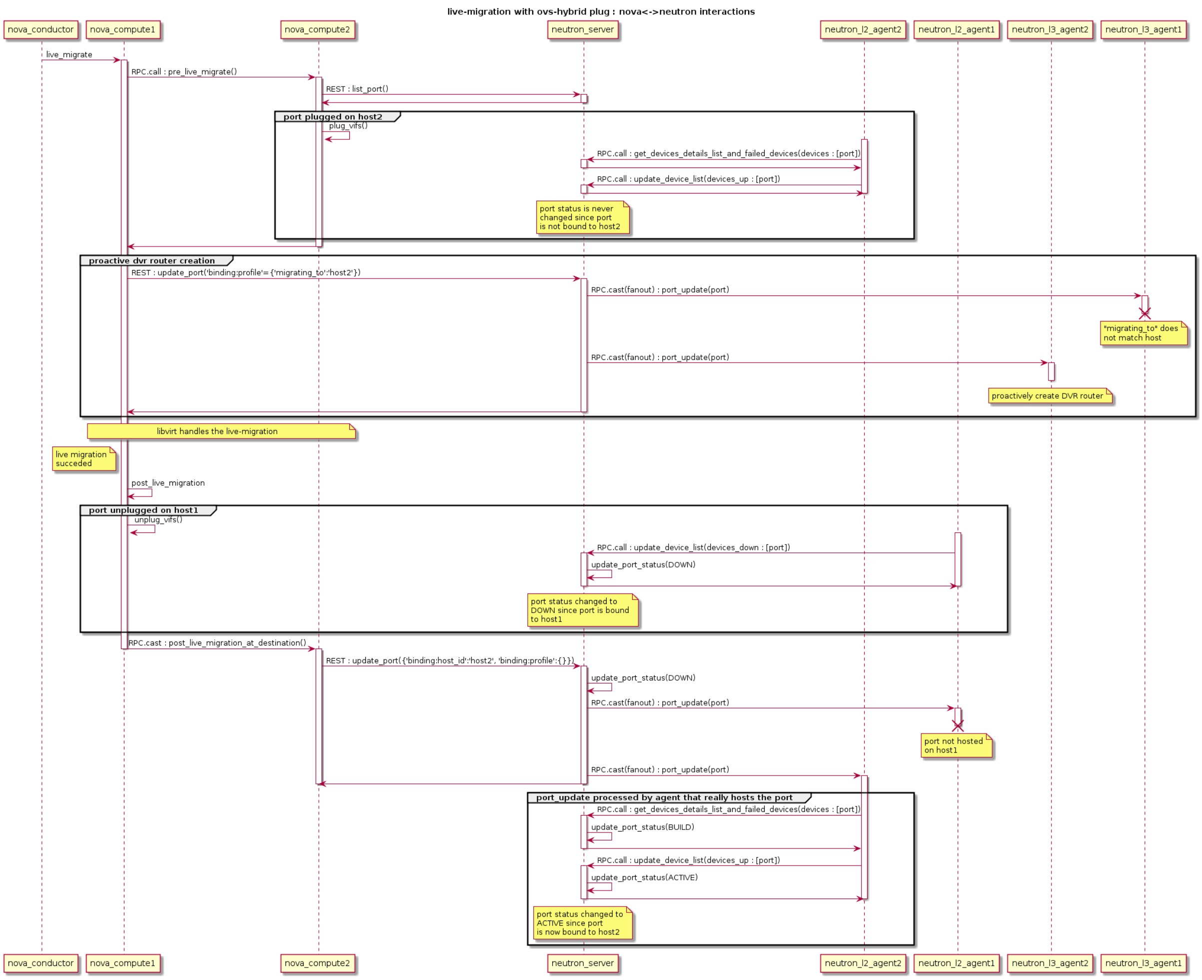
Flow Diagram

OVS Normal plug, Linux bridge, MacVTap, SR-IOV



OVS-Hybrid plug

The sequence with RPC messages from neutron-l2-agent processed first is described in the following UML sequence diagram



ML2 Extension Manager

The extension manager for ML2 was introduced in Juno (more details can be found in the approved spec). The features allows for extending ML2 resources without actually having to introduce cross cutting concerns to ML2. The mechanism has been applied for a number of use cases, and extensions that currently use this frameworks are available under ml2/extensions.

Network IP Availability Extension

This extension is an information-only API that allows a user or process to determine the amount of IPs that are consumed across networks and their subnets allocation pools. Each network and embedded subnet returns with values for **used_ips** and **total_ips** making it easy to determine how much of your networks IP space is consumed.

This API provides the ability for network administrators to periodically list usage (manual or automated) in order to preemptively add new network capacity when thresholds are exceeded.

Important Note:

This API tracks a networks consumable IPs. Whats the distinction? After a network and its subnets are created, consumable IPs are:

- Consumed in the subnets allocations (derives used IPs)

- Consumed from the subnets allocation pools (derives total IPs)

This API tracks consumable IPs so network administrators know when their subnets IP pools (and ultimately a networks) IPs are about to run out. This API does not account reserved IPs such as a subnets gateway IP or other reserved or unused IPs of a subnets cidr that are consumed as a result of the subnet creation itself.

API Specification

Availability for all networks

GET /v2.0/network-ip-availabilities

```
Request to url: v2.0/network-ip-availabilities
  headers: {'content-type': 'application/json', 'X-Auth-Token': 'SOME_AUTH_
↪TOKEN'}
```

Example response

```
Response:
  HTTP/1.1 200 OK
  Content-Type: application/json; charset=UTF-8
```


```
{
    "network_ip_availabilities": [
        {
            "network_id": "f944c153-3f46-417b-a3c2-487cd9a456b9",
            "network_name": "net1",
            "subnet_ip_availability": [
                {
                    "cidr": "10.0.0.0/24",
                    "ip_version": 4,
                    "subnet_id": "46b1406a-8373-454c-8eb8-500a09eb77fb",
                    "subnet_name": "",
                    "total_ips": 253,
                    "used_ips": 3
                }
            ],
            "tenant_id": "test-project",
            "total_ips": 253,
            "used_ips": 3
        },
        {
            "network_id": "47035bae-4f29-4fef-be2e-2941b72528a8",
            "network_name": "net2",
            "subnet_ip_availability": [],
            "tenant_id": "test-project",
            "total_ips": 0,
            "used_ips": 0
        },
        {
            "network_id": "2e3ea0cd-c757-44bf-bb30-42d038687e3f",
            "network_name": "net3",
            "subnet_ip_availability": [
```


(continues on next page)

(continued from previous page)


```
                {
                    "cidr": "40.0.0.0/24",
                    "ip_version": 4,
                    "subnet_id": "aab6b35c-16b5-489c-a5c7-fec778273495",
                    "subnet_name": "",
                    "total_ips": 253,
                    "used_ips": 2
                }
            ],
            "tenant_id": "test-project",
            "total_ips": 253,
            "used_ips": 2
        }
    ]
}
```


Availability by network ID

GET /v2.0/network-ip-availabilities/{network_uuid}

```
Request to url: /v2.0/network-ip-availabilities/aba3b29b-c119-4b45-afbd-
↪88e500acd970
  headers: {'content-type': 'application/json', 'X-Auth-Token': 'SOME_AUTH_
↪TOKEN'}
```

Example response

```
Response:
  HTTP/1.1 200 OK
  Content-Type: application/json; charset=UTF-8
```


```
{
    "network_ip_availability": {
        "network_id": "f944c153-3f46-417b-a3c2-487cd9a456b9",
        "network_name": "net1",
        "subnet_ip_availability": [
            {
                "cidr": "10.0.0.0/24",
                "ip_version": 4,
                "subnet_name": "",
                "subnet_id": "46b1406a-8373-454c-8eb8-500a09eb77fb",
                "total_ips": 253,
                "used_ips": 3
            }
        ],
        "tenant_id": "test-project",
        "total_ips": 253,
        "used_ips": 3
    }
}
```

Supported Query Filters

This API currently supports the following query parameters:

- **network_id**: Returns availability for the network matching the network ID. Note: This query (?network_id={network_id_guid})is roughly equivalent to *Availability by network ID* section except it returns the plural response form as a list rather than as an item.
- **network_name**: Returns availability for network matching the provided name
- **tenant_id**: Returns availability for all networks owned by the provided project ID.
- **ip_version**: Filters network subnets by those supporting the supplied ip version. Values can be either 4 or 6.

Query filters can be combined to further narrow results and what is returned will match all criteria. When a parameter is specified more than once, it will return results that match both. Examples:

```
# Fetch IPv4 availability for a specific project uuid
GET /v2.0/network-ip-availabilities?ip_version=4&tenant_id=example-project-
↪uuid

# Fetch multiple networks by their ids
GET /v2.0/network-ip-availabilities?network_id=uuid_sample_1&network_
↪id=uuid_sample_2
```

Objects in neutron

Object versioning is a key concept in achieving rolling upgrades. Since its initial implementation by the nova community, a versioned object model has been pushed to an oslo library so that its benefits can be shared across projects.

Oslo VersionedObjects (aka OVO) is a database facade, where you define the middle layer between software and the database schema. In this layer, a versioned object per database resource is created with a strict data definition and version number. With OVO, when you change the database schema, the version of the object also changes and a backward compatible translation is provided. This allows different versions of software to communicate with one another (via RPC).

OVO is also commonly used for RPC payload versioning. OVO creates versioned dictionary messages by defining a strict structure and keeping strong typing. Because of it, you can be sure of what is sent and how to use the data on the receiving end.

Usage of objects

CRUD operations

Objects support CRUD operations: `create()`, `get_object()` and `get_objects()` (equivalent of `read`), `update()`, `delete()`, `update_objects()`, and `delete_objects()`. The nature of OVO is, when any change is applied, OVO tracks it. After calling `create()` or `update()`, OVO detects this and changed fields are saved in the database. Please take a look at simple object usage scenarios using example of DNSNameServer:

```
# to create an object, you can pass the attributes in constructor:
dns = DNSNameServer(context, address='asd', subnet_id='xxx', order=1)
dns.create()

# or you can create a dict and pass it as kwargs:
dns_data = {'address': 'asd', 'subnet_id': 'xxx', 'order': 1}
dns = DNSNameServer(context, **dns_data)
dns.create()

# for fetching multiple objects:
dnses = DNSNameServer.get_objects(context)
# will return list of all dns name servers from DB

# for fetching objects with substrings in a string field:
from neutron_lib.objects import utils as obj_utils
dnses = DNSNameServer.get_objects(context, address=obj_utils.
↪StringContains('10.0.0'))
# will return list of all dns name servers from DB that has '10.0.0' in␣
↪their addresses

# to update fields:
dns = DNSNameServer.get_object(context, address='asd', subnet_id='xxx')
dns.order = 2
dns.update()

# if you don't care about keeping the object, you can execute the update
# without fetch of the object state from the underlying persistent layer
count = DNSNameServer.update_objects(
    context, {'order': 3}, address='asd', subnet_id='xxx')

# to remove object with filter arguments:
filters = {'address': 'asd', 'subnet_id': 'xxx'}
DNSNameServer.delete_objects(context, **filters)
```

Filter, sort and paginate

The `NeutronDbObject` class has strict validation on which field sorting and filtering can happen. When calling `get_objects()`, `count()`, `update_objects()`, `delete_objects()` and `objects_exist()`, `validate_filters()` is invoked, to see if its a supported filter criterion (which is by default non-synthetic fields only). Additional filters can be defined using `register_filter_hook_on_model()`. This will add the requested string to valid filter names in object implementation. It is optional.

In order to disable filter validation, `validate_filters=False` needs to be passed as an argument in aforementioned methods. It was added because the default behaviour of the neutron API is to accept everything at API level and filter it out at DB layer. This can be used by out of tree extensions.

`register_filter_hook_on_model()` is a complementary implementation in the `NeutronDbObject` layer to DB layers `neutron_lib.db.model_query.register_hook()`, which adds support for extra filtering during construction of SQL query. When extension defines extra query hook, it needs to be registered using the objects `register_filter_hook_on_model()`, if it is not already included in the objects `fields`.

To limit or paginate results, `Pager` object can be used. It accepts `sorts` (list of `(key, direction)` tuples), `limit`, `page_reverse` and `marker` keywords.

```
# filtering

# to get an object based on primary key filter
dns = DNSNameServer.get_object(context, address='asd', subnet_id='xxx')

# to get multiple objects
dnses = DNSNameServer.get_objects(context, subnet_id='xxx')

filters = {'subnet_id': ['xxx', 'yyy']}
dnses = DNSNameServer.get_objects(context, **filters)

# do not validate filters
dnses = DNSNameServer.get_objects(context, validate_filters=False,
                                  fake_filter='xxx')

# count the dns servers for given subnet
dns_count = DNSNameServer.count(context, subnet_id='xxx')

# sorting
# direction True == ASC, False == DESC
direction = False
pager = Pager(sorts=[('order', direction)])
dnses = DNSNameServer.get_objects(context, _pager=pager, subnet_id='xxx')
```

Defining your own object

In order to add a new object in neutron, you have to:

1. Create an object derived from `NeutronDbObject` (aka base object)
2. Add/reuse data model
3. Define fields

It is mandatory to define data model using `db_model` attribute from `NeutronDbObject`.

Fields should be defined using `oslo_versionobjects.fields` exposed types. If there is a special need to create a new type of field, you can use `common_types.py` in the `neutron.objects` directory. Example:

```
fields = {
    'id': common_types.UUIDField(),
    'name': obj_fields.StringField(),
    'subnetpool_id': common_types.UUIDField(nullable=True),
    'ip_version': common_types.IPVersionEnumField()
}
```

`VERSION` is mandatory and defines the version of the object. Initially, set the `VERSION` field to 1.0. Change `VERSION` if fields or their types are modified. When you change the version of objects being exposed via RPC, add method `obj_make_compatible(self, primitive, target_version)`. For example, if a new version introduces a new parameter, it needs to be removed for previous versions:

```
from oslo_utils import versionutils
```

(continues on next page)

(continued from previous page)

```
def obj_make_compatible(self, primitive, target_version):
    _target_version = versionutils.convert_version_to_tuple(target_version)
    if _target_version < (1, 1):  # version 1.1 introduces "new_parameter"
        primitive.pop('new_parameter', None)
```

In the following example the object has changed an attribute definition. For example, in version 1.1 `description` is allowed to be `None` but not in version 1.0:

```
from oslo_utils import versionutils
from oslo_versionedobjects import exception

def obj_make_compatible(self, primitive, target_version):
    _target_version = versionutils.convert_version_to_tuple(target_version)
    if _target_version < (1, 1):  # version 1.1 changes "description"
        if primitive['description'] is None:
            # "description" was not nullable before
            raise exception.IncompatibleObjectVersion(
                objver=target_version, objname='OVOName')
```

Using the first example as reference, this is how the unit test can be implemented:

```
def test_object_version_degradation_1_1_to_1_0(self):
    OVO_obj_1_1 = self._method_to_create_this_OVO()
    OVO_obj_1_0 = OVO_obj_1_1.obj_to_primitive(target_version='1.0')

    self.assertNotIn('new_parameter', OVO_obj_1_0['versioned_object.data'])
```

Note: Standard Attributes are automatically added to OVO fields in base class. Attributes[1] like `description`, `created_at`, `updated_at` and `revision_number` are added in[2].

`primary_keys` is used to define the list of fields that uniquely identify the object. In case of database backed objects, its usually mapped onto SQL primary keys. For immutable object fields that cannot be changed, there is a `fields_no_update` list, that contains `primary_keys` by default.

If there is a situation where a field needs to be named differently in an object than in the database schema, you can use `fields_need_translation`. This dictionary contains the name of the field in the object definition (the key) and the name of the field in the database (the value). This allows to have a different object layer representation for database persisted data. For example in IP allocation pools:

```
fields_need_translation = {
    'start': 'first_ip',  # field_ovo: field_db
    'end': 'last_ip'
}
```

The above dictionary is used in `modify_fields_from_db()` and in `modify_fields_to_db()` methods which are implemented in base class and will translate the software layer to database schema naming, and vice versa. It can also be used to rename `orm.relationship` backed object-type fields.

Most object fields are usually directly mapped to database model attributes. Sometimes its useful to expose attributes that are not defined in the model table itself, like relationships and such. In this case,

[1] https://opendev.org/openstack/neutron/tree/neutron/objects/base.py?h=stable/ocata#n258
[2] https://opendev.org/openstack/neutron/tree/neutron/db/standard_attr.py?h=stable/ocata

synthetic_fields may become handy. This object property can define a list of object fields that dont belong to the object database model and that are hence instead to be implemented in some custom way. Some of those fields map to orm.relationships defined on models, while others are completely untangled from the database layer.

When exposing existing orm.relationships as an ObjectField-typed field, you can use the foreign_keys object property that defines a link between two object types. When used, it allows objects framework to automatically instantiate child objects, and fill the relevant parent fields, based on orm.relationships defined on parent models. In order to automatically populate the synthetic_fields, the foreign_keys property is introduced. load_synthetic_db_fields() [3] method from NeutronDbObject uses foreign_keys to match the foreign key in related object and local field that the foreign key is referring to. See simplified examples:

```
class DNSNameServerSqlModel(model_base.BASEV2):
    address = sa.Column(sa.String(128), nullable=False, primary_key=True)
    subnet_id = sa.Column(sa.String(36),
                          sa.ForeignKey('subnets.id', ondelete="CASCADE"),
                          primary_key=True)

class SubnetSqlModel(model_base.BASEV2, HasId, HasProject):
    name = sa.Column(sa.String(attr.NAME_MAX_LEN))
    allocation_pools = orm.relationship(IPAllocationPoolSqlModel)
    dns_nameservers = orm.relationship(DNSNameServerSqlModel,
                                       backref='subnet',
                                       cascade='all, delete, delete-orphan
↪',
                                       lazy='subquery')

class IPAllocationPoolSqlModel(model_base.BASEV2, HasId):
    subnet_id = sa.Column(sa.String(36), sa.ForeignKey('subnets.id'))

@obj_base.VersionedObjectRegistry.register
class DNSNameServerOVO(base.NeutronDbObject):
    VERSION = '1.0'
    db_model = DNSNameServerSqlModel

    # Created based on primary_key=True in model definition.
    # The object is uniquely identified by the pair of address and
    # subnet_id fields. Override the default 'id' 1-tuple.
    primary_keys = ['address', 'subnet_id']

    # Allow to link DNSNameServerOVO child objects into SubnetOVO parent
    # object fields via subnet_id child database model attribute.
    # Used during loading synthetic fields in SubnetOVO get_objects.
    foreign_keys = {'SubnetOVO': {'subnet_id': 'id'}}

    fields = {
        'address': obj_fields.StringField(),
        'subnet_id': common_types.UUIDField(),
    }

@obj_base.VersionedObjectRegistry.register
class SubnetOVO(base.NeutronDbObject):
    VERSION = '1.0'
```

(continues on next page)

[3] https://opendev.org/openstack/neutron/tree/neutron/objects/base.py?h=stable/ocata#n516

(continued from previous page)

```
    db_model =  SubnetSqlModel

    fields = {
        'id': common_types.UUIDField(),  # HasId from model class
        'project_id': obj_fields.StringField(nullable=True),  # HasProject␣
↪from model class
        'subnet_name': obj_fields.StringField(nullable=True),
        'dns_nameservers': obj_fields.ListOfObjectsField('DNSNameServer',
                                                          nullable=True),
        'allocation_pools': obj_fields.ListOfObjectsField(
↪'IPAllocationPoolOVO',
                                                           nullable=True)
    }

    # Claim dns_nameservers field as not directly mapped into the object
    # database model table.
    synthetic_fields = ['allocation_pools', 'dns_nameservers']

    # Rename in-database subnet_name attribute into name object field
    fields_need_translation = {
        'name': 'subnet_name'
    }


@obj_base.VersionedObjectRegistry.register
class IPAllocationPoolOVO(base.NeutronDbObject):
    VERSION = '1.0'
    db_model = IPAllocationPoolSqlModel

    fields = {
        'subnet_id': common_types.UUIDField()
    }

    foreign_keys = {'SubnetOVO': {'subnet_id': 'id'}}
```

The `foreign_keys` is used in `SubnetOVO` to populate the `allocation_pools`[4] synthetic field using the `IPAllocationPoolOVO` class. Single object type may be linked to multiple parent object types, hence `foreign_keys` property may have multiple keys in the dictionary.

Note: `foreign_keys` is declared in related object `IPAllocationPoolOVO`, the same way as its done in the SQL model `IPAllocationPoolSqlModel`: `sa.ForeignKey('subnets.id')`

Note: Only single foreign key is allowed (usually parent ID), you cannot link through multiple model attributes.

It is important to remember about the nullable parameter. In the SQLAlchemy model, the nullable parameter is by default `True`, while for OVO fields, the nullable is set to `False`. Make sure you correctly map database column nullability properties to relevant object fields.

[4] https://opendev.org/openstack/neutron/tree/neutron/objects/base.py?h=stable/ocata#n542

Database session activation

By default, all objects use old `oslo.db` engine facade. To enable the new facade for a particular object, set `new_facade` class attribute to `True`:

```
@obj_base.VersionedObjectRegistry.register
class ExampleObject(base.NeutronDbObject):
    new_facade = True
```

It will make all OVO actions - `get_object`, `update`, `count` etc. - to use new `reader.using` or `writer.using` decorators to manage database transactions.

Whenever you need to open a new subtransaction in scope of OVO code, use the following database session decorators:

```
@obj_base.VersionedObjectRegistry.register
class ExampleObject(base.NeutronDbObject):

    @classmethod
    def get_object(cls, context, **kwargs):
        with cls.db_context_reader(context):
            super(ExampleObject, cls).get_object(context, **kwargs)
            # fetch more data in the same transaction

    def create(self):
        with self.db_context_writer(self.obj_context):
            super(ExampleObject, self).create()
            # apply more changes in the same transaction
```

`db_context_reader` and `db_context_writer` decorators abstract the choice of engine facade used for particular object from action implementation.

Alternatively, you can call all OVO actions under an active `reader.using` / `writer.using` context manager (or `session.begin`). In this case, OVO will pick the appropriate method to open a subtransaction.

Synthetic fields

`synthetic_fields` is a list of fields, that are not directly backed by corresponding object SQL table attributes. Synthetic fields are not limited in types that can be used to implement them.

```
fields = {
    'dhcp_agents': obj_fields.ObjectField('NetworkDhcpAgentBinding',
                                          nullable=True), # field that␣
↪contains another single NeutronDbObject of NetworkDhcpAgentBinding type
    'shared': obj_fields.BooleanField(default=False),
    'subnets': obj_fields.ListOfObjectsField('Subnet', nullable=True)
}

# All three fields do not belong to corresponding SQL table, and will be
# implemented in some object-specific way.
synthetic_fields = ['dhcp_agents', 'shared', 'subnets']
```

`ObjectField` and `ListOfObjectsField` take the name of object class as an argument.

Implementing custom synthetic fields

Sometimes you may want to expose a field on an object that is not mapped into a corresponding database model attribute, or its `orm.relationship`; or may want to expose a `orm.relationship` data in a format that is not directly mapped onto a child object type. In this case, here is what you need to do to implement custom getters and setters for the custom field. The custom method to load the synthetic fields can be helpful if the field is not directly defined in the database, OVO class is not suitable to load the data or the related object contains only the ID and property of the parent object, for example `subnet_id` and property of it: `is_external`.

In order to implement the custom method to load the synthetic field, you need to provide loading method in the OVO class and override the base class method `from_db_object()` and `obj_load_attr()`. The first one is responsible for loading the fields to object attributes when calling `get_object()` and `get_objects()`, `create()` and `update()`. The second is responsible for loading attribute when it is not set in object. Also, when you need to create related object with attributes passed in constructor, `create()` and `update()` methods need to be overwritten. Additionally `is_external` attribute can be exposed as a boolean, instead of as an object-typed field. When field is changed, but it doesnt need to be saved into database, `obj_reset_changes()` can be called, to tell OVO library to ignore that. Lets see an example:

```
@obj_base.VersionedObjectRegistry.register
class ExternalSubnet(base.NeutronDbObject):
    VERSION = '1.0'
    fields = {'subnet_id': common_types.UUIDField(),
              'is_external': obj_fields.BooleanField()}
    primary_keys = ['subnet_id']
    foreign_keys = {'Subnet': {'subnet_id': 'id'}}


@obj_base.VersionedObjectRegistry.register
class Subnet(base.NeutronDbObject):
    VERSION = '1.0'
    fields = {'external': obj_fields.BooleanField(nullable=True),}
    synthetic_fields = ['external']

    # support new custom 'external=' filter for get_objects family of
    # objects API
    def __init__(self, context=None, **kwargs):
        super(Subnet, self).__init__(context, **kwargs)
        self.add_extra_filter_name('external')

    def create(self):
        fields = self.get_changes()
        with db_api.context_manager.writer.using(context):
            if 'external' in fields:
                ExternalSubnet(context, subnet_id=self.id,
                    is_external=fields['external']).create()
            # Call to super() to create the SQL record for the object, and
            # reload its fields from the database, if needed.
            super(Subnet, self).create()

    def update(self):
        fields = self.get_changes()
        with db_api.context_manager.writer.using(context):
            if 'external' in fields:
```

(continues on next page)

(continued from previous page)

```
                # delete the old ExternalSubnet record, if present
                obj_db_api.delete_objects(
                    self.obj_context, ExternalSubnet.db_model,
                    subnet_id=self.id)
                # create the new intended ExternalSubnet object
                ExternalSubnet(context, subnet_id=self.id,
                    is_external=fields['external']).create()
            # calling super().update() will reload the synthetic fields
            # and also will update any changed non-synthetic fields, if any
            super(Subnet, self).update()

    # this method is called when user of an object accesses the attribute
    # and requested attribute is not set.
    def obj_load_attr(self, attrname):
        if attrname == 'external':
            return self._load_external()
        # it is important to call super if attrname does not match
        # because the base implementation is handling the nullable case
        super(Subnet, self).obj_load_attr(attrname)

    def _load_external(self, db_obj=None):
        # do the loading here
        if db_obj:
            # use DB model to fetch the data that may be side-loaded
            external = db_obj.external.is_external if db_obj.external else␣
↪None
        else:
            # perform extra operation to fetch the data from DB
            external_obj = ExternalSubnet.get_object(context,
                subnet_id=self.id)
            external = external_obj.is_external if external_obj else None

        # it is important to set the attribute and call obj_reset_changes
        setattr(self, 'external', external)
        self.obj_reset_changes(['external'])

    # this is defined in NeutronDbObject and is invoked during get_
↪object(s)
    # and create/update.
    def from_db_object(self, obj):
        super(Subnet, self).from_db_object(obj)
        self._load_external(obj)
```

In the above example, the `get_object(s)` methods do not have to be overwritten, because `from_db_object()` takes care of loading the synthetic fields in custom way.

Standard attributes

The standard attributes are added automatically in metaclass `DeclarativeObject`. If adding standard attribute, it has to be added in `neutron/objects/extensions/standardattributes.py`. It will be added to all relevant objects that use the `standardattributes` model. Be careful when adding something to the above, because it could trigger a change in the objects `VERSION`. For more on how standard attributes work, check[5].

RBAC handling in objects

The RBAC is implemented currently for resources like: Subnet(*), Network and QosPolicy. Subnet is a special case, because access control of Subnet depends on Network RBAC entries.

The RBAC support for objects is defined in `neutron/objects/rbac_db.py`. It defines new base class `NeutronRbacObject`. The new class wraps standard `NeutronDbObject` methods like `create()`, `update()` and `to_dict()`. It checks if the `shared` attribute is defined in the `fields` dictionary and adds it to `synthetic_fields`. Also, `rbac_db_model` is required to be defined in Network and QosPolicy classes.

`NeutronRbacObject` is a common place to handle all operations on the RBAC entries, like getting the info if resource is shared or not, creation and updates of them. By wrapping the `NeutronDbObject` methods, it is manipulating the shared attribute while `create()` and `update()` methods are called.

The example of defining the Network OVO:

```
class Network(standard_attr.HasStandardAttributes, model_base.BASEV2,
          model_base.HasId, model_base.HasProject):
    """Represents a v2 neutron network."""
    name = sa.Column(sa.String(attr.NAME_MAX_LEN))
    rbac_entries = orm.relationship(rbac_db_models.NetworkRBAC,
                                    backref='network', lazy='joined',
                                    cascade='all, delete, delete-orphan')


# Note the base class for Network OVO:
@obj_base.VersionedObjectRegistry.register
class Network(rbac_db.NeutronRbacObject):
    # Version 1.0: Initial version
    VERSION = '1.0'

    # rbac_db_model is required to be added here
    rbac_db_model = rbac_db_models.NetworkRBAC
    db_model = models_v2.Network

    fields = {
        'id': common_types.UUIDField(),
        'project_id': obj_fields.StringField(nullable=True),
        'name': obj_fields.StringField(nullable=True),
        # share is required to be added to fields
        'shared': obj_fields.BooleanField(default=False),
    }
```

[5] https://docs.openstack.org/neutron/latest/contributor/internals/db_layer.html#the-standard-attribute-table

Note: The `shared` field is not added to the `synthetic_fields`, because `NeutronRbacObject` requires to add it by itself, otherwise `ObjectActionError` is raised.[6]

Extensions to neutron resources

One of the methods to extend neutron resources is to add an arbitrary value to dictionary representing the data by providing `extend_(subnet|port|network)_dict()` function and defining loading method.

From DB perspective, all the data will be loaded, including all declared fields from DB relationships. Current implementation for core resources (Port, Subnet, Network etc.) is that DB result is parsed by `make_<resource>_dict()` and `extend_<resource>_dict()`. When extension is enabled, `extend_<resource>_dict()` takes the DB results and declares new fields in resulting dict. When extension is not enabled, data will be fetched, but will not be populated into resulting dict, because `extend_<resource>_dict()` will not be called.

Plugins can still use objects for some work, but then convert them to dicts and work as they please, extending the dict as they wish.

For example:

```
class TestSubnetExtension(model_base.BASEV2):
    subnet_id = sa.Column(sa.String(36),
                          sa.ForeignKey('subnets.id', ondelete="CASCADE"),
                          primary_key=True)
    value = sa.Column(sa.String(64))
    subnet = orm.relationship(
        models_v2.Subnet,
        # here is the definition of loading the extension with Subnet␣
↪model:
        backref=orm.backref('extension', cascade='delete', uselist=False))


@oslo_obj_base.VersionedObjectRegistry.register_if(False)
class TestSubnetExtensionObject(obj_base.NeutronDbObject):
    # Version 1.0: Initial version
    VERSION = '1.0'

    db_model = TestSubnetExtension

    fields = {
        'subnet_id': common_types.UUIDField(),
        'value': obj_fields.StringField(nullable=True)
    }

    primary_keys = ['subnet_id']
    foreign_keys = {'Subnet': {'subnet_id': 'id'}}


@obj_base.VersionedObjectRegistry.register
class Subnet(base.NeutronDbObject):
```

(continues on next page)

6 https://opendev.org/openstack/neutron/tree/neutron/objects/rbac_db.py?h=stable/ocata#n291

(continued from previous page)

```
    # Version 1.0: Initial version
    VERSION = '1.0'

    fields = {
        'id': common_types.UUIDField(),
        'extension': obj_fields.ObjectField(TestSubnetExtensionObject.__
↪name__,
                                            nullable=True),
    }

    synthetic_fields = ['extension']


# when defining the extend_subnet_dict function:
def extend_subnet_dict(self, session, subnet_ovo, result):
    value = subnet_ovo.extension.value if subnet_ovo.extension else ''
    result['subnet_extension'] = value
```

The above example is the ideal situation, where all extensions have objects adopted and enabled in core neutron resources.

By introducing the OVO work in tree, interface between base plugin code and registered extension functions hasnt been changed. Those still receive a SQLAlchemy model, not an object. This is achieved by capturing the corresponding database model on `get_***/create/update`, and exposing it via `<object>.db_obj`

Removal of downgrade checks over time

While the code to check object versions is meant to remain for a long period of time, in the interest of not accruing too much cruft over time, they are not intended to be permanent. OVO downgrade code should account for code that is within the upgrade window of any major OpenStack distribution. The longest currently known is for Ubuntu Cloud Archive which is to upgrade four versions, meaning during the upgrade the control nodes would be running a release that is four releases newer than what is running on the computes.

Known fast forward upgrade windows are:

- Red Hat OpenStack Platform (RHOSP): X -> X+3[7]
- SuSE OpenStack Cloud (SOC): X -> X+2[8]
- Ubuntu Cloud Archive: X -> X+4[9]

Therefore removal of OVO version downgrade code should be removed in the fifth cycle after the code was introduced. For example, if an object version was introduced in Ocata then it can be removed in Train.

[7] https://access.redhat.com/support/policy/updates/openstack/platform/
[8] https://www.suse.com/releasenotes/x86_64/SUSE-OPENSTACK-CLOUD/8/#Upgrade
[9] https://www.ubuntu.com/about/release-cycle

Backward compatibility for tenant_id

All objects can support `tenant_id` and `project_id` filters and fields at the same time; it is automatically enabled for all objects that have a `project_id` field. The base `NeutronDbObject` class has support for exposing `tenant_id` in dictionary access to the object fields (`subnet['tenant_id']`) and in `to_dict()` method. There is a `tenant_id` read-only property for every object that has `project_id` in `fields`. It is not exposed in `obj_to_primitive()` method, so it means that `tenant_id` will not be sent over RPC callback wire. When talking about filtering/sorting by `tenant_id`, the filters should be converted to expose `project_id` field. This means that for the long run, the API layer should translate it, but as temporary workaround it can be done at DB layer before passing filters to objects `get_objects()` method, for example:

```
def convert_filters(result):
    if 'tenant_id' in result:
        result['project_id'] = result.pop('tenant_id')
    return result

def get_subnets(context, filters):
    filters = convert_filters(**filters)
    return subnet_obj.Subnet.get_objects(context, **filters)
```

The `convert_filters` method is available in `neutron_lib.objects.utils`[10].

References

Open vSwitch Firewall Driver

The OVS driver has the same API as the current iptables firewall driver, keeping the state of security groups and ports inside of the firewall. Class `SGPortMap` was created to keep state consistent, and maps from ports to security groups and vice-versa. Every port and security group is represented by its own object encapsulating the necessary information.

Note: Open vSwitch firewall driver uses `register 5` for identifying the port related to the flow and `register 6` which identifies the network, used in particular for conntrack zones.

Ingress/Egress Terminology

In this document, the terms `ingress` and `egress` are relative to a VM instance connected to OVS (or a netns connected to OVS):

- `ingress` applies to traffic that will ultimately go into a VM (or into a netns), assuming it is not dropped
- `egress` applies to traffic coming from a VM (or from a netns)

```
                 .                                            .
      _______|\                                    _______|\
      \ ingress \                                  \ ingress \
```

(continues on next page)

[10] https://opendev.org/openstack/neutron-lib/tree/neutron_lib/objects/utils.py

(continued from previous page)

```
            /________  /                              /________  /
                     |/         .------------------.           |/
                     '          |                  |           '
                                |                  |------------( netns␣
↪interface )
( non-VM, non-netns      )---|        OVS          |
( interface: phy, patch )    |                     |-------------( VM interface )
              .                 |                  |    .
             /|_________        '------------------'   /|_________
            /   egress /                              /   egress /
            \  _________\                             \  _________\
             \|                                        \|
              '                                         '
```

Note that these terms are used differently in OVS code and documentation, where they are relative to the OVS bridge, with `ingress` applying to traffic as it comes into the OVS bridge, and `egress` applying to traffic as it leaves the OVS bridge.

Firewall API calls

There are two main calls performed by the firewall driver in order to either create or update a port with security groups - `prepare_port_filter` and `update_port_filter`. Both methods rely on the security group objects that are already defined in the driver and work similarly to their iptables counterparts. The definition of the objects will be described later in this document. `prepare_port_filter` must be called only once during port creation, and it defines the initial rules for the port. When the port is updated, all filtering rules are removed, and new rules are generated based on the available information about security groups in the driver.

Security group rules can be defined in the firewall driver by calling `update_security_group_rules`, which rewrites all the rules for a given security group. If a remote security group is changed, then `update_security_group_members` is called to determine the set of IP addresses that should be allowed for this remote security group. Calling this method will not have any effect on existing instance ports. In other words, if the port is using security groups and its rules are changed by calling one of the above methods, then no new rules are generated for this port. `update_port_filter` must be called for the changes to take effect.

All the machinery above is controlled by security group RPC methods, which mean the firewall driver doesnt have any logic of which port should be updated based on the provided changes, it only accomplishes actions when called from the controller.

OpenFlow rules

At first, every connection is split into ingress and egress processes based on the input or output port respectively. Each port contains the initial hardcoded flows for ARP, DHCP and established connections, which are accepted by default. To detect established connections, a flow must by marked by conntrack first with an `action=ct()` rule. An accepted flow means that ingress packets for the connection are directly sent to the port, and egress packets are left to be normally switched by the integration bridge.

Connections that are not matched by the above rules are sent to either the ingress or egress filtering table, depending on its direction. The reason the rules are based on security group rules in separate tables is to make it easy to detect these rules during removal.

Security group rules are treated differently for those without a remote group ID and those with a remote group ID. A security group rule without a remote group ID is expanded into several OpenFlow rules by the method `create_flows_from_rule_and_port`. A security group rule with a remote group ID is expressed by three sets of flows. The first two are conjunctive flows which will be described in the next section. The third set matches on the conjunction IDs and does accept actions.

Flow priorities for security group rules

The OpenFlow spec says a packet should not match against multiple flows at the same priority[1]. The firewall driver uses 8 levels of priorities to achieve this. The method `flow_priority_offset` calculates a priority for a given security group rule. The use of priorities is essential with conjunction flows, which will be described later in the conjunction flows examples.

Uses of conjunctive flows

With a security group rule with a remote group ID, flows that match on nw_src for remote_group_id addresses and match on dl_dst for port MAC addresses are needed (for ingress rules; likewise for egress rules). Without conjunction, this results in O(n*m) flows where n and m are number of ports in the remote group ID and the port security group, respectively.

A conj_id is allocated for each (remote_group_id, security_group_id, direction, ethertype, flow_priority_offset) tuple. The class `ConjIdMap` handles the mapping. The same conj_id is shared between security group rules if multiple rules belong to the same tuple above.

Conjunctive flows consist of 2 dimensions. Flows that belong to the dimension 1 of 2 are generated by the method `create_flows_for_ip_address` and are in charge of IP address based filtering specified by their remote group IDs. Flows that belong to the dimension 2 of 2 are generated by the method `create_flows_from_rule_and_port` and modified by the method `substitute_conjunction_actions`, which represents the portion of the rule other than its remote group ID.

Those dimension 2 of 2 flows are per port and contain no remote group information. When there are multiple security group rules for a port, those flows can overlap. To avoid such a situation, flows are sorted and fed to `merge_port_ranges` or `merge_common_rules` methods to rearrange them.

Rules example with explanation:

The following example presents two ports on the same host. They have different security groups and there is ICMP traffic allowed from first security group to the second security group. Ports have following attributes:

```
Port 1
  - plugged to the port 1 in OVS bridge
  - IP address: 192.168.0.1
  - MAC address: fa:16:3e:a4:22:10
  - security group 1: can send ICMP packets out
  - allowed address pair: 10.0.0.1/32, fa:16:3e:8c:84:13

Port 2
```

(continues on next page)

[1] Although OVS seems to magically handle overlapping flows under some cases, we shouldnt rely on that.

(continued from previous page)

```
  - plugged to the port 2 in OVS bridge
  - IP address: 192.168.0.2
  - MAC address: fa:16:3e:24:57:c7
  - security group 2:
    - can receive ICMP packets from security group 1
    - can receive TCP packets from security group 1
    - can receive TCP packets to port 80 from security group 2
    - can receive IP packets from security group 3
  - allowed address pair: 10.1.0.0/24, fa:16:3e:8c:84:14
```

`table 0` (LOCAL_SWITCHING) contains a low priority rule to continue packets processing in `table 60` (TRANSIENT) aka TRANSIENT table. `table 0` (LOCAL_SWITCHING) is left for use to other features that take precedence over firewall, e.g. DVR. The only requirement is that after such a feature is done with its processing, it needs to pass packets for processing to the TRANSIENT table. This TRANSIENT table distinguishes the ingress traffic from the egress traffic and loads into `register 5` a value identifying the port (for egress traffic based on the switch port number, and for ingress traffic based on the network id and destination MAC address); `register 6` contains a value identifying the network (which is also the OVSDB port tag) to isolate connections into separate conntrack zones. For VLAN networks, the physical VLAN tag will be used to act as an extra match rule to do such identifying work as well.

```
table=60,  priority=100,in_port=1 actions=load:0x1->NXM_NX_REG5[],
↪load:0x284->NXM_NX_REG6[],resubmit(,71)
table=60,  priority=100,in_port=2 actions=load:0x2->NXM_NX_REG5[],
↪load:0x284->NXM_NX_REG6[],resubmit(,71)
table=60,  priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:a4:22:10␣
↪actions=load:0x1->NXM_NX_REG5[],load:0x284->NXM_NX_REG6[],resubmit(,81)
table=60,  priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:8c:84:13␣
↪actions=load:0x1->NXM_NX_REG5[],load:0x284->NXM_NX_REG6[],resubmit(,81)
table=60,  priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:24:57:c7␣
↪actions=load:0x2->NXM_NX_REG5[],load:0x284->NXM_NX_REG6[],resubmit(,81)
table=60,  priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:8c:84:14␣
↪actions=load:0x2->NXM_NX_REG5[],load:0x284->NXM_NX_REG6[],resubmit(,81)
table=60,  priority=0 actions=NORMAL
```

The following table, `table 71` (BASE_EGRESS) implements ARP spoofing protection, IP spoofing protection, allows traffic related to IP address allocations (dhcp, dhcpv6, slaac, ndp) for egress traffic, and allows ARP replies. Also identifies not tracked connections which are processed later with information obtained from conntrack. Notice the `zone=NXM_NX_REG6[0..15]` in `actions` when obtaining information from conntrack. It says every port has its own conntrack zone defined by the value in `register 6` (OVSDB port tag identifying the network). Its there to avoid accepting established traffic that belongs to different port with same conntrack parameters.

The very first rule in `table 71` (BASE_EGRESS) is a rule removing conntrack information for a use-case where Neutron logical port is placed directly to the hypervisor. In such case kernel does conntrack lookup before packet reaches Open vSwitch bridge. Tracked packets are sent back for processing by the same table after conntrack information is cleared.

```
table=71, priority=110,ct_state=+trk actions=ct_clear,resubmit(,71)
```

Rules below allow ICMPv6 traffic for multicast listeners, neighbour solicitation and neighbour advertisement.

```
table=71, priority=95,icmp6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:11,
↪ipv6_src=fe80::11,icmp_type=130 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:11,
↪ipv6_src=fe80::11,icmp_type=131 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:11,
↪ipv6_src=fe80::11,icmp_type=132 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:11,
↪ipv6_src=fe80::11,icmp_type=135 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:11,
↪ipv6_src=fe80::11,icmp_type=136 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x2,in_port=2,dl_src=fa:16:3e:a4:22:22,
↪ipv6_src=fe80::22,icmp_type=130 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x2,in_port=2,dl_src=fa:16:3e:a4:22:22,
↪ipv6_src=fe80::22,icmp_type=131 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x2,in_port=2,dl_src=fa:16:3e:a4:22:22,
↪ipv6_src=fe80::22,icmp_type=132 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x2,in_port=2,dl_src=fa:16:3e:a4:22:22,
↪ipv6_src=fe80::22,icmp_type=135 actions=resubmit(,94)
table=71, priority=95,icmp6,reg5=0x2,in_port=2,dl_src=fa:16:3e:a4:22:22,
↪ipv6_src=fe80::22,icmp_type=136 actions=resubmit(,94)
```

Following rules implement ARP spoofing protection

```
table=71, priority=95,arp,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:10,arp_
↪spa=192.168.0.1 actions=resubmit(,94)
table=71, priority=95,arp,reg5=0x1,in_port=1,dl_src=fa:16:3e:8c:84:13,arp_
↪spa=10.0.0.1 actions=resubmit(,94)
table=71, priority=95,arp,reg5=0x2,in_port=2,dl_src=fa:16:3e:24:57:c7,arp_
↪spa=192.168.0.2 actions=resubmit(,94)
table=71, priority=95,arp,reg5=0x2,in_port=2,dl_src=fa:16:3e:8c:84:14,arp_
↪spa=10.1.0.0/24 actions=resubmit(,94)
```

DHCP and DHCPv6 traffic is allowed to instance but DHCP servers are blocked on instances.

```
table=71, priority=80,udp,reg5=0x1,in_port=1,tp_src=68,tp_dst=67␣
↪actions=resubmit(,73)
table=71, priority=80,udp6,reg5=0x1,in_port=1,tp_src=546,tp_dst=547␣
↪actions=resubmit(,73)
table=71, priority=70,udp,reg5=0x1,in_port=1,tp_src=67,tp_dst=68␣
↪actions=resubmit(,93)
table=71, priority=70,udp6,reg5=0x1,in_port=1,tp_src=547,tp_dst=546␣
↪actions=resubmit(,93)
table=71, priority=80,udp,reg5=0x2,in_port=2,tp_src=68,tp_dst=67␣
↪actions=resubmit(,73)
table=71, priority=80,udp6,reg5=0x2,in_port=2,tp_src=546,tp_dst=547␣
↪actions=resubmit(,73)
table=71, priority=70,udp,reg5=0x2,in_port=2,tp_src=67,tp_dst=68␣
↪actions=resubmit(,93)
table=71, priority=70,udp6,reg5=0x2,in_port=2,tp_src=547,tp_dst=546␣
↪actions=resubmit(,93)
```

Flowing rules obtain conntrack information for valid IP and MAC address combinations. All other packets are dropped.

```
table=71, priority=65,ip,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:10,nw_
↪src=192.168.0.1 actions=ct(table=72,zone=NXM_NX_REG6[0..15])
```

(continues on next page)

(continued from previous page)

```
table=71, priority=65,ip,reg5=0x1,in_port=1,dl_src=fa:16:3e:8c:84:13,nw_
↪src=10.0.0.1 actions=ct(table=72,zone=NXM_NX_REG6[0..15])
table=71, priority=65,ip,reg5=0x2,in_port=2,dl_src=fa:16:3e:24:57:c7,nw_
↪src=192.168.0.2 actions=ct(table=72,zone=NXM_NX_REG6[0..15])
table=71, priority=65,ip,reg5=0x2,in_port=2,dl_src=fa:16:3e:8c:84:14,nw_
↪src=10.1.0.0/24 actions=ct(table=72,zone=NXM_NX_REG6[0..15])
table=71, priority=65,ipv6,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4:22:10,
↪ipv6_src=fe80::f816:3eff:fea4:2210 actions=ct(table=72,zone=NXM_NX_
↪REG6[0..15])
table=71, priority=65,ipv6,reg5=0x2,in_port=2,dl_src=fa:16:3e:24:57:c7,
↪ipv6_src=fe80::f816:3eff:fe24:57c7 actions=ct(table=72,zone=NXM_NX_
↪REG6[0..15])
table=71, priority=10,reg5=0x1,in_port=1 actions=resubmit(,93)
table=71, priority=10,reg5=0x2,in_port=2 actions=resubmit(,93)
table=71, priority=0 actions=drop
```

`table 72` (RULES_EGRESS) accepts only established or related connections, and implements rules defined by security groups. As this egress connection might also be an ingress connection for some other port, its not switched yet but eventually processed by the ingress pipeline.

All established or new connections defined by security group rules are `accepted`, which will be explained later. All invalid packets are dropped. In the case below we allow all ICMP egress traffic.

```
table=72, priority=75,ct_state=+est-rel-rpl,icmp,reg5=0x1␣
↪actions=resubmit(,73)
table=72, priority=75,ct_state=+new-est,icmp,reg5=0x1 actions=resubmit(,73)
table=72, priority=50,ct_state=+inv+trk actions=resubmit(,93)
```

Important on the flows below is the `ct_mark=0x1`. Flows that were marked as not existing anymore by rule introduced later will value this value. Those are typically connections that were allowed by some security group rule and the rule was removed.

```
table=72, priority=50,ct_mark=0x1,reg5=0x1 actions=resubmit(,93)
table=72, priority=50,ct_mark=0x1,reg5=0x2 actions=resubmit(,93)
```

All other connections that are not marked and are established or related are allowed.

```
table=72, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,reg5=0x1␣
↪actions=resubmit(,94)
table=72, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,reg5=0x2␣
↪actions=resubmit(,94)
table=72, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,ct_mark=0,
↪reg5=0x1 actions=resubmit(,94)
table=72, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,ct_mark=0,
↪reg5=0x2 actions=resubmit(,94)
```

In the following, flows are marked established connections that werent matched in the previous flows, which means they dont have accepting security group rule anymore.

```
table=72, priority=40,ct_state=-est,reg5=0x1 actions=resubmit(,93)
table=72, priority=40,ct_state=+est,reg5=0x1 actions=ct(commit,zone=NXM_NX_
↪REG6[0..15],exec(load:0x1->NXM_NX_CT_MARK[]))
table=72, priority=40,ct_state=-est,reg5=0x2 actions=resubmit(,93)
table=72, priority=40,ct_state=+est,reg5=0x2 actions=ct(commit,zone=NXM_NX_
↪REG6[0..15],exec(load:0x1->NXM_NX_CT_MARK[]))
```

(continues on next page)

(continued from previous page)

```
table=72, priority=0 actions=drop
```

In following `table 73` (ACCEPT_OR_INGRESS) are all detected ingress connections sent to ingress pipeline. Since the connection was already accepted by egress pipeline, all remaining egress connections are sent to normal floodnlearn switching in `table 94` (ACCEPTED_EGRESS_TRAFFIC_NORMAL).

```
table=73, priority=100,reg6=0x284,dl_dst=fa:16:3e:a4:22:10␣
↪actions=load:0x1->NXM_NX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,dl_dst=fa:16:3e:8c:84:13␣
↪actions=load:0x1->NXM_NX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,dl_dst=fa:16:3e:24:57:c7␣
↪actions=load:0x2->NXM_NX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,dl_dst=fa:16:3e:8c:84:14␣
↪actions=load:0x2->NXM_NX_REG5[],resubmit(,81)
table=73, priority=90,ct_state=+new-est,reg5=0x1 actions=ct(commit,
↪zone=NXM_NX_REG6[0..15]),resubmit(,91)
table=73, priority=90,ct_state=+new-est,reg5=0x2 actions=ct(commit,
↪zone=NXM_NX_REG6[0..15]),resubmit(,91)
table=73, priority=80,reg5=0x1 actions=resubmit(,94)
table=73, priority=80,reg5=0x2 actions=resubmit(,94)
table=73, priority=0 actions=drop
```

`table 81` (BASE_INGRESS) is similar to `table 71` (BASE_EGRESS), allows basic ingress traffic for obtaining IP address and ARP queries. Note that vlan tag must be removed by adding `strip_vlan` to actions list, prior to injecting packet directly to port. Not tracked packets are sent to obtain conntrack information.

```
table=81, priority=100,arp,reg5=0x1 actions=strip_vlan,output:1
table=81, priority=100,arp,reg5=0x2 actions=strip_vlan,output:2
table=81, priority=100,icmp6,reg5=0x1,icmp_type=130 actions=strip_vlan,
↪output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=131 actions=strip_vlan,
↪output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=132 actions=strip_vlan,
↪output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=135 actions=strip_vlan,
↪output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=136 actions=strip_vlan,
↪output:1
table=81, priority=100,icmp6,reg5=0x2,icmp_type=130 actions=strip_vlan,
↪output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=131 actions=strip_vlan,
↪output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=132 actions=strip_vlan,
↪output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=135 actions=strip_vlan,
↪output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=136 actions=strip_vlan,
↪output:2
table=81, priority=95,udp,reg5=0x1,tp_src=67,tp_dst=68 actions=strip_vlan,
↪output:1
table=81, priority=95,udp6,reg5=0x1,tp_src=547,tp_dst=546 actions=strip_
↪vlan,output:1
table=81, priority=95,udp,reg5=0x2,tp_src=67,tp_dst=68 actions=strip_vlan,
↪output:2
```

(continues on next page)

(continued from previous page)

```
table=81, priority=95,udp6,reg5=0x2,tp_src=547,tp_dst=546 actions=strip_
↪vlan,output:2
table=81, priority=90,ct_state=-trk,ip,reg5=0x1 actions=ct(table=82,
↪zone=NXM_NX_REG6[0..15])
table=81, priority=90,ct_state=-trk,ipv6,reg5=0x1 actions=ct(table=82,
↪zone=NXM_NX_REG6[0..15])
table=81, priority=90,ct_state=-trk,ip,reg5=0x2 actions=ct(table=82,
↪zone=NXM_NX_REG6[0..15])
table=81, priority=90,ct_state=-trk,ipv6,reg5=0x2 actions=ct(table=82,
↪zone=NXM_NX_REG6[0..15])
table=81, priority=80,ct_state=+trk,reg5=0x1 actions=resubmit(,82)
table=81, priority=80,ct_state=+trk,reg5=0x2 actions=resubmit(,82)
table=81, priority=0 actions=drop
```

Similarly to `table 72` (RULES_EGRESS), `table 82` (RULES_INGRESS) accepts established and related connections. In this case we allow all ICMP traffic coming from `security group 1` which is in this case only `port 1`. The first four flows match on the IP addresses, and the next two flows match on the ICMP protocol. These six flows define conjunction flows, and the next two define actions for them.

```
table=82, priority=71,ct_state=+est-rel-rpl,ip,reg6=0x284,nw_src=192.168.0.
↪1 actions=conjunction(18,1/2)
table=82, priority=71,ct_state=+est-rel-rpl,ip,reg6=0x284,nw_src=10.0.0.1␣
↪actions=conjunction(18,1/2)
table=82, priority=71,ct_state=+new-est,ip,reg6=0x284,nw_src=192.168.0.1␣
↪actions=conjunction(19,1/2)
table=82, priority=71,ct_state=+new-est,ip,reg6=0x284,nw_src=10.0.0.1␣
↪actions=conjunction(19,1/2)
table=82, priority=71,ct_state=+est-rel-rpl,icmp,reg5=0x2␣
↪actions=conjunction(18,2/2)
table=82, priority=71,ct_state=+new-est,icmp,reg5=0x2␣
↪actions=conjunction(19,2/2)
table=82, priority=71,conj_id=18,ct_state=+est-rel-rpl,ip,reg5=0x2␣
↪actions=strip_vlan,output:2
table=82, priority=71,conj_id=19,ct_state=+new-est,ip,reg5=0x2␣
↪actions=ct(commit,zone=NXM_NX_REG6[0..15]),strip_vlan,output:2,resubmit(,
↪92)
table=82, priority=50,ct_state=+inv+trk actions=resubmit(,93)
```

There are some more security group rules with remote group IDs. Next we look at TCP related ones. Excerpt of flows that correspond to those rules are:

```
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=0x60/
↪0xffe0 actions=conjunction(22,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=0x60/0xffe0␣
↪actions=conjunction(23,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=0x40/
↪0xfff0 actions=conjunction(22,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=0x40/0xfff0␣
↪actions=conjunction(23,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=0x58/
↪0xfff8 actions=conjunction(22,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=0x58/0xfff8␣
↪actions=conjunction(23,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=0x54/
↪0xfffc actions=conjunction(22,2/2)
```

(continues on next page)

(continued from previous page)

```
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=0x54/0xfffc␣
↪actions=conjunction(23,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=0x52/
↪0xfffe actions=conjunction(22,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=0x52/0xfffe␣
↪actions=conjunction(23,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=80␣
↪actions=conjunction(22,2/2),conjunction(14,2/2)
table=82, priority=73,ct_state=+est-rel-rpl,tcp,reg5=0x2,tp_dst=81␣
↪actions=conjunction(22,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=80␣
↪actions=conjunction(23,2/2),conjunction(15,2/2)
table=82, priority=73,ct_state=+new-est,tcp,reg5=0x2,tp_dst=81␣
↪actions=conjunction(23,2/2)
```

Only dimension 2/2 flows are shown here, as the other are similar to the previous ICMP example. There are many more flows but only the port ranges that covers from 64 to 127 are shown for brevity.

The conjunction IDs 14 and 15 correspond to packets from the security group 1, and the conjunction IDs 22 and 23 correspond to those from the security group 2. These flows are from the following security group rules,

```
- can receive TCP packets from security group 1
- can receive TCP packets to port 80 from security group 2
```

and these rules have been processed by `merge_port_ranges` into:

```
- can receive TCP packets to port != 80 from security group 1
- can receive TCP packets to port 80 from security group 1 or 2
```

before translating to flows so that there is only one matching flow even when the TCP destination port is 80.

The remaining is a L4 protocol agnostic rule.

```
table=82, priority=70,ct_state=+est-rel-rpl,ip,reg5=0x2␣
↪actions=conjunction(24,2/2)
table=82, priority=70,ct_state=+new-est,ip,reg5=0x2 actions=conjunction(25,
↪2/2)
```

Any IP packet that matches the previous TCP flows matches one of these flows, but the corresponding security group rules have different remote group IDs. Unlike the above TCP example, theres no convenient way of expressing `protocol != TCP` or `icmp_code != 1`. So the OVS firewall uses a different priority than the previous TCP flows so as not to mix them up.

The mechanism for dropping connections that are not allowed anymore is the same as in `table 72` (RULES_EGRESS).

```
table=82, priority=50,ct_mark=0x1,reg5=0x1 actions=resubmit(,93)
table=82, priority=50,ct_mark=0x1,reg5=0x2 actions=resubmit(,93)
table=82, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,reg5=0x1␣
↪actions=strip_vlan,output:1
table=82, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,reg5=0x2␣
↪actions=strip_vlan,output:2
table=82, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,ct_mark=0,
↪reg5=0x1 actions=strip_vlan,output:1
```

(continues on next page)

(continued from previous page)

```
table=82, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,ct_mark=0,
↪reg5=0x2 actions=strip_vlan,output:2
table=82, priority=40,ct_state=-est,reg5=0x1 actions=resubmit(,93)
table=82, priority=40,ct_state=+est,reg5=0x1 actions=ct(commit,zone=NXM_NX_
↪REG6[0..15],exec(load:0x1->NXM_NX_CT_MARK[]))
table=82, priority=40,ct_state=-est,reg5=0x2 actions=resubmit(,93)
table=82, priority=40,ct_state=+est,reg5=0x2 actions=ct(commit,zone=NXM_NX_
↪REG6[0..15],exec(load:0x1->NXM_NX_CT_MARK[]))
table=82, priority=0 actions=drop
```

Note: Conntrack zones on a single node are now based on the network to which a port is plugged in. That makes a difference between traffic on hypervisor only and east-west traffic. For example, if a port has a VIP that was migrated to a port on a different node, then the new port wont contain conntrack information about previous traffic that happened with VIP.

OVS firewall integration points

There are three tables where packets are sent once after going through the OVS firewall pipeline. The tables can be used by other mechanisms that are supposed to work with the OVS firewall, typically L2 agent extensions.

Egress pipeline

Packets are sent to `table 91` (ACCEPTED_EGRESS_TRAFFIC) and `table 94` (ACCEPTED_EGRESS_TRAFFIC_NORMAL) when they are considered accepted by the egress pipeline, and they will be processed so that they are forwarded to their destination by being submitted to a NORMAL action, that results in Ethernet flood/learn processing.

Two tables are used to differentiate between the first packets of a connection and the following packets. This was introduced for performance reasons to allow the logging extension to only log the first packets of a connection. Only the first accepted packet of each connection session will go to `table 91` (ACCEPTED_EGRESS_TRAFFIC) and the following ones will go to `table 94` (ACCEPTED_EGRESS_TRAFFIC_NORMAL).

Note that `table 91` (ACCEPTED_EGRESS_TRAFFIC) merely resubmits to `table 94` (ACCEPTED_EGRESS_TRAFFIC_NORMAL) that contains the actual NORMAL action; this allows to have a single place where the NORMAL action can be overridden by other components (currently used by `networking-bagpipe` driver for `networking-bgpvpn`).

Ingress pipeline

The first packet of each connection accepted by the ingress pipeline is sent to `table 92` (ACCEPTED_INGRESS_TRAFFIC). The default action in this table is DROP because at this point the packets have already been delivered to their destination port. This integration point is essentially provided for the logging extension.

Packets are sent to `table 93` (DROPPED_TRAFFIC) if processing by the ingress filtering concluded that they should be dropped.

Upgrade path from iptables hybrid driver

During an upgrade, the agent will need to re-plug each instances tap device into the integration bridge while trying to not break existing connections. One of the following approaches can be taken:

1) Pause the running instance in order to prevent a short period of time where its network interface does not have firewall rules. This can happen due to the firewall driver calling OVS to obtain information about OVS the port. Once the instance is paused and no traffic is flowing, we can delete the qvo interface from integration bridge, detach the tap device from the qbr bridge and plug the tap device back into the integration bridge. Once this is done, the firewall rules are applied for the OVS tap interface and the instance is started from its paused state.

2) Set drop rules for the instances tap interface, delete the qbr bridge and related veths, plug the tap device into the integration bridge, apply the OVS firewall rules and finally remove the drop rules for the instance.

3) Compute nodes can be upgraded one at a time. A free node can be switched to use the OVS firewall, and instances from other nodes can be live-migrated to it. Once the first node is evacuated, its firewall driver can be then be switched to the OVS driver.

Neutron Open vSwitch vhost-user support

Neutron supports using Open vSwitch + DPDK vhost-user interfaces directly in the OVS ML2 driver and agent. The current implementation relies on a multiple configuration values and includes runtime verification of Open vSwitchs capability to provide these interfaces.

The OVS agent detects the capability of the underlying Open vSwitch installation and passes that information over RPC via the agent configurations dictionary. The ML2 driver uses this information to select the proper VIF type and binding details.

Platform requirements

- OVS 2.4.0+
- DPDK 2.0+

Configuration

```
[OVS]
datapath_type=netdev
vhostuser_socket_dir=/var/run/openvswitch
```

When OVS is running with DPDK support enabled, and the `datapath_type` is set to `netdev`, then the OVS ML2 driver will use the `vhost-user` VIF type and pass the necessary binding details to use OVS+DPDK and vhost-user sockets. This includes the `vhostuser_socket_dir` setting, which must match the directory passed to `ovs-vswitchd` on startup.

What about the networking-ovs-dpdk repo?

The networking-ovs-dpdk repo will continue to exist and undergo active development. This feature just removes the necessity for a separate ML2 driver and OVS agent in the networking-ovs-dpdk repo. The networking-ovs-dpdk project also provides a devstack plugin which also allows automated CI, a Puppet module, and an OpenFlow-based security group implementation.

Neutron Plugin Architecture

Salvatore Orlando: How to write a Neutron Plugin (if you really need to)

Plugin API

v2 Neutron Plug-in API specification.

NeutronPluginBaseV2 provides the definition of minimum set of methods that needs to be implemented by a v2 Neutron Plug-in.

class neutron.neutron_plugin_base_v2.**NeutronPluginBaseV2**

abstract create_network(*context*, *network*)
Create a network.

Create a network, which represents an L2 network segment which can have a set of subnets and ports associated with it.

Parameters

- **context** neutron api request context
- **network** dictionary describing the network, with keys as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. All keys will be populated.

abstract create_port(*context*, *port*)
Create a port.

Create a port, which is a connection point of a device (e.g., a VM NIC) to attach to a L2 neutron network.

Parameters

- **context** neutron api request context

- **port** dictionary describing the port, with keys as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. All keys will be populated.

abstract create_subnet(*context*, *subnet*)
Create a subnet.

Create a subnet, which represents a range of IP addresses that can be allocated to devices

Parameters

- **context** neutron api request context
- **subnet** dictionary describing the subnet, with keys as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. All keys will be populated.

create_subnetpool(*context*, *subnetpool*)
Create a subnet pool.

Parameters

- **context** neutron api request context
- **subnetpool** Dictionary representing the subnetpool to create.

abstract delete_network(*context*, *id*)
Delete a network.

Parameters

- **context** neutron api request context
- **id** UUID representing the network to delete.

abstract delete_port(*context*, *id*)
Delete a port.

Parameters

- **context** neutron api request context
- **id** UUID representing the port to delete.

abstract delete_subnet(*context*, *id*)
Delete a subnet.

Parameters

- **context** neutron api request context
- **id** UUID representing the subnet to delete.

delete_subnetpool(*context*, *id*)
Delete a subnet pool.

Parameters

- **context** neutron api request context
- **id** The UUID of the subnet pool to delete.

abstract get_network(*context*, *id*, *fields=None*)
Retrieve a network.

Parameters

- **context** neutron api request context
- **id** UUID representing the network to fetch.
- **fields** a list of strings that are valid keys in a network dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

abstract get_networks(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

Retrieve a list of networks.

The contents of the list depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a network as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.
- **fields** a list of strings that are valid keys in a network dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

get_networks_count(*context*, *filters=None*)

Return the number of networks.

The result depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a network as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.

NOTE: this method is optional, as it was not part of the originally defined plugin API.

abstract get_port(*context*, *id*, *fields=None*)

Retrieve a port.

Parameters

- **context** neutron api request context
- **id** UUID representing the port to fetch.

- **fields** a list of strings that are valid keys in a port dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

abstract get_ports (*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

Retrieve a list of ports.

The contents of the list depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a port as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.
- **fields** a list of strings that are valid keys in a port dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

get_ports_count (*context*, *filters=None*)

Return the number of ports.

The result depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a network as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.

Note: this method is optional, as it was not part of the originally defined plugin API.

abstract get_subnet (*context*, *id*, *fields=None*)

Retrieve a subnet.

Parameters

- **context** neutron api request context
- **id** UUID representing the subnet to fetch.
- **fields** a list of strings that are valid keys in a subnet dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

get_subnetpool(*context*, *id*, *fields=None*)
Show a subnet pool.

Parameters

- **context** neutron api request context
- **id** The UUID of the subnetpool to show.

get_subnetpools(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)
Retrieve list of subnet pools.

abstract get_subnets(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)
Retrieve a list of subnets.

The contents of the list depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a subnet as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.
- **fields** a list of strings that are valid keys in a subnet dictionary as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Only these fields will be returned.

get_subnets_count(*context*, *filters=None*)
Return the number of subnets.

The result depends on the identity of the user making the request (as indicated by the context) as well as any filters.

Parameters

- **context** neutron api request context
- **filters** a dictionary with keys that are valid keys for a network as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`. Values in this dictionary are an iterable containing values that will be used for an exact match comparison for that value. Each result returned by this function will have matched one of the values for each key in filters.

Note: this method is optional, as it was not part of the originally defined plugin API.

---

has_native_datastore()
Return True if the plugin uses Neutrons native datastore.

Note: plugins like ML2 should override this method and return True.

rpc_state_report_workers_supported()
Return whether the plugin supports state report RPC workers.

Note: this method is optional, as it was not part of the originally defined plugin API.

rpc_workers_supported()
Return whether the plugin supports multiple RPC workers.

A plugin that supports multiple RPC workers should override the start_rpc_listeners method to ensure that this method returns True and that start_rpc_listeners is called at the appropriate time. Alternately, a plugin can override this method to customize detection of support for multiple rpc workers

Note: this method is optional, as it was not part of the originally defined plugin API.

start_rpc_listeners()
Start the RPC listeners.

Most plugins start RPC listeners implicitly on initialization. In order to support multiple process RPC, the plugin needs to expose control over when this is started.

Note: this method is optional, as it was not part of the originally defined plugin API.

start_rpc_state_reports_listener()
Start the RPC listeners consuming state reports queue.

This optional method creates rpc consumer for REPORTS queue only.

Note: this method is optional, as it was not part of the originally defined plugin API.

abstract update_network(*context*, *id*, *network*)
Update values of a network.

Parameters

- **context** neutron api request context
- **id** UUID representing the network to update.
- **network** dictionary with keys indicating fields to update. valid keys are those that have a value of True for allow_put as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`.

abstract update_port(*context*, *id*, *port*)
Update values of a port.

Parameters

- **context** neutron api request context
- **id** UUID representing the port to update.
- **port** dictionary with keys indicating fields to update. valid keys are those that have a value of True for allow_put as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`.

abstract update_subnet (*context, id, subnet*)
Update values of a subnet.

Parameters

- **context** neutron api request context
- **id** UUID representing the subnet to update.
- **subnet** dictionary with keys indicating fields to update. valid keys are those that have a value of True for allow_put as listed in the `RESOURCE_ATTRIBUTE_MAP` object in `neutron/api/v2/attributes.py`.

update_subnetpool (*context, id, subnetpool*)
Update a subnet pool.

Parameters

- **context** neutron api request context
- **subnetpool** Dictionary representing the subnetpool attributes to update.

Authorization Policy Enforcement

As most OpenStack projects, Neutron leverages oslo_policy[1]. However, since Neutron loves to be special and complicate every developers life, it also augments oslo_policy capabilities by:

- A wrapper module with its own API: neutron.policy;
- The ability of adding fine-grained checks on attributes for resources in request bodies;
- The ability of using the policy engine to filter out attributes in responses;
- Adding some custom rule checks beyond those defined in oslo_policy;

This document discusses Neutron-specific aspects of policy enforcement, and in particular how the enforcement logic is wired into API processing. For any other information please refer to the developer documentation for oslo_policy[2].

[1] Oslo policy module
[2] Oslo policy developer

Authorization workflow

The Neutron API controllers perform policy checks in two phases during the processing of an API request:

- Request authorization, immediately before dispatching the request to the plugin layer for POST, PUT, and DELETE, and immediately after returning from the plugin layer for GET requests;
- Response filtering, when building the response to be returned to the API consumer.

Request authorization

The aim of this step is to authorize processing for a request or reject it with an error status code. This step uses the neutron.policy.enforce routine. This routine raises oslo_policy.PolicyNotAuthorized when policy enforcement fails. The Neutron REST API controllers catch this exception and return:

- A 403 response code on a POST request or an PUT request for an object owned by the project submitting the request;
- A 403 response for failures while authorizing API actions such as add_router_interface;
- A 404 response for DELETE, GET and all other PUT requests.

For DELETE operations the resource must first be fetched. This is done invoking the same _item[3] method used for processing GET requests. This is also true for PUT operations, since the Neutron API implements PATCH semantics for PUTs. The criteria to evaluate are built in the _build_match_rule[4] routine. This routine takes in input the following parameters:

- The action to be performed, in the <operation>_<resource> form, e.g.: create_network
- The data to use for performing checks. For POST operations this could be a partial specification of the object, whereas it is always a full specification for GET, PUT, and DELETE requests, as resource data are retrieved before dispatching the call to the plugin layer.
- The collection name for the resource specified in the previous parameter; for instance, for a network it would be the networks.

The _build_match_rule routine returns a oslo_policy.RuleCheck instance built in the following way:

- Always add a check for the action being performed. This will match a policy like create_network in policy.json;
- Return for GET operations; more detailed checks will be performed anyway when building the response;
- For each attribute which has been explicitly specified in the request create a rule matching policy names in the form <operation>_<resource>:<attribute> rule, and link it with the previous rule with an And relationship (using oslo_policy.AndCheck); this step will be performed only if the enforce_policy flag is set to True in the resource attribute descriptor (usually found in a data structure called RESOURCE_ATTRIBUTE_MAP);

[3] API controller item method
[4] Policy engines build_match_rule method

- If the attribute is a composite one then further rules will be created; These will match policy names in the form <operation>_<resource>:<attribute>:<sub_attribute>. An And relationship will be used in this case too.

As all the rules to verify are linked by And relationships, all the policy checks should succeed in order for a request to be authorized. Rule verification is performed by `oslo_policy` with no customization from the Neutron side.

Response Filtering

Some Neutron extensions, like the provider networks one, add some attribute to resources which are however not meant to be consumed by all clients. This might be because these attributes contain implementation details, or are meant only to be used when exchanging information between services, such as Nova and Neutron;

For this reason the policy engine is invoked again when building API responses. This is achieved by the `_exclude_attributes_by_policy`[5] method in `neutron.api.v2.base.Controller`;

This method, for each attribute in the response returned by the plugin layer, first checks if the `is_visible` flag is True. In that case it proceeds to checking policies for the attribute; if the policy check fails the attribute is added to a list of attributes that should be removed from the response before returning it to the API client.

The neutron.policy API

The `neutron.policy` module exposes a simple API whose main goal if to allow the REST API controllers to implement the authorization workflow discussed in this document. It is a bad practice to call the policy engine from within the plugin layer, as this would make request authorization dependent on configured plugins, and therefore make API behaviour dependent on the plugin itself, which defies Neutron tenet of being backend agnostic.

The neutron.policy API exposes the following routines:

- `init` Initializes the policy engine loading rules from the json policy (files). This method can safely be called several times.
- `reset` Clears all the rules currently configured in the policy engine. It is called in unit tests and at the end of the initialization of core API router[6] in order to ensure rules are loaded after all the extensions are loaded.
- `refresh` Combines init and reset. Called when a SIGHUP signal is sent to an API worker.
- `set_rules` Explicitly set policy engines rules. Used only in unit tests.
- `check` Perform a check using the policy engine. Builds match rules as described in this document, and then evaluates the resulting rule using oslo_policys policy engine. Returns True if the checks succeeds, false otherwise.
- `enforce` Operates like the check routine but raises if the check in oslo_policy fails.
- `check_is_admin` Enforce the predefined context_is_admin rule; used to determine the is_admin property for a neutron context.

[5] exclude_attributes_by_policy method
[6] Policy reset in neutron.api.v2.router

- `check_is_advsvc` Enforce the predefined context_is_advsvc rule; used to determine the is_advsvc property for a neutron context.

Neutron specific policy rules

Neutron provides two additional policy rule classes in order to support the augmented authorization capabilities it provides. They both extend `oslo_policy.RuleCheck` and are registered using the `oslo_policy.register` decorator.

OwnerCheck: Extended Checks for Resource Ownership

This class is registered for rules matching the `tenant_id` keyword and overrides the generic check performed by oslo_policy in this case. It uses for those cases where neutron needs to check whether the project submitting a request for a new resource owns the parent resource of the one being created. Current usages of `OwnerCheck` include, for instance, creating and updating a subnet. This class supports the extension parent resources owner check which the parent resource introduced by service plugins. Such as router and floatingip owner check for `router` service plugin. Developers can register the extension resource name and service plugin name which were registered in neutron-lib into `EXT_PARENT_RESOURCE_MAPPING` which is located in `neutron_lib.services.constants`.

The check, performed in the `__call__` method, works as follows:

- verify if the target field is already in the target data. If yes, then simply verify whether the value for the target field in target data is equal to value for the same field in credentials, just like `oslo_policy.GenericCheck` would do. This is also the most frequent case as the target field is usually `tenant_id`;
- if the previous check failed, extract a parent resource type and a parent field name from the target field. For instance `networks:tenant_id` identifies the `tenant_id` attribute of the `network` resource. For extension parent resource case, `ext_parent:tenant_id` identifies the `tenant_id` attribute of the registered extension resource in `EXT_PARENT_RESOURCE_MAPPING`;
- if no parent resource or target field could be identified raise a `PolicyCheckError` exception;
- Retrieve a parent foreign key from the `_RESOURCE_FOREIGN_KEYS` data structure in `neutron.policy`. This foreign key is simply the attribute acting as a primary key in the parent resource. A `PolicyCheckError` exception will be raised if such parent foreign key cannot be retrieved;
- Using the core plugin, retrieve an instance of the resource having parent foreign key as an identifier;
- Finally, verify whether the target field in this resource matches the one in the initial request data. For instance, for a port create request, verify whether the `tenant_id` of the port data structure matches the `tenant_id` of the network where this port is being created.

FieldCheck: Verify Resource Attributes

This class is registered with the policy engine for rules matching the field keyword, and provides a way to perform fine grained checks on resource attributes. For instance, using this class of rules it is possible to specify a rule for granting every project read access to shared resources.

In policy.json, a FieldCheck rules is specified in the following way:

```
> field: <resource>:<field>=<value>
```

This will result in the initialization of a FieldCheck that will check for `<field>` in the target resource data, and return `True` if it is equal to `<value>` or return `False` is the `<field>` either is not equal to `<value>` or does not exist at all.

Guidance for Neutron API developers

When developing REST APIs for Neutron it is important to be aware of how the policy engine will authorize these requests. This is true both for APIs served by Neutron core and for the APIs served by the various Neutron stadium services.

- If an attribute of a resource might be subject to authorization checks then the `enforce_policy` attribute should be set to `True`. While setting this flag to `True` for each attribute is a viable strategy, it is worth noting that this will require a call to the policy engine for each attribute, thus consistently increasing the time required to complete policy checks for a resource. This could result in a scalability issue, especially in the case of list operations retrieving a large number of resources;
- Some resource attributes, even if not directly used in policy checks might still be required by the policy engine. This is for instance the case of the `tenant_id` attribute. For these attributes the `required_by_policy` attribute should always set to `True`. This will ensure that the attribute is included in the resource data sent to the policy engine for evaluation;
- The `tenant_id` attribute is a fundamental one in Neutron API request authorization. The default policy, `admin_or_owner`, uses it to validate if a project owns the resource it is trying to operate on. To this aim, if a resource without a tenant_id is created, it is important to ensure that ad-hoc authZ policies are specified for this resource.
- There is still only one check which is hardcoded in Neutrons API layer: the check to verify that a project owns the network on which it is creating a port. This check is hardcoded and is always executed when creating a port, unless the network is shared. Unfortunately a solution for performing this check in an efficient way through the policy engine has not yet been found. Due to its nature, there is no way to override this check using the policy engine.
- It is strongly advised to not perform policy checks in the plugin or in the database management classes. This might lead to divergent API behaviours across plugins. Also, it might leave the Neutron DB in an inconsistent state if a request is not authorized after it has already been dispatched to the backend.

Notes

- No authorization checks are performed for requests coming from the RPC over AMQP channel. For all these requests a neutron admin context is built, and the plugins will process them as such.
- For `PUT` and `DELETE` requests a 404 error is returned on request authorization failures rather than a 403, unless the project submitting the request own the resource to update or delete. This is to avoid conditions in which an API client might try and find out other projects resource identifiers by sending out `PUT` and `DELETE` requests for random resource identifiers.
- There is no way at the moment to specify an `OR` relationship between two attributes of a given resource (eg.: `port.name == 'meh' or port.status == 'DOWN'`), unless the rule with the or condition is explicitly added to the policy.json file.
- `OwnerCheck` performs a plugin access; this will likely require a database access, but since the behaviour is implementation specific it might also imply a round-trip to the backend. This class of checks, when involving retrieving attributes for parent resources should be used very sparingly.
- In order for `OwnerCheck` rules to work, parent resources should have an entry in `neutron.policy._RESOURCE_FOREIGN_KEYS`; moreover the resource must be managed by the core plugin (ie: the one defined in the core_plugin configuration variable)

Policy-in-Code support

Guideline on defining in-code policies

The following is the guideline of policy definitions.

Ideally we should define all available policies, but in the neutron policy enforcement it is not practical to define all policies because we check all attributes of a target resource in the *Response Filtering*. Considering this, we have the special guidelines for get operation.

- All policies of `<action>_<resource>` must be defined for all types of operations. Valid actions are `create`, `update`, `delete` and `get`.
- `get_<resourceS>` (get plural) is unnecessary. The neutron API layer use a single form policy `get_<resource>` when listing resources[7][8].
- Member actions for individual resources must be defined. For example, `add_router_interface` of `router` resource.
- All policies with attributes on create, update and delete actions must be defined. `<action>_<resource>:<attribute>(:<sub_attribute>)` policy is required for attributes with `enforce_policy` in the API definitions. Note that it is recommended to define even if a rule is same as for `<action>_<resource>` from the documentation perspective.
- For a policy with attributes of get actions like `get_<resource>:<attribute>(:<sub_attribute>)`, the following guideline is applied:
  - A policy with an attribute must be defined if the policy is different from the policy for `get_<resource>` (without attributes).

[7] https://github.com/openstack/neutron/blob/051b6b40f3921b9db4f152a54f402c402cbf138c/neutron/pecan_wsgi/hooks/policy_enforcement.py#L173

[8] https://github.com/openstack/neutron/blob/051b6b40f3921b9db4f152a54f402c402cbf138c/neutron/pecan_wsgi/hooks/policy_enforcement.py#L143

- If a policy with an attribute is same as for `get_<resource>`, there is no need to define it explicitly. This is for simplicity. We check all attributes of a target resource in the process of *Response Filtering* so it leads to a long long policy definitions for get actions in our documentation. It is not happy for operators either.
- If an attribute is marked as `enforce_policy`, it is recommended to define the corresponding policy with the attribute. This is for clarification. If an attribute is marked as `enforce_policy` in the API definitions, for example, the neutron API limits to set such attribute only to admin users but allows to retrieve a value for regular users. If policies for the attribute are different across the types of operations, it is better to define all of them explicitly.

Registering policies in neutron related projects

Policy-in-code support in neutron is a bit different from other projects because the neutron server needs to load policies in code from multiple projects. Each neutron related project should register the following two entry points `oslo.policy.policies` and `neutron.policies` in `setup.cfg` like below:

```
oslo.policy.policies =
    neutron = neutron.conf.policies:list_rules
neutron.policies =
    neutron = neutron.conf.policies:list_rules
```

The above two entries are same, but they have different purposes.

- The first entry point is a normal entry point defined by oslo.policy and it is used to generate a sample policy file[9][10].
- The second one is specific to neutron. It is used by `neutron.policy` module to load policies of neutron related projects.

`oslo.policy.policies` entry point is used by all projects which adopt oslo.policy, so we cannot determine which projects are neutron related projects, so the second entry point is required.

The recommended entry point name is a repository name: For example, neutron-fwaas for FWaaS and networking-sfc for SFC:

```
oslo.policy.policies =
    neutron-fwaas = neutron_fwaas.policies:list_rules
neutron.policies =
    neutron-fwaas = neutron_fwaas.policies:list_rules
```

Except registering the `neutron.policies` entry point, other steps to be done in each neutron related project for policy-in-code support are same for all OpenStack projects.

[9] https://docs.openstack.org/oslo.policy/latest/user/usage.html#sample-file-generation
[10] https://docs.openstack.org/oslo.policy/latest/cli/index.html#oslopolicy-sample-generator

References

Composite Object Status via Provisioning Blocks

We use the STATUS field on objects to indicate when a resource is ready by setting it to ACTIVE so external systems know when its safe to use that resource. Knowing when to set the status to ACTIVE is simple when there is only one entity responsible for provisioning a given object. When that entity has finishing provisioning, we just update the STATUS directly to active. However, there are resources in Neutron that require provisioning by multiple asynchronous entities before they are ready to be used so managing the transition to the ACTIVE status becomes more complex. To handle these cases, Neutron has the provisioning_blocks module to track the entities that are still provisioning a resource.

The main example of this is with ML2, the L2 agents and the DHCP agents. When a port is created and bound to a host, its placed in the DOWN status. The L2 agent now has to setup flows, security group rules, etc for the port and the DHCP agent has to setup a DHCP reservation for the ports IP and MAC. Before the transition to ACTIVE, both agents must complete their work or the port user (e.g. Nova) may attempt to use the port and not have connectivity. To solve this, the provisioning_blocks module is used to track the provisioning state of each agent and the status is only updated when both complete.

High Level View

To make use of the provisioning_blocks module, provisioning components should be added whenever there is work to be done by another entity before an objects status can transition to ACTIVE. This is accomplished by calling the add_provisioning_component method for each entity. Then as each entity finishes provisioning the object, the provisioning_complete must be called to lift the provisioning block.

When the last provisioning block is removed, the provisioning_blocks module will trigger a callback notification containing the object ID for the objects resource type with the event PROVISIONING_COMPLETE. A subscriber to this event can now update the status of this object to ACTIVE or perform any other necessary actions.

A normal state transition will look something like the following:

1. Request comes in to create an object
2. Logic on the Neutron server determines which entities are required to provision the object and adds a provisioning component for each entity for that object.
3. A notification is emitted to the entities so they start their work.
4. Object is returned to the API caller in the DOWN (or BUILD) state.
5. Each entity tells the server when it has finished provisioning the object. The server calls provisioning_complete for each entity that finishes.
6. When provisioning_complete is called on the last remaining entity, the provisioning_blocks module will emit an event indicating that provisioning has completed for that object.
7. A subscriber to this event on the server will then update the status of the object to ACTIVE to indicate that it is fully provisioned.

For a more concrete example, see the section below.

ML2, L2 agents, and DHCP agents

ML2 makes use of the provisioning_blocks module to prevent the status of ports from being transitioned to ACTIVE until both the L2 agent and the DHCP agent have finished wiring a port.

When a port is created or updated, the following happens to register the DHCP agents provisioning blocks:

1. The subnet_ids are extracted from the fixed_ips field of the port and then ML2 checks to see if DHCP is enabled on any of the subnets.
2. The configuration for the DHCP agents hosting the network are looked up to ensure that at least one of them is new enough to report back that it has finished setting up the port reservation.
3. If either of the preconditions above fail, a provisioning block for the DHCP agent is not added and any existing DHCP agent blocks for that port are cleared to ensure the port isnt blocked waiting for an event that will never happen.
4. If the preconditions pass, a provisioning block is added for the port under the DHCP entity.

When a port is created or updated, the following happens to register the L2 agents provisioning blocks:

1. If the port is not bound, nothing happens because we dont know yet if an L2 agent is involved so we have to wait until a port update that binds it.
2. Once the port is bound, the agent based mechanism drivers will check if they have an agent on the bound host and if the VNIC type belongs to the mechanism driver, a provisioning block is added for the port under the L2 Agent entity.

Once the DHCP agent has finished setting up the reservation, it calls dhcp_ready_on_ports via the RPC API with the port ID. The DHCP RPC handler receives this and calls provisioning_complete in the provisioning module with the port ID and the DHCP entity to remove the provisioning block.

Once the L2 agent has finished setting up the reservation, it calls the normal update_device_list (or update_device_up) via the RPC API. The RPC callbacks handler calls provisioning_complete with the port ID and the L2 Agent entity to remove the provisioning block.

On the provisioning_complete call that removes the last record, the provisioning_blocks module emits a callback PROVISIONING_COMPLETE event with the port ID. A function subscribed to this in ML2 then calls update_port_status to set the port to ACTIVE.

At this point the normal notification is emitted to Nova allowing the VM to be unpaused.

In the event that the DHCP or L2 agent is down, the port will not transition to the ACTIVE status (as is the case now if the L2 agent is down). Agents must account for this by telling the server that wiring has been completed after configuring everything during startup. This ensures that ports created on offline agents (or agents that crash and restart) eventually become active.

To account for server instability, the notifications about port wiring be complete must use RPC calls so the agent gets a positive acknowledgement from the server and it must keep retrying until either the port is deleted or it is successful.

If an ML2 driver immediately places a bound port in the ACTIVE state (e.g. after calling a backend in update_port_postcommit), this patch will not have any impact on that process.

Quality of Service

Quality of Service advanced service is designed as a service plugin. The service is decoupled from the rest of Neutron code on multiple levels (see below).

QoS extends core resources (ports, networks) without using mixins inherited from plugins but through an ml2 extension driver.

Details about the DB models, API extension, and use cases can be found here: qos spec .

Service side design

- neutron.extensions.qos: base extension + API controller definition. Note that rules are subattributes of policies and hence embedded into their URIs.
- neutron.extensions.qos_fip: base extension + API controller definition. Adds qos_policy_id to floating IP, enabling users to set/update the binding QoS policy of a floating IP.
- neutron.services.qos.qos_plugin: QoSPlugin, service plugin that implements qos extension, receiving and handling API calls to create/modify policies and rules.
- neutron.services.qos.drivers.manager: the manager that passes object actions down to every enabled QoS driver and issues RPC calls when any of the drivers require RPC push notifications.
- neutron.services.qos.drivers.base: the interface class for pluggable QoS drivers that are used to update backends about new {create, update, delete} events on any rule or policy change, including precommit events that some backends could need for synchronization reason. The drivers also declare which QoS rules, VIF drivers and VNIC types are supported.
- neutron.core_extensions.base: Contains an interface class to implement core resource (port/network) extensions. Core resource extensions are then easily integrated into interested plugins. We may need to have a core resource extension manager that would utilize those extensions, to avoid plugin modifications for every new core resource extension.
- neutron.core_extensions.qos: Contains QoS core resource extension that conforms to the interface described above.
- neutron.plugins.ml2.extensions.qos: Contains ml2 extension driver that handles core resource updates by reusing the core_extensions.qos module mentioned above. In the future, we would like to see a plugin-agnostic core resource extension manager that could be integrated into other plugins with ease.

QoS plugin implementation guide

The neutron.extensions.qos.QoSPluginBase class uses method proxies for methods relating to QoS policy rules. Each of these such methods is generic in the sense that it is intended to handle any rule type. For example, QoSPluginBase has a create_policy_rule method instead of both create_policy_dscp_marking_rule and create_policy_bandwidth_limit_rule methods. The logic behind the proxies allows a call to a plugins create_policy_dscp_marking_rule to be handled by the create_policy_rule method, which will receive a QosDscpMarkingRule object as an argument in order to execute behavior specific to the DSCP marking rule type. This approach allows new rule types to be introduced without requiring a plugin to modify code as a result. As would be expected, any subclass of QoSPluginBase must override the base classs abc.abstractmethod methods, even if to raise NotImplemented.

Supported QoS rule types

Each QoS driver has a property called supported_rule_types, where the driver exposes the rules its able to handle.

For a list of all rule types, see: neutron.services.qos.qos_consts.VALID_RULE_TYPES.

The list of supported QoS rule types exposed by neutron is calculated as the common subset of rules supported by all active QoS drivers.

Note: the list of supported rule types reported by core plugin is not enforced when accessing QoS rule resources. This is mostly because then we would not be able to create rules while at least one of the QoS driver in gate lacks support for the rules were trying to test.

Database models

QoS design defines the following two conceptual resources to apply QoS rules for a port, a network or a floating IP:

- QoS policy
- QoS rule (type specific)

Each QoS policy contains zero or more QoS rules. A policy is then applied to a network or a port, making all rules of the policy applied to the corresponding Neutron resource.

When applied through a network association, policy rules could apply or not to neutron internal ports (like router, dhcp, etc..). The QosRule base object provides a default should_apply_to_port method which could be overridden. In the future we may want to have a flag in QoSNetworkPolicyBinding or QosRule to enforce such type of application (for example when limiting all the ingress of routers devices on an external network automatically).

Each project can have at most one default QoS policy, although is not mandatory. If a default QoS policy is defined, all new networks created within this project will have assigned this policy, as long as no other QoS policy is explicitly attached during the creation process. If the default QoS policy is unset, no change to existing networks will be made.

From database point of view, following objects are defined in schema:

- QosPolicy: directly maps to the conceptual policy resource.
- QosNetworkPolicyBinding, QosPortPolicyBinding, QosFIPPolicyBinding: define attachment between a Neutron resource and a QoS policy.
- QosPolicyDefault: defines a default QoS policy per project.
- QosBandwidthLimitRule: defines the rule to limit the maximum egress bandwidth.
- QosDscpMarkingRule: defines the rule that marks the Differentiated Service bits for egress traffic.
- QosMinimumBandwidthRule: defines the rule that creates a minimum bandwidth constraint.

All database models are defined under:

- neutron.db.qos.models

QoS versioned objects

For QoS, the following neutron objects are implemented:

- QosPolicy: directly maps to the conceptual policy resource, as defined above.
- QosPolicyDefault: defines a default QoS policy per project.
- QosBandwidthLimitRule: defines the instance bandwidth limit rule type, characterized by a max kbps and a max burst kbits. This rule has also a direction parameter to set the traffic direction, from the instances point of view.
- QosDscpMarkingRule: defines the DSCP rule type, characterized by an even integer between 0 and 56. These integers are the result of the bits in the DiffServ section of the IP header, and only certain configurations are valid. As a result, the list of valid DSCP rule types is: 0, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 46, 48, and 56.
- QosMinimumBandwidthRule: defines the minimum assured bandwidth rule type, characterized by a min_kbps parameter. This rule has also a direction parameter to set the traffic direction, from the instance point of view. The only direction now implemented is egress.

Those are defined in:

- neutron.objects.qos.policy
- neutron.objects.qos.rule

For QosPolicy neutron object, the following public methods were implemented:

- get_network_policy/get_port_policy/get_fip_policy: returns a policy object that is attached to the corresponding Neutron resource.
- attach_network/attach_port/attach_floatingip: attach a policy to the corresponding Neutron resource.
- detach_network/detach_port/detach_floatingip: detach a policy from the corresponding Neutron resource.

In addition to the fields that belong to QoS policy database object itself, synthetic fields were added to the object that represent lists of rules that belong to the policy. To get a list of all rules for a specific policy, a consumer of the object can just access the corresponding attribute via:

- policy.rules

Implementation is done in a way that will allow adding a new rule list field with little or no modifications in the policy object itself. This is achieved by smart introspection of existing available rule object definitions and automatic definition of those fields on the policy class.

Note that rules are loaded in a non lazy way, meaning they are all fetched from the database on policy fetch.

For Qos<type>Rule objects, an extendable approach was taken to allow easy addition of objects for new rule types. To accommodate this, fields common to all types are put into a base class called QosRule that is then inherited into type-specific rule implementations that, ideally, only define additional fields and some other minor things.

Note that the QosRule base class is not registered with oslo.versionedobjects registry, because its not expected that generic rules should be instantiated (and to suggest just that, the base rule class is marked as ABC).

QoS objects rely on some primitive database API functions that are added in:

- neutron_lib.db.api: those can be reused to fetch other models that do not have corresponding versioned objects yet, if needed.
- neutron.db.qos.api: contains database functions that are specific to QoS models.

RPC communication

Details on RPC communication implemented in reference backend driver are discussed in a separate page.

The flow of updates is as follows:

- if a port that is bound to the agent is attached to a QoS policy, then ML2 plugin detects the change by relying on ML2 QoS extension driver, and notifies the agent about a port change. The agent proceeds with the notification by calling to get_device_details() and getting the new port dict that contains a new qos_policy_id. Each device details dict is passed into l2 agent extension manager that passes it down into every enabled extension, including QoS. QoS extension sees that there is a new unknown QoS policy for a port, so it uses ResourcesPullRpcApi to fetch the current state of the policy (with all the rules included) from the server. After that, the QoS extension applies the rules by calling into QoS driver that corresponds to the agent.
- For floating IPs, a `fip_qos` L3 agent extension was implemented. This extension receives and processes router updates. For each update, it goes over each floating IP associated to the router. If a floating IP has a QoS policy associated to it, the extension uses ResourcesPullRpcApi to fetch the policy details from the Neutron server. If the policy includes `bandwidth_limit` rules, the extension applies them to the appropriate router device by directly calling the l3_tc_lib.
- on existing QoS policy update (it includes any policy or its rules change), server pushes the new policy object state through ResourcesPushRpcApi interface. The interface fans out the serialized (dehydrated) object to any agent that is listening for QoS policy updates. If an agent have seen the policy before (it is attached to one of the ports/floating IPs it maintains), then it goes with applying the updates to the port/floating IP. Otherwise, the agent silently ignores the update.

Agent side design

Reference agents implement QoS functionality using an L2 agent extension.

- neutron.agent.l2.extensions.qos defines QoS L2 agent extension. It receives handle_port and delete_port events and passes them down into QoS agent backend driver (see below). The file also defines the QosAgentDriver interface. Note: each backend implements its own driver. The driver handles low level interaction with the underlying networking technology, while the QoS extension handles operations that are common to all agents.

For L3 agent:

- neutron.agent.l3.extensions.fip_qos defines QoS L3 agent extension. It implements the L3 agent side of floating IP rate limit. For all routers, if floating IP has QoS `bandwidth_limit` rules, the corresponding TC filters will be added to the appropriate router device, depending on the router type.

Agent backends

At the moment, QoS is supported by Open vSwitch, SR-IOV and Linux bridge ml2 drivers.

Each agent backend defines a QoS driver that implements the QosAgentDriver interface:

- Open vSwitch (QosOVSAgentDriver);
- SR-IOV (QosSRIOVAgentDriver);
- Linux bridge (QosLinuxbridgeAgentDriver).

For the Networking back ends, QoS supported rules, and traffic directions (from the VM point of view), please see the table: Networking back ends, supported rules, and traffic direction.

Open vSwitch

Open vSwitch implementation relies on the new ovs_lib OVSBridge functions:

- get_egress_bw_limit_for_port
- create_egress_bw_limit_for_port
- delete_egress_bw_limit_for_port
- get_ingress_bw_limit_for_port
- update_ingress_bw_limit_for_port
- delete_ingress_bw_limit_for_port

An egress bandwidth limit is effectively configured on the port by setting the port Interface parameters ingress_policing_rate and ingress_policing_burst.

That approach is less flexible than linux-htb, Queues and OvS QoS profiles, which we may explore in the future, but which will need to be used in combination with openflow rules.

An ingress bandwidth limit is effectively configured on the port by setting Queue and OvS QoS profile with linux-htb type for port.

The Open vSwitch DSCP marking implementation relies on the recent addition of the ovs_agent_extension_api OVSAgentExtensionAPI to request access to the integration bridge functions:

- add_flow
- mod_flow
- delete_flows
- dump_flows_for

The DSCP markings are in fact configured on the port by means of openflow rules.

Note: As of Ussuri release, the QoS rules can be applied for direct ports with hardware offload capability (switchdev), this requires Open vSwitch version 2.11.0 or newer and Linux kernel based on kernel 5.4.0 or newer.

SR-IOV

SR-IOV bandwidth limit and minimum bandwidth implementation relies on the new pci_lib function:

- set_vf_rate

As the name of the function suggests, the limit is applied on a Virtual Function (VF). This function has a parameter called rate_type and its value can be set to rate or min_tx_rate, which is for enforcing bandwidth limit or minimum bandwidth respectively.

ip link interface has the following limitation for bandwidth limit: it uses Mbps as units of bandwidth measurement, not kbps, and does not support float numbers. So in case the limit is set to something less than 1000 kbps, its set to 1 Mbps only. If the limit is set to something that does not divide to 1000 kbps chunks, then the effective limit is rounded to the nearest integer Mbps value.

Linux bridge

The Linux bridge implementation relies on the new tc_lib functions.

For egress bandwidth limit rule:

- set_filters_bw_limit
- update_filters_bw_limit
- delete_filters_bw_limit

The egress bandwidth limit is configured on the tap port by setting traffic policing on tc ingress queueing discipline (qdisc). Details about ingress qdisc can be found on lartc how-to. The reason why ingress qdisc is used to configure egress bandwidth limit is that tc is working on traffic which is visible from inside bridge perspective. So traffic incoming to bridge via tap interface is in fact outgoing from Neutrons port. This implementation is the same as what Open vSwitch is doing when ingress_policing_rate and ingress_policing_burst are set for port.

For ingress bandwidth limit rule:

- set_tbf_bw_limit
- update_tbf_bw_limit
- delete_tbf_bw_limit

The ingress bandwidth limit is configured on the tap port by setting a simple tc-tbf queueing discipline (qdisc) on the port. It requires a value of HZ parameter configured in kernel on the host. This value is necessary to calculate the minimal burst value which is set in tc. Details about how it is calculated can be found in here. This solution is similar to Open vSwitch implementation.

The Linux bridge DSCP marking implementation relies on the linuxbridge_extension_api to request access to the IptablesManager class and to manage chains in the `mangle` table in iptables.

QoS driver design

QoS framework is flexible enough to support any third-party vendor. To integrate a third party driver (that just wants to be aware of the QoS create/update/delete API calls), one needs to implement neutron.services.qos.drivers.base, and register the driver during the core plugin or mechanism driver load, see

neutron.services.qos.drivers.openvswitch.driver register method for an example.

Note: All the functionality MUST be implemented by the vendor, neutrons QoS framework will just act as an interface to bypass the received QoS API request and help with database persistence for the API operations.

Note: L3 agent `fip_qos` extension does not have a driver implementation, it directly uses the `l3_tc_lib` for all types of routers.

Configuration

To enable the service, the following steps should be followed:

On server side:

- enable qos service in service_plugins;
- for ml2, add qos to extension_drivers in [ml2] section;
- for L3 floating IP QoS, add qos and router to service_plugins.

On agent side (OVS):

- add qos to extensions in [agent] section.

On L3 agent side:

- For for floating IPs QoS support, add fip_qos to extensions in [agent] section.

Testing strategy

All the code added or extended as part of the effort got reasonable unit test coverage.

Neutron objects

Base unit test classes to validate neutron objects were implemented in a way that allows code reuse when introducing a new object type.

There are two test classes that are utilized for that:

- BaseObjectIfaceTestCase: class to validate basic object operations (mostly CRUD) with database layer isolated.

- BaseDbObjectTestCase: class to validate the same operations with models in place and database layer unmocked.

Every new object implemented on top of one of those classes is expected to either inherit existing test cases as is, or reimplement it, if it makes sense in terms of how those objects are implemented. Specific test classes can obviously extend the set of test cases as they see needed (f.e. you need to define new test cases for those additional methods that you may add to your object implementations on top of base semantics common to all neutron objects).

Functional tests

Additions to ovs_lib to set bandwidth limits on ports are covered in:

- neutron.tests.functional.agent.test_ovs_lib

New functional tests for tc_lib to set bandwidth limits on ports are in:

- neutron.tests.functional.agent.linux.test_tc_lib

New functional tests for test_l3_tc_lib to set TC filters on router floating IP related device are covered in:

- neutron.tests.functional.agent.linux.test_l3_tc_lib

New functional tests for L3 agent floating IP rate limit:

- neutron.tests.functional.agent.l3.extensions.test_fip_qos_extension

API tests

API tests for basic CRUD operations for ports, networks, policies, and rules were added in:

- neutron-tempest-plugin.api.test_qos

Quota Management and Enforcement

Most resources exposed by the Neutron API are subject to quota limits. The Neutron API exposes an extension for managing such quotas. Quota limits are enforced at the API layer, before the request is dispatched to the plugin.

Default values for quota limits are specified in neutron.conf. Admin users can override those defaults values on a per-project basis. Limits are stored in the Neutron database; if no limit is found for a given resource and project, then the default value for such resource is used. Configuration-based quota management, where every project gets the same quota limit specified in the configuration file, has been deprecated as of the Liberty release.

Please note that Neutron does not support both specification of quota limits per user and quota management for hierarchical multitenancy (as a matter of fact Neutron does not support hierarchical multitenancy at all). Also, quota limits are currently not enforced on RPC interfaces listening on the AMQP bus.

Plugin and ML2 drivers are not supposed to enforce quotas for resources they manage. However, the subnet_allocation[1] extension is an exception and will be discussed below.

[1] Subnet allocation extension: http://opendev.org/openstack/neutron/tree/neutron/extensions/subnetallocation.py

The quota management and enforcement mechanisms discussed here apply to every resource which has been registered with the Quota engine, regardless of whether such resource belongs to the core Neutron API or one of its extensions.

High Level View

There are two main components in the Neutron quota system:

- The Quota API extensions.
- The Quota Engine.

Both components rely on a quota driver. The neutron codebase currently defines two quota drivers:

- neutron.db.quota.driver.DbQuotaDriver
- neutron.quota.ConfDriver

The latter driver is however deprecated.

The Quota API extension handles quota management, whereas the Quota Engine component handles quota enforcement. This API extension is loaded like any other extension. For this reason plugins must explicitly support it by including quotas in the supported_extension_aliases attribute.

In the Quota API simple CRUD operations are used for managing project quotas. Please note that the current behaviour when deleting a project quota is to reset quota limits for that project to configuration defaults. The API extension does not validate the project identifier with the identity service.

In addition, the Quota Detail API extension complements the Quota API extension by allowing users (typically admins) the ability to retrieve details about quotas per project. Quota details include the used/limit/reserved count for the projects resources (networks, ports, etc.).

Performing quota enforcement is the responsibility of the Quota Engine. RESTful API controllers, before sending a request to the plugin, try to obtain a reservation from the quota engine for the resources specified in the client request. If the reservation is successful, then it proceeds to dispatch the operation to the plugin.

For a reservation to be successful, the total amount of resources requested, plus the total amount of resources reserved, plus the total amount of resources already stored in the database should not exceed the projects quota limit.

Finally, both quota management and enforcement rely on a quota driver[2], whose task is basically to perform database operations.

Quota Management

The quota management component is fairly straightforward.

However, unlike the vast majority of Neutron extensions, it uses it own controller class[3]. This class does not implement the POST operation. List, get, update, and delete operations are implemented by the usual index, show, update and delete methods. These method simply call into the quota driver for either fetching project quotas or updating them.

[2] DB Quota driver class: http://opendev.org/openstack/neutron/tree/neutron/db/quota/driver.py#n30
[3] Quota API extension controller: http://opendev.org/openstack/neutron/tree/neutron/extensions/quotasv2.py#n40

The _update_attributes method is called only once in the controller lifetime. This method dynamically updates Neutrons resource attribute map[4] so that an attribute is added for every resource managed by the quota engine. Request authorisation is performed in this controller, and only admin users are allowed to modify quotas for projects. As the neutron policy engine is not used, it is not possible to configure which users should be allowed to manage quotas using policy.json.

The driver operations dealing with quota management are:

- delete_tenant_quota, which simply removes all entries from the quotas table for a given project identifier;
- update_quota_limit, which adds or updates an entry in the quotas project for a given project identifier and a given resource name;
- _get_quotas, which fetches limits for a set of resource and a given project identifier
- _get_all_quotas, which behaves like _get_quotas, but for all projects.

Resource Usage Info

Neutron has two ways of tracking resource usage info:

- CountableResource, where resource usage is calculated every time quotas limits are enforced by counting rows in the resource table and reservations for that resource.
- TrackedResource, which instead relies on a specific table tracking usage data, and performs explicitly counting only when the data in this table are not in sync with actual used and reserved resources.

Another difference between CountableResource and TrackedResource is that the former invokes a plugin method to count resources. CountableResource should be therefore employed for plugins which do not leverage the Neutron database. The actual class that the Neutron quota engine will use is determined by the track_quota_usage variable in the quota configuration section. If True, TrackedResource instances will be created, otherwise the quota engine will use CountableResource instances. Resource creation is performed by the create_resource_instance factory method in the neutron.quota.resource module.

From a performance perspective, having a table tracking resource usage has some advantages, albeit not fundamental. Indeed the time required for executing queries to explicitly count objects will increase with the number of records in the table. On the other hand, using TrackedResource will fetch a single record, but has the drawback of having to execute an UPDATE statement once the operation is completed. Nevertheless, CountableResource instances do not simply perform a SELECT query on the relevant table for a resource, but invoke a plugin method, which might execute several statements and sometimes even interacts with the backend before returning. Resource usage tracking also becomes important for operational correctness when coupled with the concept of resource reservation, discussed in another section of this chapter.

Tracking quota usage is not as simple as updating a counter every time resources are created or deleted. Indeed a quota-limited resource in Neutron can be created in several ways. While a RESTful API request is the most common one, resources can be created by RPC handlers listing on the AMQP bus, such as those which create DHCP ports, or by plugin operations, such as those which create router ports.

To this aim, TrackedResource instances are initialised with a reference to the model class for the resource for which they track usage data. During object initialisation, SqlAlchemy event handlers are installed for this class. The event handler is executed after a record is inserted or deleted. As result usage data for

[4] Neutron resource attribute map: http://opendev.org/openstack/neutron/tree/neutron/api/v2/attributes.py#n639

that resource and will be marked as dirty once the operation completes, so that the next time usage data is requested, it will be synchronised counting resource usage from the database. Even if this solution has some drawbacks, listed in the exceptions and caveats section, it is more reliable than solutions such as:

- Updating the usage counters with the new correct value every time an operation completes.
- Having a periodic task synchronising quota usage data with actual data in the Neutron DB.

Finally, regardless of whether CountableResource or TrackedResource is used, the quota engine always invokes its count() method to retrieve resource usage. Therefore, from the perspective of the Quota engine there is absolutely no difference between CountableResource and TrackedResource.

Quota Enforcement

Before dispatching a request to the plugin, the Neutron base controller[5] attempts to make a reservation for requested resource(s). Reservations are made by calling the make_reservation method in neutron.quota.QuotaEngine. The process of making a reservation is fairly straightforward:

- Get current resource usages. This is achieved by invoking the count method on every requested resource, and then retrieving the amount of reserved resources.
- Fetch current quota limits for requested resources, by invoking the _get_tenant_quotas method.
- Fetch expired reservations for selected resources. This amount will be subtracted from resource usage. As in most cases there wont be any expired reservation, this approach actually requires less DB operations than doing a sum of non-expired, reserved resources for each request.
- For each resource calculate its headroom, and verify the requested amount of resource is less than the headroom.
- If the above is true for all resource, the reservation is saved in the DB, otherwise an OverQuotaLimit exception is raised.

The quota engine is able to make a reservation for multiple resources. However, it is worth noting that because of the current structure of the Neutron API layer, there will not be any practical case in which a reservation for multiple resources is made. For this reason performance optimisation avoiding repeating queries for every resource are not part of the current implementation.

In order to ensure correct operations, a row-level lock is acquired in the transaction which creates the reservation. The lock is acquired when reading usage data. In case of write-set certification failures, which can occur in active/active clusters such as MySQL galera, the decorator neutron_lib.db.api.retry_db_errors will retry the transaction if a DBDeadLock exception is raised. While non-locking approaches are possible, it has been found out that, since a non-locking algorithms increases the chances of collision, the cost of handling a DBDeadlock is still lower than the cost of retrying the operation when a collision is detected. A study in this direction was conducted for IP allocation operations, but the same principles apply here as well[6]. Nevertheless, moving away for DB-level locks is something that must happen for quota enforcement in the future.

Committing and cancelling a reservation is as simple as deleting the reservation itself. When a reservation is committed, the resources which were committed are now stored in the database, so the reservation itself should be deleted. The Neutron quota engine simply removes the record when cancelling a reservation (ie: the request failed to complete), and also marks quota usage info as dirty when the reservation

[5] Base controller class: http://opendev.org/openstack/neutron/tree/neutron/api/v2/base.py#n50
[6] http://lists.openstack.org/pipermail/openstack-dev/2015-February/057534.html

is committed (ie: the request completed correctly). Reservations are committed or cancelled by respectively calling the commit_reservation and cancel_reservation methods in neutron.quota.QuotaEngine.

Reservations are not perennial. Eternal reservation would eventually exhaust projects quotas because they would never be removed when an API worker crashes whilst in the middle of an operation. Reservation expiration is currently set to 120 seconds, and is not configurable, not yet at least. Expired reservations are not counted when calculating resource usage. While creating a reservation, if any expired reservation is found, all expired reservation for that project and resource will be removed from the database, thus avoiding build-up of expired reservations.

Setting up Resource Tracking for a Plugin

By default plugins do not leverage resource tracking. Having the plugin explicitly declare which resources should be tracked is a precise design choice aimed at limiting as much as possible the chance of introducing errors in existing plugins.

For this reason a plugin must declare which resource it intends to track. This can be achieved using the tracked_resources decorator available in the neutron.quota.resource_registry module. The decorator should ideally be applied to the plugins __init__ method.

The decorator accepts in input a list of keyword arguments. The name of the argument must be a resource name, and the value of the argument must be a DB model class. For example:

::

@resource_registry.tracked_resources(network=models_v2.Network, port=models_v2.Port,
subnet=models_v2.Subnet, subnetpool=models_v2.SubnetPool)

Will ensure network, port, subnet and subnetpool resources are tracked. In theory, it is possible to use this decorator multiple times, and not exclusively to __init__ methods. However, this would eventually lead to code readability and maintainability problems, so developers are strongly encourage to apply this decorator exclusively to the plugins __init__ method (or any other method which is called by the plugin only once during its initialization).

Notes for Implementors of RPC Interfaces and RESTful Controllers

Neutron unfortunately does not have a layer which is called before dispatching the operation from the plugin which can be leveraged both from RESTful and RPC over AMQP APIs. In particular the RPC handlers call straight into the plugin, without doing any request authorisation or quota enforcement.

Therefore RPC handlers must explicitly indicate if they are going to call the plugin to create or delete any sort of resources. This is achieved in a simple way, by ensuring modified resources are marked as dirty after the RPC handler execution terminates. To this aim developers can use the mark_resources_dirty decorator available in the module neutron.quota.resource_registry.

The decorator would scan the whole list of registered resources, and store the dirty status for their usage trackers in the database for those resources for which items have been created or destroyed during the plugin operation.

Exceptions and Caveats

Please be aware of the following limitations of the quota enforcement engine:

- Subnet allocation from subnet pools, in particularly shared pools, is also subject to quota limit checks. However this checks are not enforced by the quota engine, but trough a mechanism implemented in the neutron.ipam.subnetalloc module. This is because the Quota engine is not able to satisfy the requirements for quotas on subnet allocation.
- The quota engine also provides a limit_check routine which enforces quota checks without creating reservations. This way of doing quota enforcement is extremely unreliable and superseded by the reservation mechanism. It has not been removed to ensure off-tree plugins and extensions which leverage are not broken.
- SqlAlchemy events might not be the most reliable way for detecting changes in resource usage. Since the event mechanism monitors the data model class, it is paramount for a correct quota enforcement, that resources are always created and deleted using object relational mappings. For instance, deleting a resource with a query.delete call, will not trigger the event. SQLAlchemy events should be considered as a temporary measure adopted as Neutron lacks persistent API objects.
- As CountableResource instance do not track usage data, when making a reservation no write-intent lock is acquired. Therefore the quota engine with CountableResource is not concurrency-safe.
- The mechanism for specifying for which resources enable usage tracking relies on the fact that the plugin is loaded before quota-limited resources are registered. For this reason it is not possible to validate whether a resource actually exists or not when enabling tracking for it. Developers should pay particular attention into ensuring resource names are correctly specified.
- The code assumes usage trackers are a trusted source of truth: if they report a usage counter and the dirty bit is not set, that counter is correct. If its dirty than surely that counter is out of sync. This is not very robust, as there might be issues upon restart when toggling the use_tracked_resources configuration variable, as stale counters might be trusted upon for making reservations. Also, the same situation might occur if a server crashes after the API operation is completed but before the reservation is committed, as the actual resource usage is changed but the corresponding usage tracker is not marked as dirty.

References

Retrying Operations

Inside of the neutron_lib.db.api module there is a decorator called retry_if_session_inactive. This should be used to protect any functions that perform DB operations. This decorator will capture any deadlock errors, RetryRequests, connection errors, and unique constraint violations that are thrown by the function it is protecting.

This decorator will not retry an operation if the function it is applied to is called within an active session. This is because the majority of the exceptions it captures put the session into a partially rolled back state so it is no longer usable. It is important to ensure there is a decorator outside of the start of the transaction. The decorators are safe to nest if a function is sometimes called inside of another transaction.

If a function is being protected that does not take context as an argument the retry_db_errors decorator function may be used instead. It retries the same exceptions and has the same anti-nesting behavior as retry_if_session_active, but it does not check if a session is attached to any context keywords. (retry_if_session_active just uses retry_db_errors internally after checking the session)

Idempotency on Failures

The function that is being decorated should always fully cleanup whenever it encounters an exception so its safe to retry the operation. So if a function creates a DB object, commits, then creates another, the function must have a cleanup handler to remove the first DB object in the case that the second one fails. Assume any DB operation can throw a retriable error.

You may see some retry decorators at the API layers in Neutron; however, we are trying to eliminate them because each API operation has many independent steps that makes ensuring idempotency on partial failures very difficult.

Argument Mutation

A decorated function should not mutate any complex arguments which are passed into it. If it does, it should have an exception handler that reverts the change so its safe to retry.

The decorator will automatically create deep copies of sets, lists, and dicts which are passed through it, but it will leave the other arguments alone.

Retrying to Handle Race Conditions

One of the difficulties with detecting race conditions to create a DB record with a unique constraint is determining where to put the exception handler because a constraint violation can happen immediately on flush or it may not happen all of the way until the transaction is being committed on the exit of the session context manager. So we would end up with code that looks something like this:

```
def create_port(context, ip_address, mac_address):
    _ensure_mac_not_in_use(context, mac_address)
    _ensure_ip_not_in_use(context, ip_address)
    try:
        with context.session.begin():
           port_obj = Port(ip=ip_address, mac=mac_address)
           do_expensive_thing(...)
           do_extra_other_thing(...)
           return port_obj
    except DBDuplicateEntry as e:
        # code to parse columns
        if 'mac' in e.columns:
            raise MacInUse(mac_address)
        if 'ip' in e.columns:
            raise IPAddressInUse(ip)

def _ensure_mac_not_in_use(context, mac):
    if context.session.query(Port).filter_by(mac=mac).count():
        raise MacInUse(mac)

def _ensure_ip_not_in_use(context, ip):
    if context.session.query(Port).filter_by(ip=ip).count():
        raise IPAddressInUse(ip)
```

So we end up with an exception handler that has to understand where things went wrong and convert them into appropriate exceptions for the end-users. This distracts significantly from the main purpose of create_port.

Since the retry decorator will automatically catch and retry DB duplicate errors for us, we can allow it to retry on this race condition which will give the original validation logic to be re-executed and raise the appropriate error. This keeps validation logic in one place and makes the code cleaner.

```
from neutron.db import api as db_api

@db_api.retry_if_session_inactive()
def create_port(context, ip_address, mac_address):
    _ensure_mac_not_in_use(context, mac_address)
    _ensure_ip_not_in_use(context, ip_address)
    with context.session.begin():
       port_obj = Port(ip=ip_address, mac=mac_address)
       do_expensive_thing(...)
       do_extra_other_thing(...)
       return port_obj

def _ensure_mac_not_in_use(context, mac):
    if context.session.query(Port).filter_by(mac=mac).count():
        raise MacInUse(mac)

def _ensure_ip_not_in_use(context, ip):
    if context.session.query(Port).filter_by(ip=ip).count():
        raise IPAddressInUse(ip)
```

Nesting

Once the decorator retries an operation the maximum number of times, it will attach a flag to the exception it raises further up that will prevent decorators around the calling functions from retrying the error again. This prevents an exponential increase in the number of retries if they are layered.

Usage

Here are some usage examples:

```
from neutron.db import api as db_api

@db_api.retry_if_session_inactive()
def create_elephant(context, elephant_details):
    ....

@db_api.retry_if_session_inactive()
def atomic_bulk_create_elephants(context, elephants):
    with context.session.begin():
        for elephant in elephants:
            # note that if create_elephant throws a retriable
            # exception, the decorator around it will not retry
            # because the session is active. The decorator around
            # atomic_bulk_create_elephants will be responsible for
            # retrying the entire operation.
```

(continues on next page)

(continued from previous page)

```
            create_elephant(context, elephant)

# sample usage when session is attached to a var other than 'context'
@db_api.retry_if_session_inactive(context_var_name='ctx')
def some_function(ctx):
    ...
```

Neutron RPC API Layer

Neutron uses the oslo.messaging library to provide an internal communication channel between Neutron services. This communication is typically done via AMQP, but those details are mostly hidden by the use of oslo.messaging and it could be some other protocol in the future.

RPC APIs are defined in Neutron in two parts: client side and server side.

Client Side

Here is an example of an rpc client definition:

```
import oslo_messaging

from neutron.common import rpc as n_rpc


class ClientAPI(object):
    """Client side RPC interface definition.

    API version history:
        1.0 - Initial version
        1.1 - Added my_remote_method_2
    """

    def __init__(self, topic):
        target = oslo_messaging.Target(topic=topic, version='1.0')
        self.client = n_rpc.get_client(target)

    def my_remote_method(self, context, arg1, arg2):
        cctxt = self.client.prepare()
        return cctxt.call(context, 'my_remote_method', arg1=arg1,␣
↪arg2=arg2)

    def my_remote_method_2(self, context, arg1):
        cctxt = self.client.prepare(version='1.1')
        return cctxt.call(context, 'my_remote_method_2', arg1=arg1)
```

This class defines the client side interface for an rpc API. The interface has 2 methods. The first method existed in version 1.0 of the interface. The second method was added in version 1.1. When the newer method is called, it specifies that the remote side must implement at least version 1.1 to handle this request.

Server Side

The server side of an rpc interface looks like this:

```
import oslo_messaging


class ServerAPI(object):

    target = oslo_messaging.Target(version='1.1')

    def my_remote_method(self, context, arg1, arg2):
        return 'foo'

    def my_remote_method_2(self, context, arg1):
        return 'bar'
```

This class implements the server side of the interface. The oslo_messaging.Target() defined says that this class currently implements version 1.1 of the interface.

Versioning

Note that changes to rpc interfaces must always be done in a backwards compatible way. The server side should always be able to handle older clients (within the same major version series, such as 1.X).

It is possible to bump the major version number and drop some code only needed for backwards compatibility. For more information about how to do that, see https://wiki.openstack.org/wiki/RpcMajorVersionUpdates.

Example Change

As an example minor API change, lets assume we want to add a new parameter to my_remote_method_2. First, we add the argument on the server side. To be backwards compatible, the new argument must have a default value set so that the interface will still work even if the argument is not supplied. Also, the interfaces minor version number must be incremented. So, the new server side code would look like this:

```
import oslo_messaging


class ServerAPI(object):

    target = oslo_messaging.Target(version='1.2')

    def my_remote_method(self, context, arg1, arg2):
        return 'foo'

    def my_remote_method_2(self, context, arg1, arg2=None):
        if not arg2:
            # Deal with the fact that arg2 was not specified if needed.
        return 'bar'
```

We can now update the client side to pass the new argument. The client must also specify that version 1.2 is required for this method call to be successful. The updated client side would look like this:

```
import oslo_messaging

from neutron.common import rpc as n_rpc


class ClientAPI(object):
    """Client side RPC interface definition.

    API version history:
        1.0 - Initial version
        1.1 - Added my_remote_method_2
        1.2 - Added arg2 to my_remote_method_2
    """

    def __init__(self, topic):
        target = oslo_messaging.Target(topic=topic, version='1.0')
        self.client = n_rpc.get_client(target)

    def my_remote_method(self, context, arg1, arg2):
        cctxt = self.client.prepare()
        return cctxt.call(context, 'my_remote_method', arg1=arg1, 
↪arg2=arg2)

    def my_remote_method_2(self, context, arg1, arg2):
        cctxt = self.client.prepare(version='1.2')
        return cctxt.call(context, 'my_remote_method_2',
                          arg1=arg1, arg2=arg2)
```

Neutron RPC APIs

As discussed before, RPC APIs are defined in two parts: a client side and a server side. Several of these pairs exist in the Neutron code base. The code base is being updated with documentation on every rpc interface implementation that indicates where the corresponding server or client code is located.

Example: DHCP

The DHCP agent includes a client API, neutron.agent.dhcp.agent.DhcpPluginAPI. The DHCP agent uses this class to call remote methods back in the Neutron server. The server side is defined in neutron.api.rpc.handlers.dhcp_rpc.DhcpRpcCallback. It is up to the Neutron plugin in use to decide whether the DhcpRpcCallback interface should be exposed.

Similarly, there is an RPC interface defined that allows the Neutron plugin to remotely invoke methods in the DHCP agent. The client side is defined in neutron.api.rpc.agentnotifiers.dhcp_rpc_agent_api.DhcpAgentNotifyAPI. The server side of this interface that runs in the DHCP agent is neutron.agent.dhcp.agent.DhcpAgent.

More Info

For more information, see the oslo.messaging documentation: https://docs.openstack.org/oslo.messaging/latest/.

Neutron Messaging Callback System

Neutron already has a callback system for in-process resource callbacks where publishers and subscribers are able to publish and subscribe for resource events.

This system is different, and is intended to be used for inter-process callbacks, via the messaging fanout mechanisms.

In Neutron, agents may need to subscribe to specific resource details which may change over time. And the purpose of this messaging callback system is to allow agent subscription to those resources without the need to extend modify existing RPC calls, or creating new RPC messages.

A few resource which can benefit of this system:

- QoS policies;
- Security Groups.

Using a remote publisher/subscriber pattern, the information about such resources could be published using fanout messages to all interested nodes, minimizing messaging requests from agents to server since the agents get subscribed for their whole lifecycle (unless they unsubscribe).

Within an agent, there could be multiple subscriber callbacks to the same resource events, the resources updates would be dispatched to the subscriber callbacks from a single message. Any update would come in a single message, doing only a single oslo versioned objects deserialization on each receiving agent.

This publishing/subscription mechanism is highly dependent on the format of the resources passed around. This is why the library only allows versioned objects to be published and subscribed. Oslo versioned objects allow object version down/up conversion.[2][3]

For the VOs versioning schema look here:[4]

versioned_objects serialization/deserialization with the obj_to_primitive(target_version=..) and primitive_to_obj()[1] methods is used internally to convert/retrieve objects before/after messaging.

Serialized versioned objects look like:

```
{'versioned_object.version': '1.0',
 'versioned_object.name': 'QoSPolicy',
 'versioned_object.data': {'rules': [
                                    {'versioned_object.version': '1.0',
                                     'versioned_object.name':
↪'QoSBandwidthLimitRule',
                                     'versioned_object.data': {'name': u'a
↪'},
```

(continues on next page)

[2] https://github.com/openstack/oslo.versionedobjects/blob/ce00f18f7e9143b5175e889970564813189e3e6d/oslo_versionedobjects/base.py#L474

[3] https://github.com/openstack/oslo.versionedobjects/blob/ce00f18f7e9143b5175e889970564813189e3e6d/oslo_versionedobjects/tests/test_objects.py#L114

[4] https://github.com/openstack/oslo.versionedobjects/blob/ce00f18f7e9143b5175e889970564813189e3e6d/oslo_versionedobjects/base.py#L248

[1] https://github.com/openstack/oslo.versionedobjects/blob/ce00f18f7e9143b5175e889970564813189e3e6d/oslo_versionedobjects/tests/test_objects.py#L410

(continued from previous page)

```
                                               'versioned_object.namespace':
↪'versionedobjects'}
                                              ],
                                 'uuid': u'abcde',
                                 'name': u'aaa'},
 'versioned_object.namespace': 'versionedobjects'}
```

Rolling upgrades strategy

In this section we assume the standard Neutron upgrade process, which means upgrade the server first and then upgrade the agents:

More information about the upgrade strategy.

We provide an automatic method which avoids manual pinning and unpinning of versions by the administrator which could be prone to error.

Resource pull requests

Resource pull requests will always be ok because the underlying resource RPC does provide the version of the requested resource id / ids. The server will be upgraded first, so it will always be able to satisfy any version the agents request.

Resource push notifications

Agents will subscribe to the neutron-vo-<resource_type>-<version> fanout queue which carries updated objects for the version they know about. The versions they know about depend on the runtime Neutron versioned objects they started with.

When the server upgrades, it should be able to instantly calculate a census of agent versions per object (we will define a mechanism for this in a later section). It will use the census to send fanout messages on all the version span a resource type has.

For example, if neutron-server knew it has rpc-callback aware agents with versions 1.0, and versions 1.2 of resource type A, any update would be sent to neutron-vo-A_1.0 and neutron-vo-A_1.2.

TODO(mangelajo): Verify that after upgrade is finished any unused messaging resources (queues, exchanges, and so on) are released as older agents go away and neutron-server stops producing new message casts. Otherwise document the need for a neutron-server restart after rolling upgrade has finished if we want the queues cleaned up.

Leveraging agent state reports for object version discovery

We add a row to the agent db for tracking agent known objects and version numbers. This resembles the implementation of the configuration column.

Agents report at start not only their configuration now, but also their subscribed object type / version pairs, that are stored in the database and made available to any neutron-server requesting it:

```
'resource_versions': {'QosPolicy': '1.1',
                      'SecurityGroup': '1.0',
                      'Port': '1.0'}
```

There was a subset of Liberty agents depending on QosPolicy that required QosPolicy: 1.0 if the qos plugin is installed. We were able to identify those by the binary name (included in the report):

- neutron-openvswitch-agent
- neutron-sriov-nic-agent

This transition was handled in the Mitaka version, but its not handled anymore in Newton, since only one major version step upgrades are supported.

Version discovery

With the above mechanism in place and considering the exception of neutron-openvswitch-agent and neutron-sriov-agent requiring QoSpolicy 1.0, we discover the subset of versions to be sent on every push notification.

Agents that are in down state are excluded from this calculation. We use an extended timeout for agents in this calculation to make sure were on the safe side, specially if deployer marked agents with low timeouts.

Starting at Mitaka, any agent interested in versioned objects via this API should report their resource/version tuples of interest (the resource type/ version pairs theyre subscribed to).

The plugins interested in this RPC mechanism must inherit AgentDbMixin, since this mechanism is only intended to be used from agents at the moment, while it could be extended to be consumed from other components if necessary.

The AgentDbMixin provides:

```
def get_agents_resource_versions(self, tracker):
    ...
```

Caching mechanism

The version subset per object is cached to avoid DB requests on every push given that we assume that all old agents are already registered at the time of upgrade.

Cached subset is re-evaluated (to cut down the version sets as agents upgrade) after neutron.api.rpc.callbacks.version_manager.VERSIONS_TTL.

As a fast path to update this cache on all neutron-servers when upgraded agents come up (or old agents revive after a long timeout or even a downgrade) the server registering the new status update notifies the other servers about the new consumer resource versions via cast.

All notifications for all calculated version sets must be sent, as non-upgraded agents would otherwise not receive them.

It is safe to send notifications to any fanout queue as they will be discarded if no agent is listening.

Topic names for every resource type RPC endpoint

neutron-vo-<resource_class_name>-<version>

In the future, we may want to get oslo messaging to support subscribing topics dynamically, then we may want to use:

neutron-vo-<resource_class_name>-<resource_id>-<version> instead,

or something equivalent which would allow fine granularity for the receivers to only get interesting information to them.

Subscribing to resources

Imagine that you have agent A, which just got to handle a new port, which has an associated security group, and QoS policy.

The agent code processing port updates may look like:

```
from neutron.api.rpc.callbacks.consumer import registry
from neutron.api.rpc.callbacks import events
from neutron.api.rpc.callbacks import resources


def process_resource_updates(context, resource_type, resource_list, event_
↪type):

    # send to the right handler which will update any control plane
    # details related to the updated resources...


def subscribe_resources():
    registry.register(process_resource_updates, resources.SEC_GROUP)

    registry.register(process_resource_updates, resources.QOS_POLICY)

def port_update(port):

    # here we extract sg_id and qos_policy_id from port..

    sec_group = registry.pull(resources.SEC_GROUP, sg_id)
    qos_policy = registry.pull(resources.QOS_POLICY, qos_policy_id)
```

The relevant function is:

- register(callback, resource_type): subscribes callback to a resource type.

The callback function will receive the following arguments:

- context: the neutron context that triggered the notification.
- resource_type: the type of resource which is receiving the update.

- resource_list: list of resources which have been pushed by server.
- event_type: will be one of CREATED, UPDATED, or DELETED, see neutron.api.rpc.callbacks.events for details.

With the underlying oslo_messaging support for dynamic topics on the receiver we cannot implement a per resource type + resource id topic, rabbitmq seems to handle 10000s of topics without suffering, but creating 100s of oslo_messaging receivers on different topics seems to crash.

We may want to look into that later, to avoid agents receiving resource updates which are uninteresting to them.

Unsubscribing from resources

To unsubscribe registered callbacks:

- unsubscribe(callback, resource_type): unsubscribe from specific resource type.
- unsubscribe_all(): unsubscribe from all resources.

Sending resource events

On the server side, resource updates could come from anywhere, a service plugin, an extension, anything that updates, creates, or destroys the resources and that is of any interest to subscribed agents.

A callback is expected to receive a list of resources. When resources in the list belong to the same resource type, a single push RPC message is sent; if the list contains objects of different resource types, resources of each type are grouped and sent separately, one push RPC message per type. On the receiver side, resources in a list always belong to the same type. In other words, a server-side push of a list of heterogeneous objects will result into N messages on bus and N client-side callback invocations, where N is the number of unique resource types in the given list, e.g. L(A, A, B, C, C, C) would be fragmented into L1(A, A), L2(B), L3(C, C, C), and each list pushed separately.

Note: there is no guarantee in terms of order in which separate resource lists will be delivered to consumers.

The server/publisher side may look like:

```
from neutron.api.rpc.callbacks.producer import registry
from neutron.api.rpc.callbacks import events

def create_qos_policy(...):
    policy = fetch_policy(...)
    update_the_db(...)
    registry.push([policy], events.CREATED)

def update_qos_policy(...):
    policy = fetch_policy(...)
    update_the_db(...)
    registry.push([policy], events.UPDATED)

def delete_qos_policy(...):
    policy = fetch_policy(...)
    update_the_db(...)
    registry.push([policy], events.DELETED)
```

References

Segments extension

Neutron has an extension that allows CRUD operations on the `/segments` resource in the API, that corresponds to the `NetworkSegment` entity in the DB layer. The extension is implemented as a service plug-in.

Note: The `segments` service plug-in is not configured by default. To configure it, add `segments` to the `service_plugins` parameter in `neutron.conf`

Core plug-ins can coordinate with the `segments` service plug-in by subscribing callbacks to events associated to the `SEGMENT` resource. Currently, the segments plug-in notifies subscribers of the following events:

- `PRECOMMIT_CREATE`
- `AFTER_CREATE`
- `BEFORE_DELETE`
- `PRECOMMIT_DELETE`
- `AFTER_DELETE`

As of this writing, `ML2` and `OVN` register callbacks to receive events from the `segments` service plug-in. The `ML2` plug-in defines the callback `_handle_segment_change` to process all the relevant segments events.

Segments extension relevant modules

- `neutron/extensions/segment.py` defines the extension
- `neutron/db/models/segment.py` defines the DB models for segments and for the segment host mapping, that is used in the implementation of routed networks.
- `neutron/db/segments_db.py` has functions to add, retrieve and delete segments from the DB.
- `neutron/services/segments/db.py` defines a mixin class with the methods that perform API CRUD operations for the `segments` plug-in. It also has a set of functions to create and maintain the mapping of segments to hosts, which is necessary in the implementation of routed networks.
- `neutron/services/segments/plugin.py` defines the `segments` service plug-in.

Service Extensions

Historically, Neutron supported the following advanced services:

1. **FWaaS** (*Firewall-as-a-Service*): runs as part of the L3 agent.
2. **VPNaaS** (*VPN-as-a-Service*): derives from L3 agent to add VPNaaS functionality.

Starting with the Kilo release, these services are split into separate repositories, and more extensions are being developed as well. Service plugins are a clean way of adding functionality in a cohesive manner and yet, keeping them decoupled from the guts of the framework. The aforementioned features are developed as extensions (also known as service plugins), and more capabilities are being added to Neutron following the same pattern. For those that are deemed orthogonal to any network service (e.g. tags, timestamps, auto_allocate, etc), there is an informal mechanism to have these loaded automatically at server startup. If you consider adding an entry to the dictionary, please be kind and reach out to your PTL or a member of the drivers team for approval.

1. http://opendev.org/openstack/neutron-fwaas/
2. http://opendev.org/openstack/neutron-vpnaas/

Calling the Core Plugin from Services

There are many cases where a service may want to create a resource managed by the core plugin (e.g. ports, networks, subnets). This can be achieved by importing the plugins directory and getting a direct reference to the core plugin:

```
from neutron_lib.plugins import directory

plugin = directory.get_plugin()
plugin.create_port(context, port_dict)
```

However, there is an important caveat. Calls to the core plugin in almost every case should not be made inside of an ongoing transaction. This is because many plugins (including ML2), can be configured to make calls to a backend after creating or modifying an object. If the call is made inside of a transaction and the transaction is rolled back after the core plugin call, the backend will not be notified that the change was undone. This will lead to consistency errors between the core plugin and its configured backend(s).

ML2 has a guard against certain methods being called with an active DB transaction to help prevent developers from accidentally making this mistake. It will raise an error that says explicitly that the method should not be called within a transaction.

Services and agents

A usual Neutron setup consists of multiple services and agents running on one or multiple nodes (though some exotic setups potentially may not need any agents). Each of those services provides some of the networking or API services. Among those of special interest:

1. neutron-server that provides API endpoints and serves as a single point of access to the database. It usually runs on nodes called Controllers.
2. Layer2 agent that can utilize Open vSwitch, Linuxbridge or other vendor specific technology to provide network segmentation and isolation for project networks. The L2 agent should run on

every node where it is deemed responsible for wiring and securing virtual interfaces (usually both Compute and Network nodes).

3. Layer3 agent that runs on Network node and provides East-West and North-South routing plus some advanced services such as FWaaS or VPNaaS.

For the purpose of this document, we call all services, servers and agents that run on any node as just services.

Entry points

Entry points for services are defined in setup.cfg under console_scripts section. Those entry points should generally point to main() functions located under neutron/cmd/ path.

Note: some existing vendor/plugin agents still maintain their entry points in other locations. Developers responsible for those agents are welcome to apply the guideline above.

Interacting with Eventlet

Neutron extensively utilizes the eventlet library to provide asynchronous concurrency model to its services. To utilize it correctly, the following should be kept in mind.

If a service utilizes the eventlet library, then it should not call eventlet.monkey_patch() directly but instead maintain its entry point main() function under neutron/cmd/eventlet/ If that is the case, the standard Python library will be automatically patched for the service on entry point import (monkey patching is done inside python package file).

Note: an entry point main() function may just be an indirection to a real callable located elsewhere, as is done for reference services such as DHCP, L3 and the neutron-server.

For more info on the rationale behind the code tree setup, see the corresponding cross-project spec.

Connecting to the Database

Only the neutron-server connects to the neutron database. Agents may never connect directly to the database, as this would break the ability to do rolling upgrades.

Configuration Options

In addition to database access, configuration options are segregated between neutron-server and agents. Both services and agents may load the main `` `neutron.conf` `` since this file should contain the oslo.messaging configuration for internal Neutron RPCs and may contain host specific configuration such as file paths. In addition `` `neutron.conf` `` contains the database, Keystone, and Nova credentials and endpoints strictly for neutron-server to use.

In addition neutron-server may load a plugin specific configuration file, yet the agents should not. As the plugin configuration is primarily site wide options and the plugin provides the persistence layer for Neutron, agents should be instructed to act upon these values via RPC.

Each individual agent may have its own configuration file. This file should be loaded after the main `` `neutron.conf` `` file, so the agent configuration takes precedence. The agent specific configuration may contain configurations which vary between hosts in a Neutron deployment such as the `local_ip` for an L2 agent. If any agent requires access to additional external services beyond the neutron RPC, those endpoints should be defined in the agent-specific configuration file (e.g. nova metadata for metadata agent).

Add Tags to Neutron Resources

Tag service plugin allows users to set tags on their resources. Tagging resources can be used by external systems or any other clients of the Neutron REST API (and NOT backend drivers).

The following use cases refer to adding tags to networks, but the same can be applicable to any other Neutron resource:

1) Ability to map different networks in different OpenStack locations to one logically same network (for Multi site OpenStack)
2) Ability to map Ids from different management/orchestration systems to OpenStack networks in mixed environments, for example for project Kuryr, map docker network id to neutron network id
3) Leverage tags by deployment tools
4) allow operators to tag information about provider networks (e.g. high-bandwidth, low-latency, etc)
5) new features like get-me-a-network or a similar port scheduler could choose a network for a port based on tags

Which Resources

Tag system uses standardattr mechanism so its targeting to resources that have the mechanism. Some resources with standard attribute dont suit fit tag support usecases (e.g. security_group_rule). If new tag support resource is added, the resource model should inherit HasStandardAttributes and then it must implement the property api_parent and tag_support. And also the change must include a release note for API user.

Current API resources extended by tag extensions:

- floatingips
- networks
- network_segment_ranges
- policies
- ports
- routers
- security_groups
- subnetpools
- subnets
- trunks

Model

Tag is not standalone resource. Tag is always related to existing resources. The following shows tag model:

```
+------------------+        +------------------+
|     Network      |        |       Tag        |
+------------------+        +------------------+
| standard_attr_id +------> | standard_attr_id |
|                  |        | tag              |
|                  |        |                  |
+------------------+        +------------------+
```

Tag has two columns only and tag column is just string. These tags are defined per resource. Tag is unique in a resource but it can be overlapped throughout.

API

The following shows basic API for tag. Tag is regarded as a subresource of resource so API always includes id of resource related to tag.

Add a single tag on a network

```
PUT /v2.0/networks/{network_id}/tags/{tag}
```

Returns *201 Created*. If the tag already exists, no error is raised, it just returns the *201 Created* because the OpenStack Development Mailing List discussion told us that PUT should be no issue updating an existing tag.

Replace set of tags on a network

```
PUT /v2.0/networks/{network_id}/tags
```

with request payload

```
{
    'tags': ['foo', 'bar', 'baz']
}
```

Response

```
{
    'tags': ['foo', 'bar', 'baz']
}
```

Check if a tag exists or not on a network

```
GET /v2.0/networks/{network_id}/tags/{tag}
```

Remove a single tag on a network

```
DELETE /v2.0/networks/{network_id}/tags/{tag}
```

Remove all tags on a network

```
DELETE /v2.0/networks/{network_id}/tags
```

PUT and DELETE for collections are the motivation of extending the API framework.

Note: Much of this document discusses upgrade considerations for the Neutron reference implementation using Neutrons agents. Its expected that each Neutron plugin provides its own documentation that discusses upgrade considerations specific to that choice of backend. For example, OVN does not use Neutron agents, but does have a local controller that runs on each compute node. OVN supports rolling upgrades, but information about how that works should be covered in the documentation for the OVN Neutron plugin.

Upgrade strategy

There are two general upgrade scenarios supported by Neutron:

1. All services are shut down, code upgraded, then all services are started again.
2. Services are upgraded gradually, based on operator service windows.

The latter is the preferred way to upgrade an OpenStack cloud, since it allows for more granularity and less service downtime. This scenario is usually called rolling upgrade.

Rolling upgrade

Rolling upgrades imply that during some interval of time there will be services of different code versions running and interacting in the same cloud. It puts multiple constraints onto the software.

1. older services should be able to talk with newer services.
2. older services should not require the database to have older schema (otherwise newer services that require the newer schema would not work).

More info on rolling upgrades in OpenStack.

Those requirements are achieved in Neutron by:

1. If the Neutron backend makes use of Neutron agents, the Neutron server have backwards compatibility code to deal with older messaging payloads.
2. isolating a single service that accesses database (neutron-server).

To simplify the matter, its always assumed that the order of service upgrades is as following:

1. first, all neutron-servers are upgraded.
2. then, if applicable, neutron agents are upgraded.

This approach allows us to avoid backwards compatibility code on agent side and is in line with other OpenStack projects that support rolling upgrades (specifically, nova).

Server upgrade

Neutron-server is the very first component that should be upgraded to the new code. Its also the only component that relies on new database schema to be present, other components communicate with the cloud through AMQP and hence do not depend on particular database state.

Database upgrades are implemented with alembic migration chains.

Database upgrade is split into two parts:

1. `neutron-db-manage upgrade --expand`
2. `neutron-db-manage upgrade --contract`

Each part represents a separate alembic branch.

The former step can be executed while old neutron-server code is running. The latter step requires *all* neutron-server instances to be shut down. Once its complete, neutron-servers can be started again.

Note: Full shutdown of neutron-server instances can be skipped depending on whether there are pending contract scripts not applied to the database:

```
$ neutron-db-manage has_offline_migrations
Command will return a message if there are pending contract scripts.
```

More info on alembic scripts.

Agents upgrade

Note: This section does not apply when the cloud does not use AMQP agents to provide networking services to instances. In that case, other backend specific upgrade instructions may also apply.

Once neutron-server services are restarted with the new database schema and the new code, its time to upgrade Neutron agents.

Note that in the meantime, neutron-server should be able to serve AMQP messages sent by older versions of agents which are part of the cloud.

The recommended order of agent upgrade (per node) is:

1. first, L2 agents (openvswitch, linuxbridge, sr-iov).
2. then, all other agents (L3, DHCP, Metadata,).

The rationale of the agent upgrade order is that L2 agent is usually responsible for wiring ports for other agents to use, so its better to allow it to do its job first and then proceed with other agents that will use the already configured ports for their needs.

Each network/compute node can have its own upgrade schedule that is independent of other nodes.

AMQP considerations

Since its always assumed that neutron-server component is upgraded before agents, only the former should handle both old and new RPC versions.

The implication of that is that no code that handles UnsupportedVersion oslo.messaging exceptions belongs to agent code.

Notifications

For notifications that are issued by neutron-server to listening agents, special consideration is needed to support rolling upgrades. In this case, a newer controller sends newer payload to older agents.

Until we have proper RPC version pinning feature to enforce older payload format during upgrade (as its implemented in other projects like nova), we leave our agents resistant against unknown arguments sent as part of server notifications. This is achieved by consistently capturing those unknown arguments with keyword arguments and ignoring them on agent side; and by not enforcing newer RPC entry point versions on server side.

This approach is not ideal, because it makes RPC API less strict. Thats why other approaches should be considered for notifications in the future.

More information about RPC versioning.

Interface signature

An RPC interface is defined by its name, version, and (named) arguments that it accepts. There are no strict guarantees that arguments will have expected types or meaning, as long as they are serializable.

Message content versioning

To provide better compatibility guarantees for rolling upgrades, RPC interfaces could also define specific format for arguments they accept. In OpenStack world, its usually implemented using oslo.versionedobjects library, and relying on the library to define serialized form for arguments that are passed through AMQP wire.

Note that Neutron has *not* adopted oslo.versionedobjects library for its RPC interfaces yet (except for QoS feature).

More information about RPC callbacks used for QoS.

Networking backends

Backend software upgrade should not result in any data plane disruptions. Meaning, e.g. Open vSwitch L2 agent should not reset flows or rewire ports; Neutron L3 agent should not delete namespaces left by older version of the agent; Neutron DHCP agent should not require immediate DHCP lease renewal; etc.

The same considerations apply to setups that do not rely on agents. Meaning, f.e. OpenDaylight or OVN controller should not break data plane connectivity during its upgrade process.

Upgrade testing

Grenade is the OpenStack project that is designed to validate upgrade scenarios.

Currently, only offline (non-rolling) upgrade scenario is validated in Neutron gate. The upgrade scenario follows the following steps:

1. the old cloud is set up using latest stable release code
2. all services are stopped
3. code is updated to the patch under review
4. new database migration scripts are applied, if needed
5. all services are started
6. the new cloud is validated with a subset of tempest tests

The scenario validates that no configuration option names are changed in one cycle. More generally, it validates that the new cloud is capable of running using the old configuration files. It also validates that database migration scripts can be executed.

The scenario does *not* validate AMQP versioning compatibility.

Other projects (for example Nova) have so called partial grenade jobs where some services are left running using the old version of code. Such a job would be needed in Neutron gate to validate rolling upgrades for the project. Till that time, its all up to reviewers to catch compatibility issues in patches on review.

Another hole in testing belongs to split migration script branches. Its assumed that an old cloud can successfully run after expand migration scripts from the new cloud are applied to its database; but its not validated in gate.

Review guidelines

There are several upgrade related gotchas that should be tracked by reviewers.

First things first, a general advice to reviewers: make sure new code does not violate requirements set by global OpenStack deprecation policy.

Now to specifics:

1. Configuration options:
   - options should not be dropped from the tree without waiting for deprecation period (currently its one development cycle long) and a deprecation message issued if the deprecated option is used.
   - option values should not change their meaning between releases.
2. Data plane:
   - agent restart should not result in data plane disruption (no Open vSwitch ports reset; no network namespaces deleted; no device names changed).
3. RPC versioning:

- no RPC version major number should be bumped before all agents had a chance to upgrade (meaning, at least one release cycle is needed before compatibility code to handle old clients is stripped from the tree).
- no compatibility code should be added to agent side of AMQP interfaces.
- server code should be able to handle all previous versions of agents, unless the major version of an interface is bumped.
- no RPC interface arguments should change their meaning, or names.
- new arguments added to RPC interfaces should not be mandatory. It means that server should be able to handle old requests, without the new argument specified. Also, if the argument is not passed, the old behaviour before the addition of the argument should be retained.
- minimal client version must not be bumped for server initiated notification changes for at least one cycle.

4. Database migrations:
   - migration code should be split into two branches (contract, expand) as needed. No code that is unsafe to execute while neutron-server is running should be added to expand branch.
   - if possible, contract migrations should be minimized or avoided to reduce the time when API endpoints must be down during database upgrade.

OVN Design Notes

Mapping between Neutron and OVN data models

The primary job of the Neutron OVN ML2 driver is to translate requests for resources into OVNs data model. Resources are created in OVN by updating the appropriate tables in the OVN northbound database (an ovsdb database). This document looks at the mappings between the data that exists in Neutron and what the resulting entries in the OVN northbound DB would look like.

Network

```
Neutron Network:
    id
    name
    subnets
    admin_state_up
    status
    tenant_id
```

Once a network is created, we should create an entry in the Logical Switch table.

```
OVN northbound DB Logical Switch:
    external_ids: {
        'neutron:network_name': network.name
    }
```

Subnet

```
Neutron Subnet:
    id
    name
    ip_version
    network_id
    cidr
    gateway_ip
    allocation_pools
    dns_nameservers
    host_routers
    tenant_id
    enable_dhcp
    ipv6_ra_mode
    ipv6_address_mode
```

Once a subnet is created, we should create an entry in the DHCP Options table with the DHCPv4 or DHCPv6 options.

```
OVN northbound DB DHCP_Options:
    cidr
    options
    external_ids: {
        'subnet_id': subnet.id
    }
```

Port

```
Neutron Port:
    id
    name
    network_id
    admin_state_up
    mac_address
    fixed_ips
    device_id
    device_owner
    tenant_id
    status
```

When a port is created, we should create an entry in the Logical Switch Ports table in the OVN northbound DB.

```
OVN Northbound DB Logical Switch Port:
    switch: reference to OVN Logical Switch
    router_port: (empty)
    name: port.id
    up: (read-only)
    macs: [port.mac_address]
    port_security:
    external_ids: {'neutron:port_name': port.name}
```

If the port has extra DHCP options defined, we should create an entry in the DHCP Options table in the

OVN northbound DB.

```
OVN northbound DB DHCP_Options:
    cidr
    options
    external_ids: {
        'subnet_id': subnet.id,
        'port_id':  port.id
    }
```

Router

```
Neutron Router:
    id
    name
    admin_state_up
    status
    tenant_id
    external_gw_info:
        network_id
        external_fixed_ips: list of dicts
            ip_address
            subnet_id
```

```
OVN Northbound DB Logical Router:
    ip:
    default_gw:
    external_ids:
```

Router Port

```
OVN Northbound DB Logical Router Port:
    router: (reference to Logical Router)
    network: (reference to network this port is connected to)
    mac:
    external_ids:
```

Security Groups

```
Neutron Port:
    id
    security_group: id
    network_id

Neutron Security Group
    id
    name
    tenant_id
    security_group_rules
```

(continues on next page)

(continued from previous page)

```
Neutron Security Group Rule
    id
    tenant_id
    security_group_id
    direction
    remote_group_id
    ethertype
    protocol
    port_range_min
    port_range_max
    remote_ip_prefix
```

```
OVN Northbound DB ACL Rule:
    lswitch:  (reference to Logical Switch - port.network_id)
    priority: (0..65535)
    match: boolean expressions according to security rule
           Translation map (sg_rule  ==> match expression)
           -----------------------------------------------
           sg_rule.direction="Ingress" => "inport=port.id"
           sg_rule.direction="Egress" => "outport=port.id"
           sg_rule.ethertype => "eth.type"
           sg_rule.protocol => "ip.proto"
           sg_rule.port_range_min/port_range_max  =>
                   "port_range_min &lt;= tcp.src &lt;= port_range_max"
                   "port_range_min &lt;= udp.src &lt;= port_range_max"

           sg_rule.remote_ip_prefix => "ip4.src/mask, ip4.dst/mask, ipv6.
↪src/mask, ipv6.dst/mask"

           (all match options for ACL can be found here:
            http://openvswitch.org/support/dist-docs/ovn-nb.5.html)
    action: "allow-related"
    log: true/false
    external_ids: {'neutron:port_id': port.id}
                  {'neutron:security_rule_id': security_rule.id}
```

Security groups maps between three neutron objects to one OVN-NB object, this enable us to do the mapping in various ways, depending on OVN capabilities

The current implementation will use the first option in this list for simplicity, but all options are kept here for future reference

1) For every <neutron port, security rule> pair, define an ACL entry:

```
Leads to many ACL entries.
acl.match = sg_rule converted
example: ((inport==port.id) && (ip.proto == "tcp") &&
         (1024 &lt;= tcp.src &lt;= 4095) && (ip.src==192.168.0.1/16))

external_ids: {'neutron:port_id': port.id}
              {'neutron:security_rule_id': security_rule.id}
```

2) For every <neutron port, security group> pair, define an ACL entry:

```
Reduce the number of ACL entries.
Means we have to manage the match field in case specific rule changes
```

(continues on next page)

(continued from previous page)

```
example:  (((inport==port.id) && (ip.proto == "tcp") &&
          (1024 &lt;= tcp.src &lt;= 4095) && (ip.src==192.168.0.1/16))␣
↪||
          ((outport==port.id) && (ip.proto == "udp") && (1024 &lt;=␣
↪tcp.src &lt;= 4095)) ||
          ((inport==port.id) && (ip.proto == 6) ) ||
          ((inport==port.id) && (eth.type == 0x86dd)))

          (This example is a security group with four security rules)

external_ids: {'neutron:port_id': port.id}
              {'neutron:security_group_id': security_group.id}
```

3) For every <lswitch, security group> pair, define an ACL entry:

```
Reduce even more the number of ACL entries.
Manage complexity increase
example:  (((inport==port.id) && (ip.proto == "tcp") && (1024 &lt;=␣
↪tcp.src &lt;= 4095)
           && (ip.src==192.168.0.1/16)) ||
          ((outport==port.id) && (ip.proto == "udp") && (1024 &lt;=␣
↪tcp.src &lt;= 4095)) ||
          ((inport==port.id) && (ip.proto == 6) ) ||
          ((inport==port.id) && (eth.type == 0x86dd))) ||

          (((inport==port2.id) && (ip.proto == "tcp") && (1024 &lt;=␣
↪tcp.src &lt;= 4095)
          && (ip.src==192.168.0.1/16)) ||
          ((outport==port2.id) && (ip.proto == "udp") && (1024 &lt;=␣
↪tcp.src &lt;= 4095)) ||
          ((inport==port2.id) && (ip.proto == 6) ) ||
          ((inport==port2.id) && (eth.type == 0x86dd)))

external_ids: {'neutron:security_group': security_group.id}
```

Which option to pick depends on OVN match field length capabilities, and the trade off between better performance due to less ACL entries compared to the complexity to manage them.

If the default behaviour is not drop for unmatched entries, a rule with lowest priority must be added to drop all traffic (match==1)

Spoofing protection rules are being added by OVN internally and we need to ignore the automatically added rules in Neutron

Using the native DHCP feature provided by OVN

DHCPv4

OVN implements a native DHCPv4 support which caters to the common use case of providing an IP address to a booting instance by providing stateless replies to DHCPv4 requests based on statically configured address mappings. To do this it allows a short list of DHCPv4 options to be configured and applied at each compute host running ovn-controller.

OVN northbound db provides a table DHCP_Options to store the DHCP options. Logical switch port has a reference to this table.

When a subnet is created and enable_dhcp is True, a new entry is created in this table. The options column stores the DHCPv4 options. These DHCPv4 options are included in the DHCPv4 reply by the ovn-controller when the VIF attached to the logical switch port sends a DHCPv4 request.

In order to map the DHCP_Options row with the subnet, the OVN ML2 driver stores the subnet id in the external_ids column.

When a new port is created, the dhcpv4_options column of the logical switch port refers to the DHCP_Options row created for the subnet of the port. If the port has multiple IPv4 subnets, then the first subnet in the fixed_ips is used.

If the port has extra DHCPv4 options defined, then a new entry is created in the DHCP_Options table for the port. The default DHCP options are obtained from the subnet DHCP_Options table and the extra DHCPv4 options of the port are overridden. In order to map the port DHCP_Options row with the port, the OVN ML2 driver stores both the subnet id and port id in the external_ids column.

If admin wants to disable native OVN DHCPv4 for any particular port, then the admin needs to define the dhcp_disabled with the value true in the extra DHCP options.

Ex. neutron port-update <PORT_ID> extra-dhcp-opt ip_version=4, opt_name=dhcp_disabled, opt_value=false

DHCPv6

OVN implements a native DHCPv6 support similar to DHCPv4. When a v6 subnet is created, the OVN ML2 driver will insert a new entry into DHCP_Options table only when the subnet ipv6_address_mode is not slaac, and enable_dhcp is True.

OVN Neutron Worker and Port status handling

When the logical switch ports VIF is attached or removed to/from the ovn integration bridge, ovn-northd updates the Logical_Switch_Port.up to True or False accordingly.

In order for the OVN Neutron ML2 driver to update the corresponding neutron ports status to ACTIVE or DOWN in the db, it needs to monitor the OVN Northbound db. A neutron worker is created for this purpose.

The implementation of the ovn worker can be found here - networking_ovn.ovsdb.worker.OvnWorker.

Neutron service will create n api workers and m rpc workers and 1 ovn worker (all these workers are separate processes).

Api workers and rpc workers will create ovsdb idl client object (ovs.db.idl.Idl) to connect to the OVN_Northbound db. See networking_ovn.ovsdb.impl_idl_ovn.OvsdbNbOvnIdl and ovsdbapp.backend.ovs_idl.connection.Connection classes for more details.

Ovn worker will create networking_ovn.ovsdb.ovsdb_monitor.OvnIdl class object (which inherits from ovs.db.idl.Idl) to connect to the OVN_Northbound db. On receiving the OVN_Northbound db updates from the ovsdb-server, notify function of OVnIdl is called by the parent class object.

OvnIdl.notify() function passes the received events to the ovsdb_monitor.OvnDbNotifyHandler class. ovsdb_monitor.OvnDbNotifyHandler checks for any changes in the Logical_Switch_Port.up and updates the neutron ports status accordingly.

If notify_nova_on_port_status_changes configuration is set, then neutron would notify nova on port status changes.

ovsdb locks

If there are multiple neutron servers running, then each neutron server will have one ovn worker which listens for the notify events. When the Logical_Switch_Port.up is updated by ovn-northd, we do not want all the neutron servers to handle the event and update the neutron port status. In order for only one neutron server to handle the events, ovsdb locks are used.

At start, each neutron servers ovn worker will try to acquire a lock with id - neutron_ovn_event_lock. The ovn worker which has acquired the lock will handle the notify events.

In case the neutron server with the lock dies, ovsdb-server will assign the lock to another neutron server in the queue.

More details about the ovsdb locks can be found here [1] and [2]

[1] - https://tools.ietf.org/html/draft-pfaff-ovsdb-proto-04#section-4.1.8 [2] - https://github.com/openvswitch/ovs/blob/branch-2.4/python/ovs/db/idl.py#L67

One thing to note is the ovn worker (with OvnIdl) do not carry out any transactions to the OVN Northbound db.

Since the api and rpc workers are not configured with any locks, using the ovsdb lock on the OVN_Northbound and OVN_Southbound DBs by the ovn workers will not have any side effects to the transactions done by these api and rpc workers.

Handling port status changes when neutron server(s) are down

When neutron server starts, ovn worker would receive a dump of all logical switch ports as events. ovsdb_monitor.OvnDbNotifyHandler would sync up if there are any inconsistencies in the port status.

OVN Southbound DB Access

The OVN Neutron ML2 driver has a need to acquire chassis information (hostname and physnets combinations). This is required initially to support routed networks. Thus, the plugin will initiate and maintain a connection to the OVN SB DB during startup.

OpenStack Metadata API and OVN

Introduction

OpenStack Nova presents a metadata API to VMs similar to what is available on Amazon EC2. Neutron is involved in this process because the source IP address is not enough to uniquely identify the source of a metadata request since networks can have overlapping IP addresses. Neutron is responsible for intercepting metadata API requests and adding HTTP headers which uniquely identify the source of the request before forwarding it to the metadata API server.

The purpose of this document is to propose a design for how to enable this functionality when OVN is used as the backend for OpenStack Neutron.

Neutron and Metadata Today

The following blog post describes how VMs access the metadata API through Neutron today.

https://www.suse.com/communities/blog/vms-get-access-metadata-neutron/

In summary, we run a metadata proxy in either the router namespace or DHCP namespace. The DHCP namespace can be used when theres no router connected to the network. The one downside to the DHCP namespace approach is that it requires pushing a static route to the VM through DHCP so that it knows to route metadata requests to the DHCP server IP address.

- Instance sends a HTTP request for metadata to 169.254.169.254
- This request either hits the router or DHCP namespace depending on the route in the instance
- The metadata proxy service in the namespace adds the following info to the request:
  - Instance IP (X-Forwarded-For header)
  - Router or Network-ID (X-Neutron-Network-Id or X-Neutron-Router-Id header)
- The metadata proxy service sends this request to the metadata agent (outside the namespace) via a UNIX domain socket.
- The neutron-metadata-agent service forwards the request to the Nova metadata API service by adding some new headers (instance ID and Tenant ID) to the request [0].

For proper operation, Neutron and Nova must be configured to communicate together with a shared secret. Neutron uses this secret to sign the Instance-ID header of the metadata request to prevent spoofing. This secret is configured through metadata_proxy_shared_secret on both nova and neutron configuration files (optional).

[0] https://opendev.org/openstack/neutron/src/commit/f73f39f2cfcd4eace2bda14c99ead9a8cc8560f4/neutron/agent/metadata/agent.py#L175

Neutron and Metadata with OVN

The current metadata API approach does not translate directly to OVN. There are no Neutron agents in use with OVN. Further, OVN makes no use of its own network namespaces that we could take advantage of like the original implementation makes use of the router and dhcp namespaces.

We must use a modified approach that fits the OVN model. This section details a proposed approach.

Overview of Proposed Approach

The proposed approach would be similar to the *isolated network* case in the current ML2+OVS implementation. Therefore, we would be running a metadata proxy (haproxy) instance on every hypervisor for each network a VM on that host is connected to.

The downside of this approach is that well be running more metadata proxies than were doing now in case of routed networks (one per virtual router) but since haproxy is very lightweight and they will be idling most of the time, it shouldnt be a big issue overall. However, the major benefit of this approach is that we dont have to implement any scheduling logic to distribute metadata proxies across the nodes, nor any HA logic. This, however, can be evolved in the future as explained below in this document.

Also, this approach relies on a new feature in OVN that we must implement first so that an OVN port can be present on *every* chassis (similar to *localnet* ports). This new type of logical port would be *localport* and we will never forward packets over a tunnel for these ports. We would only send packets to the local instance of a *localport*.

Step 1 - Create a port for the metadata proxy

When using the DHCP agent today, Neutron automatically creates a port for the DHCP agent to use. We could do the same thing for use with the metadata proxy (haproxy). Well create an OVN *localport* which will be present on every chassis and this port will have the same MAC/IP address on every host. Eventually, we can share the same neutron port for both DHCP and metadata.

Step 2 - Routing metadata API requests to the correct Neutron port

This works similarly to the current approach.

We would program OVN to include a static route in DHCP responses that routes metadata API requests to the *localport* that is hosting the metadata API proxy.

Also, in case DHCP isnt enabled or the client ignores the route info, we will program a static route in the OVN logical router which will still get metadata requests directed to the right place.

If the DHCP route does not work and the network is isolated, VMs wont get metadata, but this already happens with the current implementation so this approach doesnt introduce a regression.

Step 3 - Management of the namespaces and haproxy instances

We propose a new agent called `neutron-ovn-metadata-agent`. We will run this agent on every hypervisor and it will be responsible for spawning the haproxy instances for managing the OVS interfaces, network namespaces and haproxy processes used to proxy metadata API requests.

Step 4 - Metadata API request processing

Similar to the existing neutron metadata agent, `neutron-ovn-metadata-agent` must act as an intermediary between haproxy and the Nova metadata API service. `neutron-ovn-metadata-agent` is the process that will have access to the host networks where the Nova metadata API exists. Each haproxy will be in a network namespace not able to reach the appropriate host network. Haproxy will add the necessary headers to the metadata API request and then forward it to `neutron-ovn-metadata-agent` over a UNIX domain socket, which matches the behavior of the current metadata agent.

Metadata Proxy Management Logic

In neutron-ovn-metadata-agent.

- On startup:
  - Do a full sync. Ensure we have all the required metadata proxies running. For that, the agent would watch the `Port_Binding` table of the OVN Southbound database and look for all rows with the `chassis` column set to the host the agent is running on. For all those entries, make sure a metadata proxy instance is spawned for every `datapath` (Neutron network) those ports are attached to. The agent will keep record of the list of networks it currently has proxies running on by updating the `external-ids` key `neutron-metadata-proxy-networks` of the OVN `Chassis` record in the OVN Southbound database that corresponds to this host. As an example, this key would look like `neutron-metadata-proxy-networks=NET1_UUID,NET4_UUID` meaning that this chassis is hosting one or more VMs connected to networks 1 and 4 so we should have a

metadata proxy instance running for each. Ensure any running metadata proxies no longer needed are torn down.

- Open and maintain a connection to the OVN Northbound database (using the ovsdbapp library). On first connection, and anytime a reconnect happens:
  - Do a full sync.
- Register a callback for creates/updates/deletes to Logical_Switch_Port rows to detect when metadata proxies should be started or torn down. `neutron-ovn-metadata-agent` will watch OVN Southbound database (`Port_Binding` table) to detect when a port gets bound to its chassis. At that point, the agent will make sure that theres a metadata proxy attached to the OVN *localport* for the network which this port is connected to.
- When a new network is created, we must create an OVN *localport* for use as a metadata proxy. This port will be owned by `network:dhcp` so that it gets auto deleted upon the removal of the network and it will remain `DOWN` and not bound to any chassis. The metadata port will be created regardless of the DHCP setting of the subnets within the network as long as the metadata service is enabled.
- When a network is deleted, we must tear down the metadata proxy instance (if present) on the host and delete the corresponding OVN *localport* (which will happen automatically as its owned by `network:dhcp`).

Launching a metadata proxy includes:

- Creating a network namespace:

```
$ sudo ip netns add <ns-name>
```

- Creating a VETH pair (OVS upgrades that upgrade the kernel module will make internal ports go away and then brought back by OVS scripts. This may cause some disruption. Therefore, veth pairs are preferred over internal ports):

```
$ sudo ip link add <iface-name>0 type veth peer name <iface-name>1
```

- Creating an OVS interface and placing one end in that namespace:

```
$ sudo ovs-vsctl add-port br-int <iface-name>0
$ sudo ip link set <iface-name>1 netns <ns-name>
```

- Setting the IP and MAC addresses on that interface:

```
$ sudo ip netns exec <ns-name> \
> ip link set <iface-name>1 address <neutron-port-mac>
$ sudo ip netns exec <ns-name> \
> ip addr add <neutron-port-ip>/<netmask> dev <iface-name>1
```

- Bringing the VETH pair up:

```
$ sudo ip netns exec <ns-name> ip link set <iface-name>1 up
$ sudo ip link set <iface-name>0 up
```

- Set `external-ids:iface-id=NEUTRON_PORT_UUID` on the OVS interface so that OVN is able to correlate this new OVS interface with the correct OVN logical port:

```
$ sudo ovs-vsctl set Interface <iface-name>0 external_ids:iface-id=
↪<neutron-port-uuid>
```

- Starting haproxy in this network namespace.
- Add the network UUID to `external-ids:neutron-metadata-proxy-networks` on the Chassis table for our chassis in OVN Southbound database.

Tearing down a metadata proxy includes:

- Removing the network UUID from our chassis.
- Stopping haproxy.
- Deleting the OVS interface.
- Deleting the network namespace.

Other considerations

This feature will be enabled by default when using `ovn` driver, but there should be a way to disable it in case operators who dont need metadata dont have to deal with the complexity of it (haproxy instances, network namespaces, etcetera). In this case, the agent would not create the neutron ports needed for metadata.

There could be a race condition when the first VM for a certain network boots on a hypervisor if it does so before the metadata proxy instance has been spawned.

Right now, the `vif-plugged` event to Nova is sent out when the up column in the OVN Northbound databases Logical_Switch_Port table changes to True, indicating that the VIF is now up. To overcome this race condition we want to wait until all network UUIDs to which this VM is connected to are present in `external-ids:neutron-metadata-proxy-networks` on the Chassis table for our chassis in OVN Southbound database. This will delay the event to Nova until the metadata proxy instance is up and running on the host ensuring the VM will be able to get the metadata on boot.

Alternatives Considered

Alternative 1: Build metadata support into ovn-controller

Weve been building some features useful to OpenStack directly into OVN. DHCP and DNS are key examples of things weve replaced by building them into ovn-controller. The metadata API case has some key differences that make this a less attractive solution:

The metadata API is an OpenStack specific feature. DHCP and DNS by contrast are more clearly useful outside of OpenStack. Building metadata API proxy support into ovn-controller means embedding an HTTP and TCP stack into ovn-controller. This is a significant degree of undesired complexity.

This option has been ruled out for these reasons.

Alternative 2: Distributed metadata and High Availability

In this approach, we would spawn a metadata proxy per virtual router or per network (if isolated), thus, improving the number of metadata proxy instances running in the cloud. However, scheduling and HA have to be considered. Also, we wouldnt need the OVN *localport* implementation.

`neutron-ovn-metadata-agent` would run on any host that we wish to be able to host metadata API proxies. These hosts must also be running ovn-controller.

Each of these hosts will have a Chassis record in the OVN southbound database created by ovn-controller. The Chassis table has a column called `external_ids` which can be used for general metadata however we see fit. `neutron-ovn-metadata-agent` will update its corresponding Chassis record with an external-id of `neutron-metadata-proxy-host=true` to indicate that this OVN chassis is one capable of hosting metadata proxy instances.

Once we have a way to determine hosts capable of hosting metadata API proxies, we can add logic to the ovn ML2 driver that schedules metadata API proxies. This would be triggered by Neutron API requests.

The output of the scheduling process would be setting an `external_ids` key on a Logical_Switch_Port in the OVN northbound database that corresponds with a metadata proxy. The key could be something like `neutron-metadata-proxy-chassis=CHASSIS_HOSTNAME`.

`neutron-ovn-metadata-agent` on each host would also be watching for updates to these Logical_Switch_Port rows. When it detects that a metadata proxy has been scheduled locally, it will kick off the process to spawn the local haproxy instance and get it plugged into OVN.

HA must also be considered. We must know when a host goes down so that all metadata proxies scheduled to that host can be rescheduled. This is almost the exact same problem we have with L3 HA. When a host goes down, we need to trigger rescheduling gateways to other hosts. We should ensure that the approach used for rescheduling L3 gateways can be utilized for rescheduling metadata proxies, as well.

In neutron-server (ovn mechanism driver) .

Introduce a new ovn driver configuration option:

- `[ovn] isolated_metadata=[True|False]`

Events that trigger scheduling a new metadata proxy:

- If isolated_metadata is True
  - When a new network is created, we must create an OVN logical port for use as a metadata proxy and then schedule this to one of the `neutron-ovn-metadata-agent` instances.
- If isolated_metadata is False
  - When a network is attached to or removed from a logical router, ensure that at least one of the networks has a metadata proxy port already created. If not, pick a network and create a metadata proxy port and then schedule it to an agent. At this point, we need to update the static route for metadata API.

Events that trigger unscheduling an existing metadata proxy:

- When a network is deleted, delete the metadata proxy port if it exists and unschedule it from a `neutron-ovn-metadata-agent`.

To schedule a new metadata proxy:

- Determine the list of available OVN Chassis that can host metadata proxies by reading the `Chassis` table of the OVN Southbound database. Look for chassis that have an external-id of `neutron-metadata-proxy-host=true`.
- Of the available OVN chassis, choose the one least loaded, or currently hosting the fewest number of metadata proxies.
- Set `neutron-metadata-proxy-chassis=CHASSIS_HOSTNAME` as an external-id on the Logical_Switch_Port in the OVN Northbound database that corresponds to the neutron port used for this metadata proxy. `CHASSIS_HOSTNAME` maps to the hostname row of a Chassis record in the OVN Southbound database.

This approach has been ruled out for its complexity although we have analyzed the details deeply because, eventually, and depending on the implementation of L3 HA, we will want to evolve to it.

Other References

- Haproxy config https://review.openstack.org/#/c/431691/34/neutron/agent/metadata/driver.py
- https://engineeringblog.yelp.com/2015/04/true-zero-downtime-haproxy-reloads.html

Neutron/OVN Database consistency

This document presents the problem and proposes a solution for the data consistency issue between the Neutron and OVN databases. Although the focus of this document is OVN this problem is common enough to be present in other ML2 drivers (e.g OpenDayLight, BigSwitch, etc). Some of them already contain a mechanism in place for dealing with it.

Problem description

In a common Neutron deployment model there could have multiple Neutron API workers processing requests. For each request, the worker will update the Neutron database and then invoke the ML2 driver to translate the information to that specific SDN data model.

There are at least two situations that could lead to some inconsistency between the Neutron and the SDN databases, for example:

Problem 1: Neutron API workers race condition

```
In Neutron:
  with neutron_db_transaction:
       update_neutron_db()
       ml2_driver.update_port_precommit()
  ml2_driver.update_port_postcommit()

In the ML2 driver:
  def update_port_postcommit:
      port = neutron_db.get_port()
      update_port_in_ovn(port)
```

Imagine the case where a port is being updated twice and each request is being handled by a different API worker. The method responsible for updating the resource in the OVN (`update_port_postcommit`) is not atomic and invoked outside of the Neutron database transaction. This could lead to a problem where the order in which the updates are committed to the Neutron database are different than the order that they are committed to the OVN database, resulting in an inconsistency.

This problem has been reported at bug #1605089.

Problem 2: Backend failures

Another situation is when the changes are already committed in Neutron but an exception is raised upon trying to update the OVN database (e.g lost connectivity to the `ovsdb-server`). We currently dont have a good way of handling this problem, obviously it would be possible to try to immediately rollback the changes in the Neutron database and raise an exception but, that rollback itself is an operation that could also fail.

Plus, rollbacks is not very straight forward when it comes to updates or deletes. In a case where a VM is being teared down and OVN fail to delete a port, re-creating that port in Neutron doesnt necessary fix the problem. The decommission of a VM involves many other things, in fact, we could make things even worse by leaving some dirty data around. I believe this is a problem that would be better dealt with by other methods.

Proposed change

In order to fix the problems presented at the *Problem description* section this document proposes a solution based on the Neutrons `revision_number` attribute. In summary, for every resource in Neutron theres an attribute called `revision_number` which gets incremented on each update made on that resource. For example:

```
$ openstack port create --network nettest porttest
...
| revision_number | 2 |
...

$ openstack port set porttest --mac-address 11:22:33:44:55:66

$ mysql -e "use neutron; select standard_attr_id from ports where id=\
↪"91c08021-ded3-4c5a-8d57-5b5c389f8e39\";"
+------------------+
| standard_attr_id |
+------------------+
|             1427 |
+------------------+

$ mysql -e "use neutron; SELECT revision_number FROM standardattributes␣
↪WHERE id=1427;"
+-----------------+
| revision_number |
+-----------------+
|               3 |
+-----------------+
```

This document proposes a solution that will use the *revision_number* attribute for three things:

1. Perform a compare-and-swap operation based on the resource version
2. Guarantee the order of the updates (*Problem 1*)
3. Detecting when resources in Neutron and OVN are out-of-sync

But, before any of points above can be done we need to change the ovn driver code to:

#1 - Store the revision_number referent to a change in OVNDB

To be able to compare the version of the resource in Neutron against the version in OVN we first need to know which version the OVN resource is present at.

Fortunately, each table in the OVNDB contains a special column called `external_ids` which external systems (like Neutron) can use to store information about its own resources that corresponds to the entries in OVNDB.

So, every time a resource is created or updated in OVNDB by ovn driver, the Neutron `revision_number` referent to that change will be stored in the `external_ids` column of that resource. That will allow ovn driver to look at both databases and detect whether the version in OVN is up-to-date with Neutron or not.

#2 - Ensure correctness when updating OVN

As stated in *Problem 1*, simultaneous updates to a single resource will race and, with the current code, the order in which these updates are applied is not guaranteed to be the correct order. That means that, if two or more updates arrives we cant prevent an older version of that update to be applied after a newer one.

This document proposes creating a special `OVSDB command` that runs as part of the same transaction that is updating a resource in OVNDB to prevent changes with a lower `revision_number` to be applied in case the resource in OVN is at a higher `revision_number` already.

This new OVSDB command needs to basically do two things:

1. Add a verify operation to the `external_ids` column in OVNDB so that if another client modifies that column mid-operation the transaction will be restarted.

A better explanation of what verify does is described at the doc string of the Transaction class in the OVS code itself, I quote:

> Because OVSDB handles multiple clients, it can happen that between the time that OVSDB client A reads a column and writes a new value, OVSDB client B has written that column. Client As write should not ordinarily overwrite client Bs, especially if the column in question is a map column that contains several more or less independent data items. If client A adds a verify operation before it writes the column, then the transaction fails in case client B modifies it first. Client A will then see the new value of the column and compose a new transaction based on the new contents written by client B.

2. Compare the `revision_number` from the update against what is presently stored in OVNDB. If the version in OVNDB is already higher than the version in the update, abort the transaction.

So basically this new command is responsible for guarding the OVN resource by not allowing old changes to be applied on top of new ones. Heres a scenario where two concurrent updates comes in the wrong order and how the solution above will deal with it:

Neutron worker 1 (NW-1): Updates a port with address A (revision_number: 2)

Neutron worker 2 (NW-2): Updates a port with address B (revision_number: 3)

TXN 1: NW-2 transaction is committed first and the OVN resource now has RN 3

TXN 2: NW-1 transaction detects the change in the external_ids column and is restarted

TXN 2: NW-1 the new command now sees that the OVN resource is at RN 3, which is higher than the update version (RN 2) and aborts the transaction.

Theres a bit more for the above to work with the current ovn driver code, basically we need to tidy up the code to do two more things.

1. Consolidate changes to a resource in a single transaction.

This is important regardless of this spec, having all changes to a resource done in a single transaction minimizes the risk of having half-changes written to the database in case of an eventual problem. This should be done already but its important to have it here in case we find more examples like that as we code.

2. When doing partial updates, use the OVNDB as the source of comparison to create the deltas.

Being able to do a partial update in a resource is important for performance reasons; its a way to minimize the number of changes that will be performed in the database.

Right now, some of the update() methods in ovn driver creates the deltas using the *current* and *original* parameters that are passed to it. The *current* parameter is, as the name says, the current version of the object present in the Neutron DB. The *original* parameter is the previous version (current - 1) of that object.

The problem of creating the deltas by comparing these two objects is because only the data in the Neutron DB is used for it. We need to stop using the *original* object for it and instead we should create the delta based on the *current* version of the Neutron DB against the data stored in the OVNDB to be able to detect the real differences between the two databases.

So in summary, to guarantee the correctness of the updates this document proposes to:

1. Create a new OVSDB command is responsible for comparing revision numbers and aborting the transaction, when needed.
2. Consolidate changes to a resource in a single transaction (should be done already)
3. When doing partial updates, create the deltas based in the current version in the Neutron DB and the OVNDB.

#3 - Detect and fix out-of-sync resources

When things are working as expected the above changes should ensure that Neutron DB and OVNDB are in sync but, what happens when things go bad ? As per *Problem 2*, things like temporarily losing connectivity with the OVNDB could cause changes to fail to be committed and the databases getting out-of-sync. We need to be able to detect the resources that were affected by these failures and fix them.

We do already have the means to do it, similar to what the ovn_db_sync.py script does we could fetch all the data from both databases and compare each resource. But, depending on the size of the deployment this can be really slow and costy.

This document proposes an optimization for this problem to make it efficient enough so that we can run it periodically (as a periodic task) and not manually as a script anymore.

First, we need to create an additional table in the Neutron database that would serve as a cache for the revision numbers in **OVNDB**.

The new table schema could look this:

Column name	Type	Description
standard_attr_id	Integer	Primary key. The reference ID from the standardattributes table in Neutron for that resource. ONDELETE SET NULL.
resource_uuid	String	The UUID of the resource
resource_type	String	The type of the resource (e.g, Port, Router,)
revision_number	Integer	The version of the object present in OVN
acquired_at	DateTime	The time that the entry was create. For troubleshooting purposes
updated_at	DateTime	The time that the entry was updated. For troubleshooting purposes

For the different actions: Create, update and delete; this table will be used as:

1. Create:

In the create_*_precommit() method, we will create an entry in the new table within the same Neutron transaction. The revision_number column for the new entry will have a placeholder value until the resource is successfully created in OVNDB.

In case we fail to create the resource in OVN (but succeed in Neutron) we still have the entry logged in the new table and this problem can be detected by fetching all resources where the revision_number column value is equal to the placeholder value.

The pseudo-code will look something like this:

```
def create_port_precommit(ctx, port):
    create_initial_revision(port['id'], revision_number=-1,
                            session=ctx.session)

def create_port_postcommit(ctx, port):
    create_port_in_ovn(port)
    bump_revision(port['id'], revision_number=port['revision_number'])
```

2. Update:

For update its simpler, we need to bump the revision number for that resource **after** the OVN transaction is committed in the update_*_postcommit() method. That way, if an update fails to be applied to OVN the inconsistencies can be detected by a JOIN between the new table and the `standardattributes` table where the revision_number columns does not match.

The pseudo-code will look something like this:

```
def update_port_postcommit(ctx, port):
    update_port_in_ovn(port)
    bump_revision(port['id'], revision_number=port['revision_number'])
```

3. Delete:

The `standard_attr_id` column in the new table is a foreign key constraint with a `ONDELETE=SET NULL` set. That means that, upon Neutron deleting a resource the `standard_attr_id` column in the new table will be set to *NULL*.

If deleting a resource succeeds in Neutron but fails in OVN, the inconsistency can be detect by looking at all resources that has a `standard_attr_id` equals to NULL.

The pseudo-code will look something like this:

```
def delete_port_postcommit(ctx, port):
    delete_port_in_ovn(port)
    delete_revision(port['id'])
```

With the above optimization its possible to create a periodic task that can run quite frequently to detect and fix the inconsistencies caused by random backend failures.

Note: Theres no lock linking both database updates in the postcommit() methods. So, its true that the method bumping the revision_number column in the new table in Neutron DB could still race but, that should be fine because this table acts like a cache and the real revision_number has been written in OVNDB.

The mechanism that will detect and fix the out-of-sync resources should detect this inconsistency as well and, based on the revision_number in OVNDB, decide whether to sync the resource or only bump the revision_number in the cache table (in case the resource is already at the right version).

Refereces

- Theres a chain of patches with a proof of concept for this approach, they start at: https://review.openstack.org/#/c/517049/

Alternatives

Journaling

An alternative solution to this problem is *journaling*. The basic idea is to create another table in the Neutron database and log every operation (create, update and delete) instead of passing it directly to the SDN controller.

A separated thread (or multiple instances of it) is then responsible for reading this table and applying the operations to the SDN backend.

This approach has been used and validated by drivers such as networking-odl.

An attempt to implement this approach in *ovn driver* can be found here.

Some things to keep in mind about this approach:

- The code can get quite complex as this approach is not only about applying the changes to the SDN backend asynchronously. The dependencies between each resource as well as their operations also needs to be computed. For example, before attempting to create a router port the router that this port belongs to needs to be created. Or, before attempting to delete a network all the dependent resources on it (subnets, ports, etc) needs to be processed first.
- The number of journal threads running can cause problems. In my tests I had three controllers, each one with 24 CPU cores (Intel Xeon E5-2620 with hyperthreading enabled) and 64GB RAM. Running 1 journal thread per Neutron API worker has caused `ovsdb-server` to misbehave when under heavy pressure[1]. Running multiple journal threads seem to be causing other types of problems in other drivers as well.
- When under heavy pressure[1], I noticed that the journal threads could come to a halt (or really slowed down) while the API workers were handling a lot of requests. This resulted in some operations taking more than a minute to be processed. This behaviour can be seem in this screenshot.
- Given that the 1 journal thread per Neutron API worker approach is problematic, determining the right number of journal threads is also difficult. In my tests, Ive noticed that 3 journal threads per controller worked better but that number was pure based on `trial & error`. In production this number should probably be calculated based in the environment, perhaps something like TripleO (or any upper layer) would be in a better position to make that decision.
- At least temporarily, the data in the Neutron database is duplicated between the normal tables and the journal one.
- Some operations like creating a new resource via Neutrons API will return HTTP 201, which indicates that the resource has been created and is ready to be used, but as these resources are created asynchronously one could argue that the HTTP codes are now misleading. As a note, the resource will be created at the Neutron database by the time the HTTP request returns but it may not be present in the SDN backend yet.

[1] I ran the tests using Browbeat which is basically orchestrate Openstack Rally and monitor the machines usage of resources.

Given all considerations, this approach is still valid and the fact that its already been used by other ML2 drivers makes it more open for collaboration and code sharing.

OpenStack LoadBalancer API and OVN

Introduction

Load balancing is essential for enabling simple or automatic delivery scaling and availability since application delivery, scaling and availability are considered vital features of any cloud. Octavia is an open source, operator-scale load balancing solution designed to work with OpenStack.

The purpose of this document is to propose a design for how we can use OVN as the backend for OpenStacks LoadBalancer API provided by Octavia.

Octavia LoadBalancers Today

A Detailed design analysis of Octavia is available here:

https://docs.openstack.org/octavia/queens/contributor/design/version0.5/component-design.html

Currently, Octavia uses the in-built Amphorae driver to fulfill the Loadbalancing requests in Openstack. Amphorae can be a Virtual machine, container, dedicated hardware, appliance or device that actually performs the task of load balancing in the Octavia system. More specifically, an amphora takes requests from clients on the front-end and distributes these to back-end systems. Amphorae communicates with its controllers over the LoadBalancers network through a driver interface on the controller.

Amphorae needs a placeholder, such as a separate VM/Container for deployment, so that it can handle the LoadBalancers requests. Along with this, it also needs a separate network (termed as lb-mgmt-network) which handles all Amphorae requests.

Amphorae has the capability to handle L4 (TCP/UDP) as well as L7 (HTTP) LoadBalancer requests and provides monitoring features using HealthMonitors.

Octavia with OVN

OVN native LoadBalancer currently supports L4 protocols, with support for L7 protocols aimed for in future releases. Currently it also does not have any monitoring facility. However, it does not need any extra hardware/VM/Container for deployment, which is a major positive point when compared with Amphorae. Also, it does not need any special network to handle the LoadBalancers requests as they are taken care by OpenFlow rules directly. And, though OVN does not have support for TLS, it is in the works and once implemented can be integrated with Octavia.

This following section details about how OVN can be used as an Octavia driver.

Overview of Proposed Approach

The OVN Driver for Octavia runs under the scope of Octavia. Octavia API receives and forwards calls to the OVN Driver.

Step 1 - Creating a LoadBalancer

Octavia API receives and issues a LoadBalancer creation request on a network to the OVN Provider driver. OVN driver creates a LoadBalancer in the OVN NorthBound DB and asynchronously updates the Octavia DB with the status response. A VIP port is created in Neutron when the LoadBalancer creation is complete. The VIP information however is not updated in the NorthBound DB until the Members are associated with the LoadBalancers Pool.

Step 2 - Creating LoadBalancer entities (Pools, Listeners, Members)

Once a LoadBalancer is created by OVN in its NorthBound DB, users can now create Pools, Listeners and Members associated with the LoadBalancer using the Octavia API. With the creation of each entity, the LoadBalancers *external_ids* column in the NorthBound DB would be updated and corresponding Logical and Openflow rules would be added for handling them.

Step 3 - LoadBalancer request processing

When a user sends a request to the VIP IP address, OVN pipeline takes care of load balancing the VIP request to one of the backend members. More information about this can be found in the ovn-northd man pages.

OVN LoadBalancer Driver Logic

- On startup: Open and maintain a connection to the OVN Northbound DB (using the ovsdbapp library). On first connection, and anytime a reconnect happens:
  - Do a full sync.
- Register a callback when a new interface is added to a router or deleted from a router.
- When a new LoadBalancer L1 is created, create a Row in OVNs `Load_Balancer` table and update its entries for name and network references. If the network on which the LoadBalancer is created, is associated with a router, say R1, then add the router reference to the LoadBalancers *external_ids* and associate the LoadBalancer to the router. Also associate the LoadBalancer L1 with all those networks which have an interface on the router R1. This is required so that Logical Flows for inter-network communication while using the LoadBalancer L1 is possible. Also, during this time, a new port is created via Neutron which acts as a VIP Port. The information of this new port is not visible on the OVNs NorthBound DB till a member is added to the LoadBalancer.
- If a new network interface is added to the router R1 described above, all the LoadBalancers on that network are associated with the router R1 and all the LoadBalancers on the router are associated with the new network.
- If a network interface is removed from the router R1, then all the LoadBalancers which have been solely created on that network (identified using the *ls_ref* attribute in the LoadBalancers *external_ids*) are removed from the router. Similarly those LoadBalancers which are associated with the network but not actually created on that network are removed from the network.
- LoadBalancer can either be deleted with all its children entities using the *cascade* option, or its members/pools/listeners can be individually deleted. When the LoadBalancer is deleted, its references and associations from all networks and routers are removed. This might change in the

future once the association of LoadBalancers with networks/routers are changed to *weak* from *strong* [3]. Also the VIP port is deleted when the LoadBalancer is deleted.

OVN LoadBalancer at work

OVN Northbound schema [5] has a table to store LoadBalancers. The table looks like:

```
"Load_Balancer": {
    "columns": {
        "name": {"type": "string"},
        "vips": {
            "type": {"key": "string", "value": "string",
                     "min": 0, "max": "unlimited"}},
        "protocol": {
            "type": {"key": {"type": "string",
                             "enum": ["set", ["tcp", "udp"]]},
                             "min": 0, "max": 1}},
        "external_ids": {
            "type": {"key": "string", "value": "string",
                     "min": 0, "max": "unlimited"}}},
        "isRoot": true},
```

There is a `load_balancer` column in the Logical_Switch table (which corresponds to a Neutron network) as well as the Logical_Router table (which corresponds to a Neutron router) referring back to the Load_Balancer table.

The OVN driver updates the OVN Northbound DB. When a LoadBalancer is created, a row in this table is created. And when the listeners and members are added, vips column is updated accordingly. And the Logical_Switchs `load_balancer` column is also updated accordingly.

ovn-northd service which monitors for changes to the OVN Northbound DB, generates OVN logical flows to enable load balancing and ovn-controller running on each compute node, translates the logical flows into actual OpenFlow rules.

The status of each entity in the Octavia DB is managed according to [4]

Below are few examples on what happens when LoadBalancer commands are executed and what changes in the Load_Balancer Northbound DB table.

1. Create a LoadBalancer:

```
$ openstack loadbalancer create --provider ovn --vip-subnet-
↪id=private lb1

$ ovn-nbctl list load_balancer
_uuid           : 9dd65bae-2501-43f2-b34e-38a9cb7e4251
external_ids    : {
    lr_ref="neutron-52b6299c-6e38-4226-a275-77370296f257",
    ls_refs="{\"neutron-2526c68a-5a9e-484c-8e00-0716388f6563\": 1}",
    neutron:vip="10.0.0.10",
    neutron:vip_port_id="2526c68a-5a9e-484c-8e00-0716388f6563"}
name            : "973a201a-8787-4f6e-9b8f-ab9f93c31f44"
protocol        : []
vips            : {}
```

2. Create a pool:

```
$ openstack loadbalancer pool create --name p1 --loadbalancer lb1
--protocol TCP --lb-algorithm SOURCE_IP_PORT

$ ovn-nbctl list load_balancer
_uuid           : 9dd65bae-2501-43f2-b34e-38a9cb7e4251
external_ids    : {
    lr_ref="neutron-52b6299c-6e38-4226-a275-77370296f257",
    ls_refs="{\"neutron-2526c68a-5a9e-484c-8e00-0716388f6563\": 1}",
    "pool_f2ddf7a6-4047-4cc9-97be-1d1a6c47ece9"="", neutron:vip="10.0.
↪0.10",
    neutron:vip_port_id="2526c68a-5a9e-484c-8e00-0716388f6563"}
name            : "973a201a-8787-4f6e-9b8f-ab9f93c31f44"
protocol        : []
vips            : {}
```

3. Create a member:

```
$ openstack loadbalancer member create --address 10.0.0.107
 --subnet-id 2d54ec67-c589-473b-bc67-41f3d1331fef --protocol-port 80␣
↪p1

$ ovn-nbctl list load_balancer
_uuid           : 9dd65bae-2501-43f2-b34e-38a9cb7e4251
external_ids    : {
    lr_ref="neutron-52b6299c-6e38-4226-a275-77370296f257",
    ls_refs="{\"neutron-2526c68a-5a9e-484c-8e00-0716388f6563\": 2}",
    "pool_f2ddf7a6-4047-4cc9-97be-1d1a6c47ece9"=
    "member_579c0c9f-d37d-4ba5-beed-cabf6331032d_10.0.0.107:80",
    neutron:vip="10.0.0.10",
    neutron:vip_port_id="2526c68a-5a9e-484c-8e00-0716388f6563"}
name            : "973a201a-8787-4f6e-9b8f-ab9f93c31f44"
protocol        : []
vips            : {}
```

4. Create another member:

```
$ openstack loadbalancer member create --address 20.0.0.107
 --subnet-id c2e2da10-1217-4fe2-837a-1c45da587df7 --protocol-port 80␣
↪p1

$ ovn-nbctl list load_balancer
_uuid           : 9dd65bae-2501-43f2-b34e-38a9cb7e4251
external_ids    : {
    lr_ref="neutron-52b6299c-6e38-4226-a275-77370296f257",
    ls_refs="{\"neutron-2526c68a-5a9e-484c-8e00-0716388f6563\": 2,
          \"neutron-12c42705-3e15-4e2d-8fc0-070d1b80b9ef\": 1}",
    "pool_f2ddf7a6-4047-4cc9-97be-1d1a6c47ece9"=
    "member_579c0c9f-d37d-4ba5-beed-cabf6331032d_10.0.0.107:80,
     member_d100f2ed-9b55-4083-be78-7f203d095561_20.0.0.107:80",
    neutron:vip="10.0.0.10",
    neutron:vip_port_id="2526c68a-5a9e-484c-8e00-0716388f6563"}
name            : "973a201a-8787-4f6e-9b8f-ab9f93c31f44"
protocol        : []
vips            : {}
```

5. Create a listener:

```
$ openstack loadbalancer listener create --name l1 --protocol TCP
 --protocol-port 82 --default-pool p1 lb1

$ ovn-nbctl list load_balancer
_uuid          : 9dd65bae-2501-43f2-b34e-38a9cb7e4251
external_ids   : {
    lr_ref="neutron-52b6299c-6e38-4226-a275-77370296f257",
    ls_refs="{\"neutron-2526c68a-5a9e-484c-8e00-0716388f6563\": 2,
              \"neutron-12c42705-3e15-4e2d-8fc0-070d1b80b9ef\": 1}",
    "pool_f2ddf7a6-4047-4cc9-97be-1d1a6c47ece9"="10.0.0.107:80,20.0.0.
↪107:80",
    "listener_12345678-2501-43f2-b34e-38a9cb7e4132"=
        "82:pool_f2ddf7a6-4047-4cc9-97be-1d1a6c47ece9",
    neutron:vip="10.0.0.10",
    neutron:vip_port_id="2526c68a-5a9e-484c-8e00-0716388f6563"}
name           : "973a201a-8787-4f6e-9b8f-ab9f93c31f44"
protocol       : []
vips           : {"10.0.0.10:82"="10.0.0.107:80,20.0.0.107:80"}
```

As explained earlier in the design section:

- If a network N1 has a LoadBalancer LB1 associated to it and one of its interfaces is added to a router R1, LB1 is associated with R1 as well.
- If a network N2 has a LoadBalancer LB2 and one of its interfaces is added to the router R1, then R1 will have both LoadBalancers LB1 and LB2. N1 and N2 will also have both the LoadBalancers associated to them. However, kindly note that though network N1 would have both LB1 and LB2 LoadBalancers associated with it, only LB1 would be the LoadBalancer which has a direct reference to the network N1, since LB1 was created on N1. This is visible in the `ls_ref` key of the `external_ids` column in LB1s entry in the `load_balancer` table.
- If a network N3 is added to the router R1, N3 will also have both LoadBalancers (LB1, LB2) associated to it.
- If the interface to network N2 is removed from R1, network N2 will now only have LB2 associated with it. Networks N1 and N3 and router R1 will have LoadBalancer LB1 associated with them.

Limitations

Following actions are not supported by OVN Driver:

- Creating a LoadBalancer/Listener/Pool with L7 Protocol
- Creating HealthMonitors
- Currently only one algorithm is supported for pool management (Source IP Port)
- Creating Listeners and Pools with different protocols. They should be of the same protocol type.

Following issue exists with OVNs integration with Octavia:

- If creation/deletion of a LoadBalancer, Listener, Pool or Member fails, then the corresponding object will remain in the DB in a PENDING_* state.

Support Matrix

A detailed matrix of the operations supported by OVN Provider driver in Octavia can be found in https://docs.openstack.org/octavia/latest/user/feature-classification/index.html

Other References

[1] Octavia API: https://docs.openstack.org/api-ref/load-balancer/v2/

[2] Octavia Glossary: https://docs.openstack.org/octavia/queens/reference/glossary.html

[3] https://github.com/openvswitch/ovs/commit/612f80fa8ebf88dad2e204364c6c02b451dca36c

[4] https://docs.openstack.org/api-ref/load-balancer/v2/index.html#status-codes

[5] https://github.com/openvswitch/ovs/blob/d1b235d7a6246e00d4afc359071d3b6b3ed244c3/ovn/ovn-nb.ovsschema#L117

Distributed OVSDB events handler

This document presents the problem and proposes a solution for handling OVSDB events in a distributed fashion in ovn driver.

Problem description

In ovn driver, the OVSDB Monitor class is responsible for listening to the OVSDB events and performing certain actions on them. We use it extensively for various tasks including critical ones such as monitoring for port binding events (in order to notify Neutron/Nova that a port has been bound to a certain chassis). Currently, this class uses a distributed OVSDB lock to ensure that only one instance handles those events at a time.

The problem with this approach is that it creates a bottleneck because even if we have multiple Neutron Workers running at the moment, only one is actively handling those events. And, this problem is highlighted even more when working with technologies such as containers which rely on creating multiple ports at a time and waiting for them to be bound.

Proposed change

In order to fix this problem, this document proposes using a Consistent Hash Ring to split the load of handling events across multiple Neutron Workers.

A new table called `ovn_hash_ring` will be created in the Neutron Database where the Neutron Workers capable of handling OVSDB events will be registered. The table will use the following schema:

Column name	Type	Description
node_uuid	String	Primary key. The unique identification of a Neutron Worker.
hostname	String	The hostname of the machine this Node is running on.
created_at	Date-Time	The time that the entry was created. For troubleshooting purposes.
updated_at	Date-Time	The time that the entry was updated. Used as a heartbeat to indicate that the Node is still alive.

This table will be used to form the Consistent Hash Ring. Fortunately, we have an implementation already in the tooz library of OpenStack. It was contributed by the Ironic team which also uses this data structure in order to spread the API request load across multiple Ironic Conductors.

Heres how a Consistent Hash Ring from tooz works:

```
from tooz import hashring

hring = hashring.HashRing({'worker1', 'worker2', 'worker3'})

# Returns set(['worker3'])
hring[b'event-id-1']

# Returns set(['worker1'])
hring[b'event-id-2']
```

How OVSDB Monitor will use the Ring

Every instance of the OVSDB Monitor class will be listening to a series of events from the OVSDB database and each of them will have a unique ID registered in the database which will be part of the *Consistent Hash Ring*.

When an event arrives, each OVSDB Monitor instance will hash that event UUID and the ring will return one instance ID, which will then be compared with its own ID and if it matches that instance will then process the event.

Verifying status of OVSDB Monitor instance

A new maintenance task will be created in ovn driver which will update the `updated_at` column from the `ovn_hash_ring` table for the entries matching its hostname indicating that all Neutron Workers running on that hostname are alive.

Note that only a single maintenance instance runs on each machine so the writes to the Neutron database are optimized.

When forming the ring, the code should check for entries where the value of `updated_at` column is newer than a given timeout. Entries that havent been updated in a certain time wont be part of the ring. If the ring already exists it will be re-balanced.

Clean up and minimizing downtime window

Apart from heartbeating, we need to make sure that we remove the Nodes from the ring when the service is stopped or killed.

By stopping the `neutron-server` service, all Nodes sharing the same hostname as the machine where the service is running will be removed from the `ovn_hash_ring` table. This is done by handling the SIGTERM event. Upon this event arriving, ovn driver should invoke the clean up method and then let the process halt.

Unfortunately nothing can be done in case of a SIGKILL, this will leave the nodes in the database and they will be part of the ring until the timeout is reached or the service is restarted. This can introduce a window of time which can result in some events being lost. The current implementation shares the same problem, if the instance holding the current OVSDB lock is killed abruptly, events will be lost until the lock is moved on to the next instance which is alive. One could argue that the current implementation aggravates the problem because all events will be lost where with the distributed mechanism **some** events will be lost. As far as distributed systems goes, thats a normal scenario and things are soon corrected.

Ideas for future improvements

This section contains some ideas that can be added on top of this work to further improve it:

- Listen to changes to the Chassis table in the OVSDB and force a ring re-balance when a Chassis is added or removed from it.
- Cache the ring for a short while to minimize the database reads when the service is under heavy load.
- To greater minimize/avoid event losses it would be possible to cache the last X events to be reprocessed in case a node times out and the ring re-balances.

L3 HA Scheduling of Gateway Chassis

Problem Description

Currently if a single network node is active in the system, gateway chassis for the routers would be scheduled on that node. However, when a new node is added to the system, neither rescheduling nor rebalancing occur automatically. This makes the router created on the first node to be not in HA mode.

Side-effects of this behavior include:

- Skewed up load on different network nodes due to lack of router rescheduling.
- If the active node, where the gateway chassis for a router is scheduled goes down, then because of lack of HA the North-South traffic from that router will be hampered.

Overview of Proposed Approach

Gateway scheduling has been proposed in [2]. However, rebalancing or rescheduling was not a part of that solution. This specification clarifies what is rescheduling and rebalancing. Rescheduling would automatically happen on every event triggered by addition or deletion of chassis. Rebalancing would be only triggered by manual operator action.

Rescheduling of Gateway Chassis

In order to provide proper rescheduling of the gateway ports during addition or deletion of the chassis, following approach can be considered:

- Identify the number of chassis in which each router has been scheduled
  - Consider router for scheduling if no. of chassis < *MAX_GW_CHASSIS*

MAX_GW_CHASSIS is defined in [0]

- Find a list of chassis where router is scheduled and reschedule it up to *MAX_GW_CHASSIS* gateways using list of available candidates. Do not modify the primary chassis association to not interrupt network flows.

Rescheduling is an event triggered operation which will occur whenever a chassis is added or removed. When it happend, `schedule_unhosted_gateways()` [1] will be called to host the unhosted gateways. Routers without gateway ports are excluded in this operation because those are not connected to provider networks and havent the gateway ports. More information about it can be found in the `gateway_chassis` table definition in OVN NorthBound DB [5].

Chassis which has the flag `enable-chassis-as-gw` enabled in their OVN southbound database table, would be the ones eligible for hosting the routers. Rescheduling of router depends on current prorities set. Each chassis is given a specific priority for the routers gateway and priority increases with increasing value (i.e. 1 < 2 < 3). The highest prioritized chassis hosts gateway port. Other chassis are selected as backups.

There are two approaches for rescheduling supported by ovn driver right now: * Least loaded - select least-loaded chassis first, * Random - select chassis randomly.

Few points to consider for the design:

- If there are 2 Chassis C1 and C2, where the routers are already balanced, and a new chassis C3 is added, then routers should be rescheduled only from C1 to C3 and C2 to C3. Rescheduling from C1 to C2 and vice-versa should not be allowed.
- In order to reschedule the routers chassis, the `primary` chassis for a gateway router will be left untouched. However, for the scenario where all routers are scheduled in only one chassis which is available as gateway, the addition of the second gateway chassis would schedule the router gateway ports at a lower priority on the new chassis.

Following scenarios are possible which have been considered in the design:

- **Case #1:**
  - System has only one chassis C1 and all router gateway ports are scheduled on it. We add a new chassis C2.
  - Behavior: All the routers scheduled on C1 will also be scheduled on C2 with priority 1.
- **Case #2:**

  - System has 2 chassis C1 and C2 during installation. C1 goes down.
  - Behavior: In this case, all routers would be rescheduled to C2. Once C1 is back up, routers would be rescheduled on it. However, since C2 is now the new primary, routers on C1 would have lower priority.

- **Case #3:**
  - System has 2 chassis C1 and C2 during installation. C3 is added to it.
  - Behavior: In this case, routers would not move their primary chassis associations. So routers which have their primary on C1, would remain there, and same for routers on C2. However, lower proritized candidates of existing gateways would be scheduled on the chassis C3, depending on the type of used scheduler (Random or LeastLoaded).

Rebalancing of Gateway Chassis

Rebalancing is the second part of the design and it assigns a new primary to already scheduled router gateway ports. Downtime is expected in this operation. Rebalancing of routers can be achieved using external cli script. Similar approach has been implemeneted for DHCP rescheduling [4]. The primary chassis gateway could be moved only to other, previously scheduled gateway. Rebalancing of chassis occurs only if number of scheduled primary chassis ports per each provider network hosted by given chassis is higher than average number of hosted primary gateway ports per chassis per provider network.

This dependency is determined by formula:

avg_gw_per_chassis = num_gw_by_provider_net / num_chassis_with_provider_net

Where:

- avg_gw_per_chassis - average number of scheduler primary gateway chassis withing same provider network.
- num_gw_by_provider_net - number of primary chassis gateways scheduled in given provider networks.
- num_chassis_with_provider_net - number of chassis that has connectivity to given provider network.

The rebalancing occurs only if:

num_gw_by_provider_net_by_chassis > avg_gw_per_chassis

Where:

- num_gw_by_provider_net_by_chassis - number of hosted primary gateways by given provider network by given chassis
- avg_gw_per_chassis - average number of scheduler primary gateway chassis withing same provider network.

Following scenarios are possible which have been considered in the design:

- **Case #1:**
  - System has only two chassis C1 and C2. Chassis host the same number of gateways.
  - Behavior: Rebalancing doesnt occur.
- **Case #2:**

- System has only two chassis C1 and C2. C1 hosts 3 gateways. C2 hosts 2 gateways.
- Behavior: Rebalancing doesnt occur to not continuously move gateways between chassis in loop.

- **Case #3:**
  - System has two chassis C1 and C2. In meantime third chassis C3 has been added to the system.
  - Behavior: Rebalancing should occur. Gateways from C1 and C2 should be moved to C3 up to avg_gw_per_chassis.
- **Case #4:**
  - System has two chassis C1 and C2. C1 is connected to provnet1, but C2 is connected to provnet2.
  - Behavior: Rebalancing shouldnt occur because of lack of chassis within same provider network.

References

ML2/OVN Port forwarding

ML2/OVN supports Port Forwarding (PF) across the North/South data plane. Specific L4 Ports of the Floating IP (FIP) can be directed to a specific FixedIP:PortNumber of a VM, so that different services running in a VM can be isolated, and can communicate with external networks easily.

OVNs native load balancing (LB) feature is used for providing this functionality. An OVN load balancer is expressed in the OVN northbound load_balancer table for all mappings for a given FIP+protocol. All PFs for the same FIP+protocol are kept as Virtual IP (VIP) mappings inside a LB entry. See the diagram below for an example of how that looks like:

```
VIP:PORT = MEMBER1:MPORT1, MEMBER2:MPORT2

The same is extended for port forwarding as:

FIP:PORT = PRIVATE_IP:PRIV_PORT

      Neutron DB                                OVN Northbound DB

+---------------------+              +-----------------------------
↪-----+
| Floating IP AA      |              | Load Balancer AA UDP        ␣
↪      |
|                     |              |                             ␣
↪      |
| +-----------------+ |              |                             ␣
↪      |
| | Port Forwarding | |    +----------->AA:portA => internal␣
↪IP1:portX   |
| |                 | |    |           |                           ␣
↪      |
| | External PortA  +-----+    +-------->AA:portB => internal␣
↪IP2:portX   |
```

(continues on next page)

(continued from previous page)

```
| | Fixed IP1 PortX | |       |      |                           ␣
↪      |
| | Protocol: UDP   | |       |      +----------------------------
↪-----+
| +-----------------+ |       |
|                     |       |      +----------------------------
↪-----+
| +-----------------+ |       |      | Load Balancer AA TCP        ␣
↪      |
| | Port Forwarding | |       |      |                           ␣
↪      |
| |                 | |       |      |                           ␣
↪      |
| | External PortB  +---------+   +--->AA:portC => internal␣
↪IP3:portX  |
| | Fixed IP2 PortX | |          |  |                           ␣
↪      |
| | Protocol: UDP   | |          |  +----------------------------
↪-----+
| +-----------------+ |          |
|                     |          |
| +-----------------+ |          |
| | Port Forwarding | |          |
| |                 | |          |  +----------------------------
↪-----+
| | External PortC  | |          |  | Load Balancer BB TCP        ␣
↪      |
| | Fixed IP3 PortX +--------------+  |                           ␣
↪      |
| | Protocol: TCP   | |             |                           ␣
↪      |
| +-----------------+ |    +----------->BB:portD => internal␣
↪IP4:portX  |
|                     |    |         |                           ␣
↪      |
+---------------------+    |         +----------------------------
↪-----+
                           |
                           |         +-------------------+
                           |         | Logical Router X1 |
+---------------------+    |         |                   |
| Floating IP BB      |    |         | Load Balancers:   |
|                     |    |         | AA UDP, AA TCP    |
| +-----------------+ |    |         +-------------------+
| | Port Forwarding | |    |
| |                 | |    |         +-------------------+
| | External PortD  | |    |         | Logical Router Z1 |
| | Fixed IP4 PortX +------+         |                   |
| | Protocol: TCP   | |              | Load Balancers:   |
| +-----------------+ |              | BB TCP            |
+---------------------+              +-------------------+
```

The OVN LB entries have names that include the id of the FIP and a protocol suffix. That protocol portion is needed because a single FIP can have multiple UDP and TCP port forwarding entries while a given LB entry can either be one or the other protocol (not both). Based on that, the format used to specify an LB entry is:

```
pf-floatingip-<NEUTRON_FIP_ID>-<PROTOCOL>
```

A revision value is present in external_ids of each OVN load balancer entry. That number is synchronized with floating IP entries (NOT the port forwarding!) of the Neutron database.

In order to differentiate a load balancer entry that was created by port forwarding vs load balancer entries maintained by ovn-octavia-provider, the external_ids field also has an owner value:

```
external_ids = {
    ovn_const.OVN_DEVICE_OWNER_EXT_ID_KEY: PORT_FORWARDING_PLUGIN,
    ovn_const.OVN_FIP_EXT_ID_KEY: pf_obj.floatingip_id,
    ovn_const.OVN_ROUTER_NAME_EXT_ID_KEY: rtr_name,
    neutron:revision_number: fip_obj.revision_number,
}
```

The following registry (API) neutron events trigger the OVN backend to map port forwarding into LB:

```
@registry.receives(PORT_FORWARDING_PLUGIN, [events.AFTER_INIT])
def register(self, resource, event, trigger, payload=None):
    registry.subscribe(self._handle_notification, PORT_FORWARDING,␣
↪events.AFTER_CREATE)
    registry.subscribe(self._handle_notification, PORT_FORWARDING,␣
↪events.AFTER_UPDATE)
    registry.subscribe(self._handle_notification, PORT_FORWARDING,␣
↪events.AFTER_DELETE)
```

14.5.2 Module Reference

Todo: Add in all the big modules as automodule indexes.

14.6 OVN Driver

14.6.1 OVN backend

OVN Tools

This document offers details on Neutron tools available for assisting with using the Open Virtual Network (OVN) backend.

Patches and Cherry-picks

Overview

As described in the ovn-migration blueprint, Neutrons OVN ML2 plugin has merged to the Neutron repository as of the Ussuri release. With that, special care must be taken to apply Neutron changes to the proper stable branches of the networking-ovn repo.

Note: These scripts are generic enough to work on any patch file, but particularly handy with the networking-ovn migration.

---

tools/files_in_patch.py

Use this to show files that are changed in a patch file.

```
$ # Make a patch to use as example
$ git show > /tmp/commit.patch

$ ./tools/files_in_patch.py /tmp/commit.patch | grep .py
tools/download_gerrit_change.py
tools/files_in_patch.py
tools/migrate_names.py
```

tools/download_gerrit_change.py

This tool is needed by `migrate_names.py` (see below), but it can be used independently. Given a Gerrit change id, it will fetch the latest patchset of the change from review.opendev.org as a patch file. The output can be stdout or an optional filename.

```
$ ./tools/download_gerrit_change.py --help
Usage: download_gerrit_change.py [OPTIONS] GERRIT_CHANGE

Options:
  -o, --output_patch TEXT  Output patch file. Default: stdout
  -g, --gerrit_url TEXT    The url to Gerrit server  [default:
                           https://review.opendev.org/]
  -t, --timeout INTEGER    Timeout, in seconds  [default: 10]
  --help                   Show this message and exit.

$ ./tools/download_gerrit_change.py 698863 -o /tmp/change.patch
$ ./tools/files_in_patch.py /tmp/change.patch
networking_ovn/ml2/mech_driver.py
networking_ovn/ml2/trunk_driver.py
networking_ovn/tests/unit/ml2/test_mech_driver.py
networking_ovn/tests/unit/ml2/test_trunk_driver.py
```

tools/migrate_names.py

Use this tool to modify the name of the files in a patchfile so it can be converted to/from the legacy networking-ovn and Neutron repositories.

The mapping of how the files are renamed is based on `migrate_names.txt`, which is located in the same directory where `migrate_names.py` is installed. That behavior can be modified via the `--mapfile` option. More information on how the map is parsed is provided in the header section of that file.

```
$ ./tools/migrate_names.py --help
Usage: migrate_names.py [OPTIONS]

Options:
  -i, --input_patch TEXT    input_patch patch file or gerrit change
  -o, --output_patch TEXT   Output patch file. Default: stdout
  -m, --mapfile PATH        Data file that specifies mapping to be applied␣
↪to
                            input  [default: /home/user/openstack/neutron.
↪git
                            /tools/migrate_names.txt]
  --reverse / --no-reverse  Map filenames from networking-ovn to Neutron␣
↪repo
  --help                    Show this message and exit.
$ ./tools/migrate_names.py -i 701646 > /tmp/ovn_change.patch
$ ./tools/migrate_names.py -o /tmp/reverse.patch -i /tmp/ovn_change.patch -
↪-reverse
$ diff /tmp/reverse.patch /tmp/ovn_change.patch | grep .py
< --- a/neutron/plugins/ml2/drivers/ovn/mech_driver/mech_driver.py
< +++ b/neutron/plugins/ml2/drivers/ovn/mech_driver/mech_driver.py
> --- a/networking_ovn/ml2/mech_driver.py
> +++ b/networking_ovn/ml2/mech_driver.py
<... snip ...>

$ ./tools/files_in_patch.py /tmp/ovn_change.patch
networking_ovn/ml2/mech_driver.py
networking_ovn/ml2/trunk_driver.py
networking_ovn/tests/unit/ml2/test_mech_driver.py
networking_ovn/tests/unit/ml2/test_trunk_driver.py
```

14.7 Dashboards

There is a collection of dashboards to help developers and reviewers located here.

14.7.1 CI Status Dashboards

Gerrit Dashboards

- Neutron priority reviews
- Neutron master branch reviews
- Neutron subproject reviews (master branch)
- Neutron stable branch reviews
- Neutron Infra reviews

These dashboard links can be generated by Gerrit Dashboard Creator. Useful dashboard definitions are found in `dashboards` directory.

Grafana Dashboards

Look for neutron and networking-* dashboard by names by going to the following link:

Grafana

For instance:

- Neutron
- Neutron-lib