

OpenStack Networking

Administration Guide

Grizzly, 2013.1 (Jul 2, 2013)



OpenStack Networking Administration Guide

Grizzly, 2013.1 (2013-07-02)

Copyright © 2011-2013 OpenStack Foundation All rights reserved.

This document is for administrators who run the OpenStack Networking Virtual Network Service.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

Preface	6
Intended Audience	6
Document Change History	7
Resources	7
1. Overview	1
What Is OpenStack Networking?	1
OpenStack Networking Architecture	4
OpenStack Networking Deployment Use Cases	7
2. OpenStack Networking Installation	11
Install Packages (Ubuntu)	11
Installing Packages (Fedora)	20
3. Required Configuration for OpenStack Identity & Compute	23
OpenStack Identity	23
OpenStack Compute	24
4. Using OpenStack Networking	29
Core OpenStack Networking API Features	29
Using OpenStack Compute with OpenStack Networking	32
5. Under the Hood	35
Open vSwitch	35
Linux bridge	45
6. Advanced Features through API Extensions	54
Provider Networks	54
L3 Routing and NAT	56
Security Groups	59
7. Advanced Configuration Options	62
OpenStack Networking Server with Plugin	62
DHCP Agent	63
L3 Agent	64
8. Authentication and Authorization	68
9. Advanced Operational Features	72
Logging Settings	72
Notifications	72
Quotas	74
10. High Availability	79
OpenStack Networking High Availability with Pacemaker	79
11. Limitations	80
A. Demos Setup	82
Single Flat Network	82
Provider Router with Private Networks	88
Per-tenant Routers with Private Networks	96
Scalable and Highly Available DHCP Agents	110
B. Core Configuration File Options	120
quantum.conf	120
ovs_quantum_plugin.ini	124
linuxbridge_conf.ini	126
dhcp_agent.ini	127
l3_agent.ini	128
Common Device Manager Options	128

C. Plugin pagination and sorting support	130
--	-----

List of Tables

1.1. Plugin Compatibility with OpenStack Compute Drivers	3
4.1. Network	30
4.2. Subnet	30
4.3. Port	30
6.1. Provider Network Attributes	55
6.2. Router	57
6.3. Floating IP	57
6.4. Security Group Attributes	60
6.5. Security Group Rules	60
7.1. Settings	63
7.2. Basic settings	63
7.3. Basic settings	64
A.1. Nodes for Demo	88
A.2. Hosts for Demo	111
B.1. Debugging Options	120
B.2. Logging Options	120
B.3. Authentication Options	121
B.4. Service Options	121
B.5. Base Plugin Options	121
B.6. Common RPC Message Options	122
B.7. Rabbit RPC Options	123
B.8. QPID RPC Options	123
B.9. Notification Options	124
B.10. Quota Options	124
B.11. Database Access by Plugin	124
B.12. OVS Options Common to Plugin and Agent	125
B.13. Agent Options	126
B.14. Database Access by Plugin	126
B.15. VLAN Configurations	126
B.16. Networking Options on the Agent	126
B.17. Agent Options	127
B.18. DHCP-Specific Options	127
B.19. dnsmasq Options	127
B.20. L3 Specific Options	128
B.21. Device Manager Options	128
C.1. The plugins are supporting native pagination and sorting	130

Preface

Intended Audience	6
Document Change History	7
Resources	7

OpenStack Networking was created to provide a rich and tenant-facing API for defining network connectivity and addressing in the cloud. The project, code-named "quantum," gives operators the ability to leverage different networking technologies to power their cloud networking.

Intended Audience

This guide assists OpenStack administrators in leveraging different networking technologies to power their cloud networking. This document covers how to install, configure, and run OpenStack Networking.

The user should also have access to a plugin providing the implementation of the OpenStack Networking service. Several plugins are included in the OpenStack Networking distribution:

- **Openvswitch**. Implements OpenStack Networking with Open vSwitch for the KVM and XenServer compute platforms.
- **Cisco**. Implements OpenStack Networking for deployments by using Cisco UCS blades and Nexus switches for KVM compute platforms.
- **Linux Bridge**. Implements OpenStack Networking with the Linux Bridge for the KVM compute platforms.
- **Nicira NVP**. Implements OpenStack Networking by using the Nicira Network Virtualization Platform (NVP) for KVM and XenServer compute platforms.
- **Ryu**. Implements OpenStack Networking by using the Ryu OpenFlow Controller for KVM compute platforms.
- **NEC OpenFlow**. Implements OpenStack Networking by using Trema Sliceable Switch OpenFlow Controller or NEC ProgrammableFlow Controller for KVM compute platforms.
- **Big Switch, Floodlight REST Proxy**. Implements the OpenStack Networking plugin API and realizes the network abstractions by leveraging the Big Network Controller (BNC) or Floodlight OpenFlow Controller for KVM hypervisor based compute platforms.
- **PLUMgrid**. Implements OpenStack Networking by using the PLUMgrid controller that offers on-demand enterprise-grade Virtual Network Infrastructure (VNI) for both Public and Private cloud data centers.

Plugins can also be distributed separately from OpenStack Networking.

You should also be familiar with running the OpenStack Compute service as described in the operator's documentation.

Document Change History

The most recent changes are described in the table below:

Revision Date	Summary of Changes
Jun 5, 2013	• Added Chapter 5, "Under the Hood" [35] .
Jun 1, 2013	• Updated title for consistency.
Apr 30, 2013	• Grizzly release.
Mar 24, 2013	• Added auth_token parameters to the quantum.conf file description.
Feb 4, 2013	• Use OpenStack naming conventions through out document.
Dec 20, 2012	• Trunk designation of this document.
Nov 9, 2012	• Folsom release of this document.
Sep 18, 2012	• First edition of this document.

Resources

For more information on OpenStack Networking and the other network-related projects, see the project page on the OpenStack wiki (wiki.openstack.org/Quantum).

For information about programming against the OpenStack Networking API, see the [OpenStack Networking API Guide \(v2.0\)](#).

We welcome feedback, comments, and bug reports at bugs.launchpad.net/Quantum.

1. Overview

What Is OpenStack Networking?	1
OpenStack Networking Architecture	4
OpenStack Networking Deployment Use Cases	7

This chapter describes the high-level concepts and components of an OpenStack Networking deployment.

What Is OpenStack Networking?

The OpenStack Networking project was created to provide a rich tenant-facing API for defining network connectivity and addressing in the cloud. The OpenStack Networking project gives operators the ability to leverage different networking technologies to power their cloud networking.

For a detailed description of the OpenStack Networking API abstractions and their attributes, see the [OpenStack Networking API Guide \(v2.0\)](#).

OpenStack Networking API: Rich Control over Network Functionality

OpenStack Networking is a virtual network service that provides a powerful API to define the network connectivity and addressing used by devices from other services, such as OpenStack Compute.

The OpenStack Compute API has a virtual server abstraction to describe compute resources. Similarly, the OpenStack Networking API has virtual network, subnet, and port abstractions to describe network resources. In more detail:

- **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.

You can configure rich network topologies by creating and configuring networks and subnets, and then instructing other OpenStack services like OpenStack Compute to attach virtual devices to ports on these networks. In particular, OpenStack Networking supports each tenant having multiple private networks, and allows tenants to choose their own IP addressing scheme, even if those IP addresses overlap with those used by other tenants.

This enables very advanced cloud networking use cases, such as building multi-tiered web applications and allowing applications to be migrated to the cloud without changing IP addresses.

Even if a cloud administrator does not intend to expose these capabilities to tenants directly, the OpenStack Networking API can be very useful even when used as an admin API

because it provides significantly more flexibility for the cloud administrator to customize network offerings.

The OpenStack Networking API also provides a mechanism that lets cloud administrators expose additional API capabilities through API extensions. Commonly, new capabilities are first introduced as an API extension, and over time become part of the core OpenStack Networking API.

Plugin Architecture: Flexibility to Choose Different Network Technologies

Enhancing traditional networking solutions to provide rich cloud networking is challenging. Traditional networking is not designed to scale to cloud proportions or to configure automatically.

The original OpenStack Compute network implementation assumed a very basic model of **performing all isolation through Linux VLANs and IP tables**. OpenStack Networking introduces the concept of a *plugin*, which is a pluggable back-end implementation of the OpenStack Networking API. A plugin can use a variety of technologies to implement the logical API requests. Some OpenStack Networking plugins might use basic Linux VLANs and IP tables, while others might use more advanced technologies, such as L2-in-L3 tunneling or OpenFlow, to provide similar benefits.

The current set of plugins include:

- **Open vSwitch**. Documentation included in this guide.
- **Cisco**. Documented externally at: <http://wiki.openstack.org/cisco-quantum>
- **Linux Bridge**. Documentation included in this guide and <http://wiki.openstack.org/Quantum-Linux-Bridge-Plugin>
- **Nicira NVP**. Documentation include in this guide, [NVP Product Overview](#) , and [NVP Product Support](#).
- **Ryu**. <https://github.com/osrg/ryu/wiki/OpenStack>
- **NEC OpenFlow**. <http://wiki.openstack.org/Quantum-NEC-OpenFlow-Plugin>
- **Big Switch, Floodlight REST Proxy**. <http://www.openflowhub.org/display/floodlightcontroller/Quantum+REST+Proxy+Plugin>
- **PLUMgrid** <https://wiki.openstack.org/wiki/Plumgrid-quantum>
- **Hyper-V Plugin**.
- **Brocade Plugin**.
- **Midonet Plugin**.

Plugins can have different properties in terms of hardware requirements, features, performance, scale, operator tools, etc. Supporting many plugins enables the cloud administrator to weigh different options and decide which networking technology is right for the deployment.

Not all OpenStack networking plugins are compatible with all possible OpenStack compute drivers:

Table 1.1. Plugin Compatibility with OpenStack Compute Drivers

	Libvirt (KVM/ QEMU)	XenServer	VMware	Hyper-V	Bare-metal	PowerVM
Open vSwitch	Yes					

	Libvirt (KVM/ QEMU)	XenServer	VMware	Hyper-V	Bare-metal	PowerVM
Linux Bridge	Yes					
Cisco	Yes					
Nicira NVP	Yes	Yes	Yes			
Ryu	Yes					
NEC	Yes					
Bigswitch / Floodlight	Yes					
Hyper-V				Yes		
Brocade	Yes					
Midonet	Yes					
Plumgrid	Yes					

OpenStack Networking Architecture

This section describes the high-level components of an OpenStack Networking deployment.

Overview

Before you deploy OpenStack Networking, it is useful to understand the different components that make up the solution and how those components interact with each other and with other OpenStack services.

OpenStack Networking is a standalone service, just like other OpenStack services such as OpenStack Compute, OpenStack Image service, OpenStack Identity service, and the OpenStack Dashboard. Like those services, a deployment of OpenStack Networking often involves deploying several processes on a variety of hosts.

The main process of the OpenStack Networking server is `quantum-server`, which is a Python daemon that exposes the OpenStack Networking API and passes user requests to the configured OpenStack Networking plugin for additional processing. Typically, the plugin requires access to a database for persistent storage, similar to other OpenStack services.

If your deployment uses a *controller host* to run centralized OpenStack Compute components, you can deploy the OpenStack Networking server on that same host. However, OpenStack Networking is entirely standalone and can be deployed on its own server as well. OpenStack Networking also includes additional agents that might be required depending on your deployment:

- **plugin agent** (`quantum-*agent`). Runs on each hypervisor to perform local `vswitch` configuration. Agent to be run depends on which plugin you are using, as some plugins do not require an agent.
- **dhcp agent** (`quantum-dhcp-agent`). Provides `DHCP services` to tenant networks. This agent is the same across all plugins.
- **l3 agent** (`quantum-l3-agent`). Provides `L3/NAT forwarding` to provide external network access for VMs on tenant networks. This agent is the same across all plugins.

These agents interact with the main quantum-server process in the following ways:

- Through **RPC**. For example, **rabbitmq** or **qpidd**.
- Through the **standard OpenStack Networking API**.

OpenStack Networking relies on the OpenStack **Identity Project (Keystone)** for authentication and authorization of all API request.

OpenStack Compute interacts with OpenStack Networking through calls to its standard API. As part of creating a VM, nova-compute communicates with the OpenStack Networking API to plug each virtual NIC on the VM into a particular network.

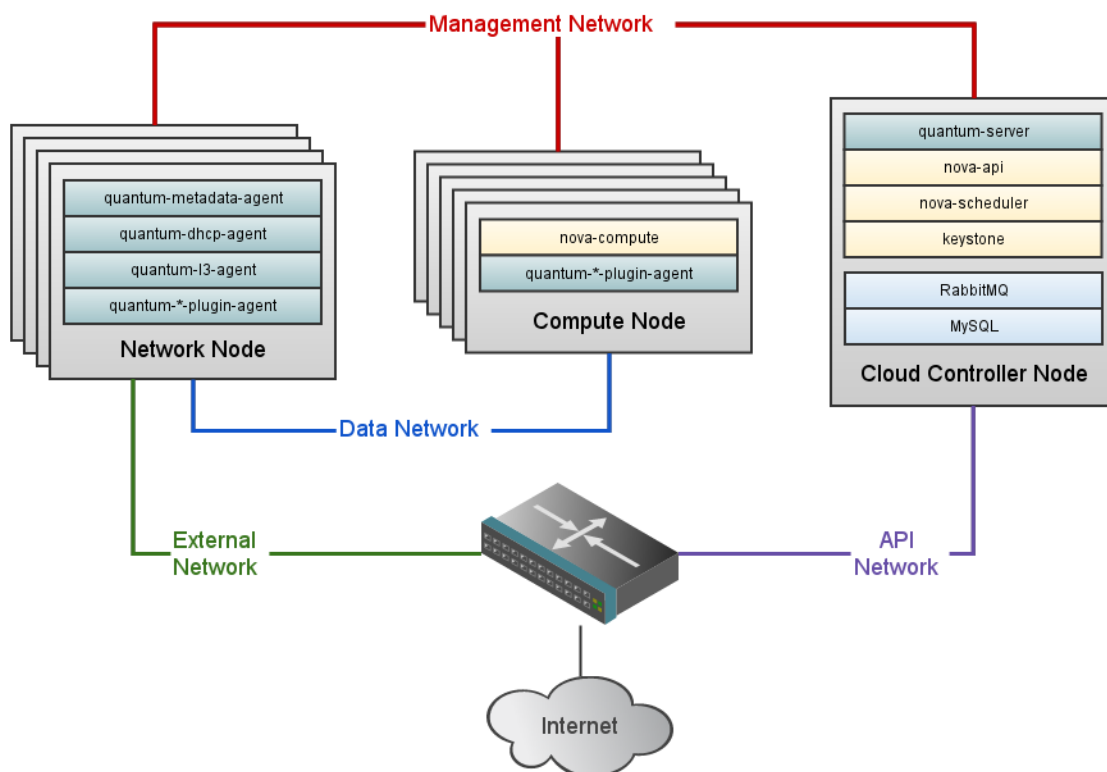
The OpenStack Dashboard (Horizon) has integration with the OpenStack Networking API, allowing administrators and tenant users, to create and manage network services through the Horizon GUI.

Place Services on Physical Hosts

Like other OpenStack services, OpenStack Networking provides cloud administrators with significant flexibility in deciding which individual services should run on which physical devices. On one extreme, all service daemons can be run on a single physical host for evaluation purposes. On the other, each service could have its own physical hosts, and some cases be replicated across multiple hosts for redundancy. See ???

In this guide, we focus primarily on a standard architecture that includes a “**cloud controller**” host, a “**network gateway**” host, and a set of hypervisors for running VMs. The “cloud controller” and “network gateway” can be combined in simple deployments, though if you expect VMs to send significant amounts of traffic to or from the Internet, a dedicated network gateway host is suggested to avoid potential **CPU contention between packet forwarding performed by the quantum-l3-agent and other OpenStack services**.

Network Connectivity for Physical Hosts



A standard OpenStack Networking setup has up to four distinct physical data center networks:

- **Management network.** Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center.
- **Data network.** Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the **OpenStack Networking plugin** in use.
- **External network.** Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet.
- **API network.** Exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet. This may be the same network as the external network, as it is possible to create a subnet for the external network that uses IP allocation ranges to use only less than the full range of IP addresses in an IP block.



Note

If the OpenStack Compute metadata service is being used, any address space used on tenant networks must be route-able on both the API network and on

the External network, since the host running nova-api must be able to reply to HTTP requests with the un-SNATED IP address of a VM.

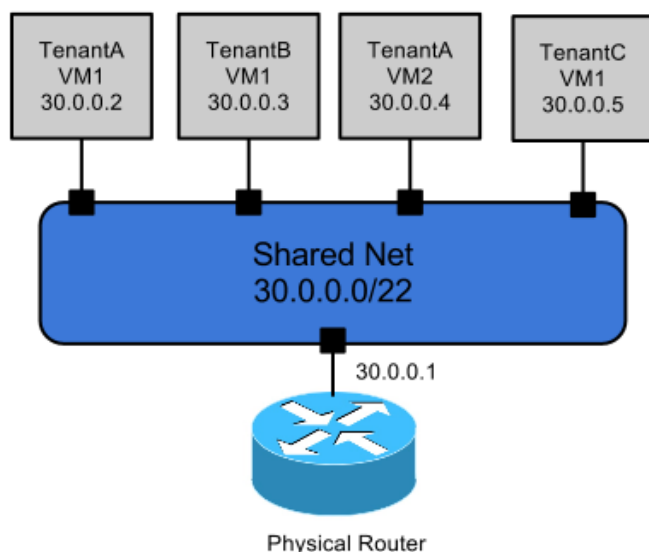
OpenStack Networking Deployment Use Cases

This section describes some of the common use cases for OpenStack Networking. These examples are not exhaustive and can be combined with each other to create more complex use cases.

Use Case: Single Flat Network

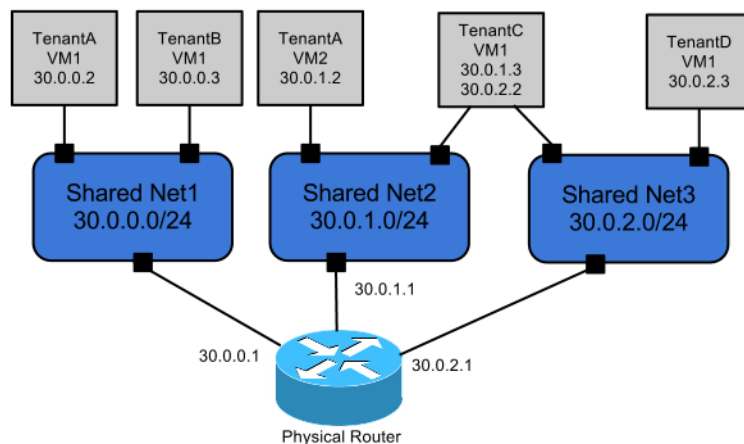
In the simplest use case, a single OpenStack Networking network exists. This is a "shared" network, meaning it is visible to all tenants via the OpenStack Networking API. Tenant VMs have a single NIC, and receive a fixed IP address from the subnet(s) associated with that network. This essentially maps to the **FlatManager** and **FlatDHCPManager** models provided by OpenStack Compute. Floating IPs are not supported.

It is common that such an OpenStack Networking network is a "provider network", meaning it was created by the OpenStack administrator to map directly to an existing physical network in the data center. This allows the provider to use a physical router on that data center network as the gateway for VMs to reach the outside world. For each subnet on an external network, the gateway configuration on the physical router must be manually configured outside of OpenStack.



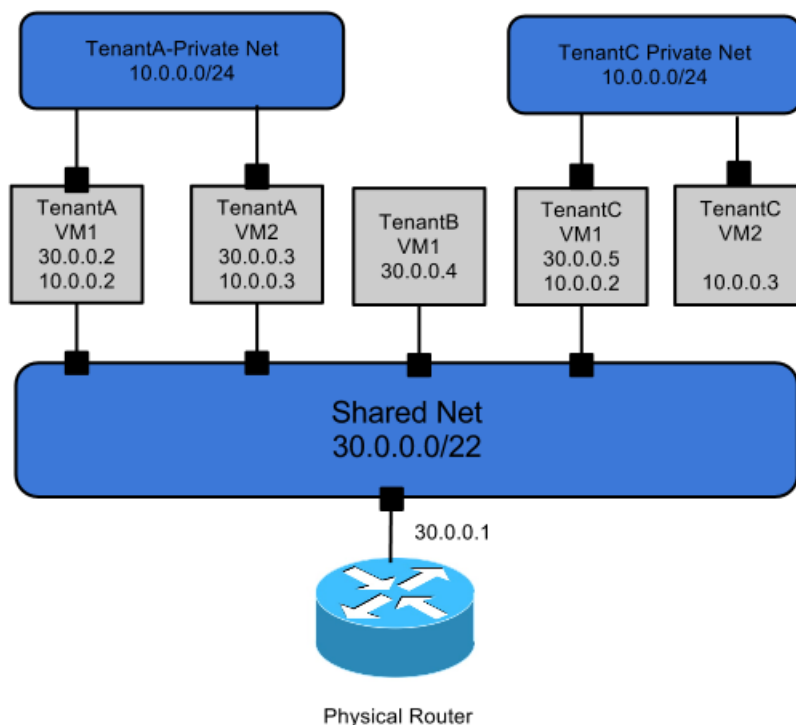
Use Case: Multiple Flat Network

This use case is very similar to the above Single Flat Network use case, except that tenants see multiple shared networks via the OpenStack Networking API and can choose which network (or networks) to plug into.



Use Case: Mixed Flat and Private Network

This use case is an extension of the above flat network use cases, in which tenants also optionally have access to private per-tenant networks. In addition to seeing one or more shared networks via the OpenStack Networking API, tenants can create additional networks that are only visible to users of that tenant. When creating VMs, those VMs can have NICs on any of the shared networks and/or any of the private networks belonging to the tenant. This enables the creation of "multi-tier" topologies using VMs with multiple NICs. It also supports a model where a VM acting as a gateway can provide services such as routing, NAT, or load balancing.

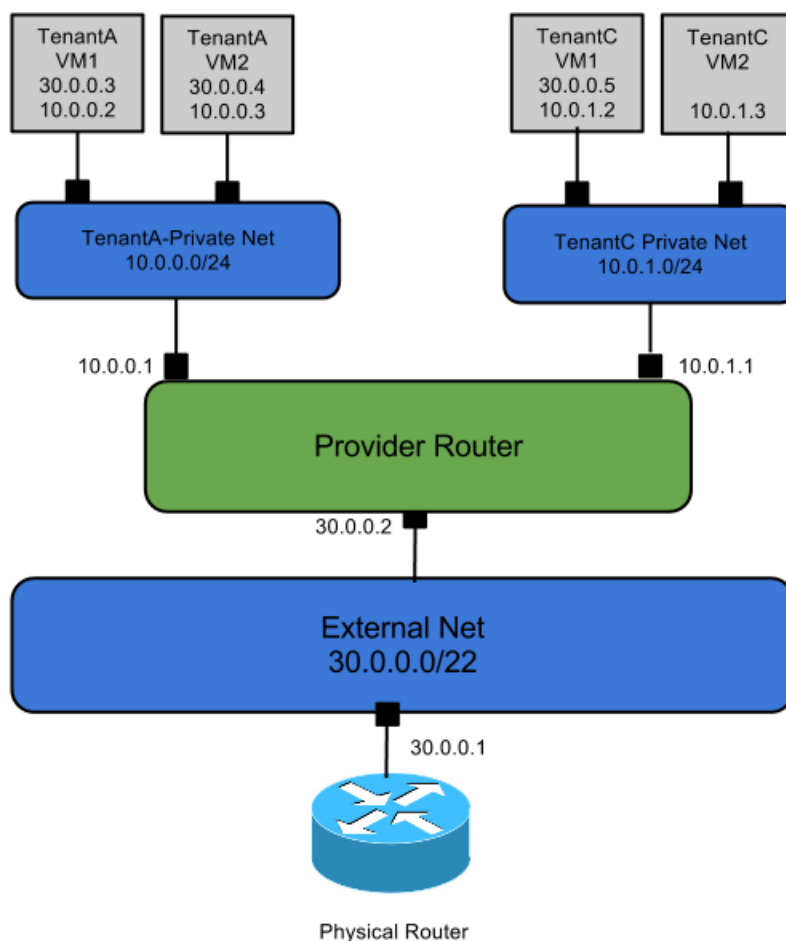


Use Case: Provider Router with Private Networks

This use provides each tenant with one or more private networks, which connect to the outside world via an OpenStack Networking router. The case where each tenant gets exactly one network in this form maps to the same logical topology as the **VlanManager** in OpenStack Compute (of course, OpenStack Networking doesn't require VLANs). Using the OpenStack Networking API, the tenant would only see a network for each private network assigned to that tenant. The router object in the API is created and owned by the cloud admin.

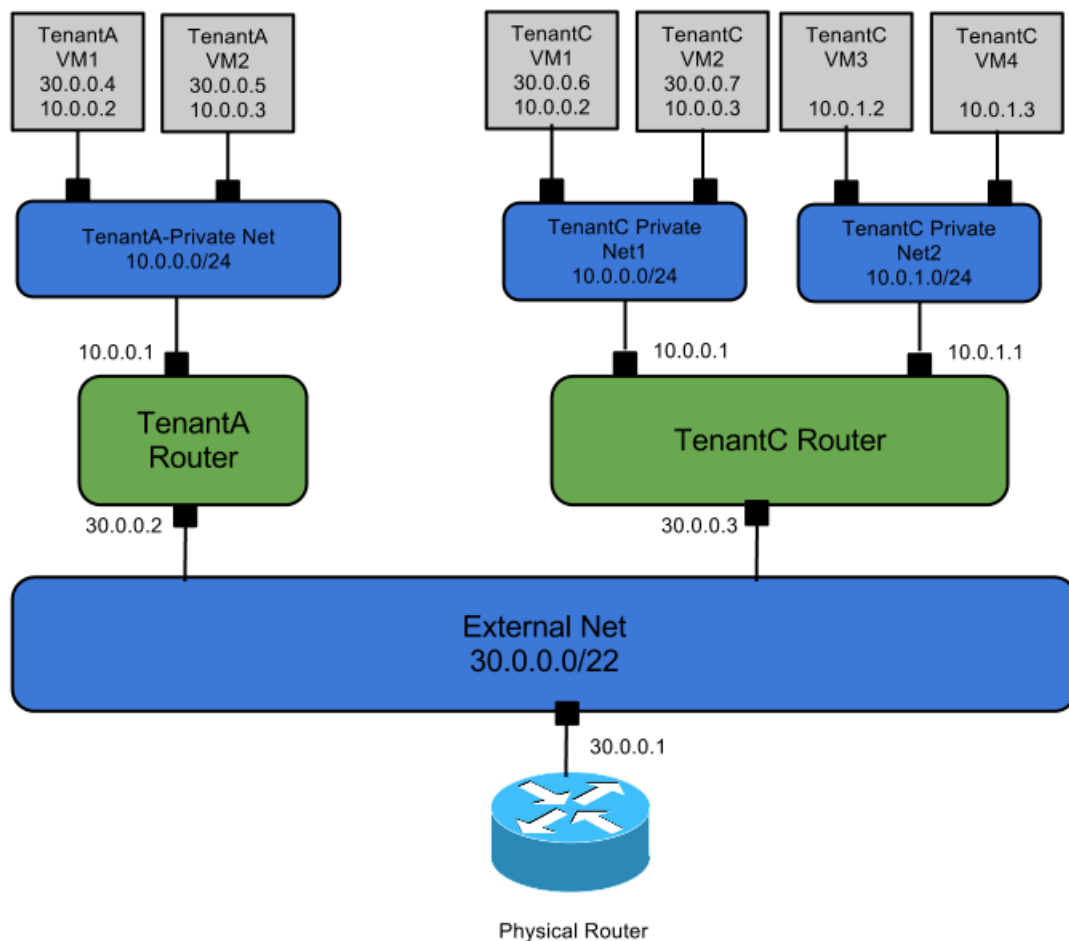
This model supports giving VMs public addresses using "floating IPs", in which the router maps public addresses from the external network to fixed IPs on private networks. Hosts without floating IPs can still create outbound connections to the external network, as the provider router performs SNAT to the router's external IP. The IP address of the physical router is used as the gateway_ip of the external network subnet, so the provider has a default router for Internet traffic.

The router provides L3 connectivity between private networks, meaning that different tenants can reach each others instances unless additional filtering (e.g., security groups) is used. Because there is only a single router, tenant networks cannot use overlapping IPs. Thus, it is likely that the admin would create the private networks on behalf of tenants.



Use Case: Per-tenant Routers with Private Networks

A more advanced router scenario in which each tenant gets at least one router, and potentially has access to the OpenStack Networking API to create additional routers. The tenant can create their own networks, potentially uplinking those networks to a router. This model enables tenant-defined multi-tier applications, with each tier being a separate network behind the router. Since there are multiple routers, tenant subnets can be overlapping without conflicting, since access to external networks all happens via **SNAT** or **Floating IPs**. Each router uplink and floating IP is allocated from the external network subnet.



2. OpenStack Networking Installation

Install Packages (Ubuntu)	11
Installing Packages (Fedora)	20

This chapter describes how to install the OpenStack Networking service and get it up and running.

If you are building a host from scratch to use for OpenStack Networking, we strongly recommend using Ubuntu 12.04/12.10 or Fedora 17/18 as these platforms have OpenStack Networking packages and receive significant testing.

OpenStack Networking requires at least dnsmasq 2.59, to support all the options it requires.

Install Packages (Ubuntu)



Note

We are using Ubuntu cloud archive you can read more about it Explanation of each possible sources.list entry can be found here: <http://bit.ly/Q8OJ9M>

Point to Grizzly PPAs:

```
# echo deb http://ubuntu-cloud.archive.canonical.com/ubuntu precise-updates/
grizzly main >>/etc/apt/sources.list.d/grizzly.list
# apt-get install ubuntu-cloud-keyring
# apt-get update
# apt-get upgrade
```



Note

Please use "sudo" in order to install and configure packages with superuser privileges.

Install quantum-server

Install quantum-server and CLI for accessing the API:

```
apt-get -y install quantum-server python-quantumclient
```

You will also want to install the plugin you choose to use, for example:

```
apt-get -y install quantum-plugin-<plugin-name>
```

Most plugins require a database to be installed and configured in a plugin configuration file. For example:

```
apt-get -y install mysql-server python-mysqldb python-sqlalchemy
```

A database that you are already using for other OpenStack services will work fine for this. Simply create a 'quantum' database:

```
mysql -u <user> -p <pass> -e "create database quantum"
```

And then configure the plugin's configuration file to use this database. Find the plugin configuration file in `/etc/quantum/plugins/<plugin-name>` (For example, `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini`) and set:

```
sql_connection = mysql://<user>:<password>@localhost/quantum?charset=utf8
```

RPC Setup

Many OpenStack Networking plugins uses RPC to allow agents to communicate with the main quantum-server process. If your plugin requires agents, this can use the same RPC mechanism used by other OpenStack components like Nova.

To use RabbitMQ as the message bus for RPC, make sure that rabbit is installed on a host reachable via the management network (if this is already the case because of deploying another service like Nova, this existing RabbitMQ setup is sufficient):

```
apt-get install rabbitmq-server
rabbitmqctl change_password guest <password>
```

Then update `/etc/quantum/quantum.conf` with these values:

```
rabbit_host=<mgmt-IP-of-rabbit-host>
rabbit_password=<password>

rabbit_userid=guest
```



Important

This `/etc/quantum/quantum.conf` file should be copied to and used on all hosts running quantum-server or any quantum-* agent binaries.

Plugin Configuration: OVS Plugin

Using the Open vSwitch (OVS) plugin in a deployment with multiple hosts requires the using of either tunneling or vlans in order to isolate traffic from multiple networks.

Tunneling is easier to deploy, as it does not require configuring VLANs on network switches, so that is what we describe here. More advanced deployment options are described in the ???

Edit `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini` to specify the following values:

```
enable_tunneling=True
tenant_network_type=gre
tunnel_id_ranges=1:1000
# only required for nodes running agents
local_ip=<data-net-IP-address-of-node>
```

After performing that change on the node running quantum-server, restart quantum-server to pick up the new settings.

```
service quantum-server restart
```

Plugin Configuration: Nicira NVP Plugin

Make sure the NVP plugin is installed using:

```
apt-get -y install quantum-plugin-nicira
```

To configure OpenStack Networking to use the NVP plugin first edit `/etc/quantum/quantum.conf` and set:

```
core_plugin = quantum.plugins.nicira.nicira_nvp_plugin.QuantumPlugin.  
NvpPluginV2
```

Edit `/etc/quantum/plugins/nicira/nvp.ini` in order to configure the plugin.

In the [DATABASE] section, specify the quantum database created in the previous step using the following line, substituting your database server IP address for localhost if the database is not local:

```
sql_connection = mysql://<user>:<password>@localhost/quantum?charset=utf8
```

In order to tell OpenStack Networking about a controller cluster, create a new [CLUSTER:<name>] section in the config file, and add the following entries:

The UUID of the NVP Transport Zone that should be used by default when a tenant creates a network. This value can be retrieved from the NVP Manager Transport Zones page:

```
default_tz_uuid = <uuid_of_the_transport_zone>
```

A connection string indicating parameters to be used by the NVP plugin when connecting to the NVP webservice API. There will be one of these lines in the config file for each NVP controller in your deployment. An NVP operator will likely want to update the NVP controller IP and password, but the remaining fields can be the defaults:

```
nvp_controller_connection =  
<controller_node_ip>:<controller_port>:<api_user>:<api_password>:<request_timeout>:<http_t
```

The UUID of an NVP L3 Gateway Service that should be used by default when a tenant creates a router. This value can be retrieved from the NVP Manager Gateway Services page:

```
default_l3_gw_service_uuid = <uuid_of_the_gateway_service>
```



Warning

Ubuntu packaging currently does not update the quantum init script to point to the NVP config file. Instead, manually update `/etc/default/quantum-server` to set:

```
QUANTUM_PLUGIN_CONFIG = /etc/quantum/plugins/nicira/nvp.ini
```

Lastly, restart quantum-server to pick up the new settings.

```
service quantum-server restart
```

An example quantum.conf file to use with NVP would be:

```
core_plugin = quantum.plugins.nicira.nicira_nvp_plugin.QuantumPlugin.  
NvpPluginV2  
rabbit_host = 192.168.203.10  
allow_overlapping_ips = True
```

An example nvp.ini file to use with NVP would be:

```
[DATABASE]
sql_connection=mysql://root:root@127.0.0.1/quantum

[CLUSTER:main]
default_tz_uuid = d3afb164-b263-4aaa-a3e4-48e0e09bb33c
default_l3_gw_service_uuid=5c8622cc-240a-40a1-9693-e6a5fca4e3cf
nvp_controller_connection=10.0.0.2:443:admin:admin:30:10:2:2
nvp_controller_connection=10.0.0.3:443:admin:admin:30:10:2:2
nvp_controller_connection=10.0.0.4:443:admin:admin:30:10:2:2
```

Configuring Big Switch, Floodlight REST Proxy Plugin

To configure OpenStack Networking to use the REST Proxy plugin first edit `/etc/quantum/quantum.conf` and set:

```
core_plugin = quantum.plugins.bigswitch.plugin.QuantumRestProxyV2
```

Edit `/etc/quantum/plugins/bigswitch/restproxy.ini` in order to configure the plugin. The quantum database created previously will be used by setting:

```
sql_connection = mysql://<user>:<password>@localhost/restproxy_quantum?
charset=utf8
```

Specify a comma separated list controller_ip:port pairs:

```
server = <controller-ip>:<port>
```

Lastly, restart quantum-server to pick up the new settings.

```
service quantum-server restart
```

Configuring Ryu Plugin

Make sure the ryu plugin is installed using:

```
apt-get -y install quantum-plugin-ryu
```

To configure OpenStack Networking to use the Ryu plugin first edit `/etc/quantum/quantum.conf` and set:

```
core_plugin = quantum.plugins.ryu.ryu_quantum_plugin.RyuQuantumPluginV2
```

Edit `/etc/quantum/plugins/ryu/ryu.ini` in order to configure the plugin. In the [DATABASE] section, specify the quantum database created in the previous step using the following line, substituting your database server user/password/IP address/port based on your setting:

```
sql_connection = mysql://<user>:<password>@<ip-address>:<port>/quantum?
charset=utf8
```

In [OVS] section, set the necessary values for ryu-quantum-agent. `openflow_rest_api` is used to tell where Ryu is listening for REST API. Substitute ip-address and port-no based on your ryu setup. `ovsdb_interface` is used for Ryu to access ovsdb-server. Substitute `eth0` based on your setup. IP address is derived from the interface name. If you want to change those value irrelevant to the interface name, `ovsdb_ip` can be specified. If you use non-default port for ovsdb-server, it can be specified by `ovsdb_port`. `tunnel_interface` needs to be set to tell what IP address is used for tunneling. (If tunneling isn't used, this value will be ignored.) The IP address is derived from the network interface name. The same configuration file

can be used for many compute-node by using network interface name with different IP address.

```
openflow_rest_api = <ip-address>:<port-no>
ovsdb_interface = <eth0>
tunnel_interface = <eth0>
```

Lastly, restart quantum-server to pick up the new settings.

```
service quantum-server restart
```

Configuring PLUMgrid Plugin

To configure OpenStack Networking to use the PLUMgrid plugin first edit `/etc/quantum/quantum.conf` and set:

```
core_plugin = quantum.plugins.plumgrid.plumgrid_nos_plugin.plumgrid_plugin.
QuantumPluginPLUMgridV2
```

Edit `/etc/quantum/plugins/plumgrid/plumgrid.ini` in order to configure the plugin. The quantum database created previously will be used by setting:

```
sql_connection = mysql://<user>:<password>@localhost/plumgrid_quantum?charset=
utf8
```

Under the [PLUMgridNOS] section specify the IP address of the PLUMgrid director also known as NOS. Behind the director information admin username and password are also required:

```
servers=<plumgrid_NOS_IP>
username=<username>
password=<password>
```

Lastly, restart quantum-server to pick up the new settings.

```
service quantum-server restart
```

Install Software on Data Forwarding Nodes

Plugins commonly have requirements for particular software that must be run on each node that handles data packets. This includes any node running nova-compute, as well as nodes running dedicated OpenStack Networking service agents like quantum-dhcp-agent, quantum-l3-agent, quantum-lbaas-agent, etc (see below for more information about individual services agents).

Commonly, any data forwarding node should have a network interface with an IP address on the "management network" and another interface on the "data network".

In this section, we describe the requirements for particular plugins, which may include the installation of switching software (e.g., Open vSwitch) as well as agents used to communicate with the quantum-server process running elsewhere in the data center.

Node Setup: OVS Plugin

The Open vSwitch plugin requires Open vSwitch as well as the quantum-plugin-openvswitch-agent agent to be installed on each Data Forwarding Node.

Install the OVS agent package, will pull in the Open vSwitch software as a dependency:

```
apt-get -y install quantum-plugin-openvswitch-agent
```

The `ovs_quantum_plugin.ini` created in the above step must be replicated on all nodes `quantum-plugin-openvswitch-agent`. When using tunneling, each node running `quantum-plugin-openvswitch` agent should have an IP address configured on the Data Network, and that IP address should be specified using the `local_ip` value in the `ovs_quantum_plugin.ini` file.

Then restart Open vSwitch to properly load the kernel module:

```
service openvswitch-switch restart
```

And restart the agent:

```
service quantum-plugin-openvswitch-agent restart
```

All hosts running `quantum-plugin-openvswitch-agent` also requires that an OVS bridge named "br-int" exists. To create it, run:

```
ovs-vsctl add-br br-int
```

Node Setup: Nicira NVP Plugin

The Nicira NVP plugin requires a version of Open vSwitch to be installed on each data forwarding node, but does not require an additional agent on data forwarding nodes.



Warning

It is critical that you are running a version of Open vSwitch that is compatible with the current version of the NVP Controller software. Do not use the version of Open vSwitch installed by default on Ubuntu. Instead, use the version of Open Vswitch provided on the Nicira support portal for your version of the NVP Controller.

Each data forwarding node should have an IP address on the "management network", as well as an IP address on the "data network" used for tunneling data traffic.

For full details on configuring your forwarding node, please see the NVP Administrator Guide. Next, use the same guide to add the node as a "Hypervisor" using the NVP Manager GUI (Note: even if your forwarding node has no VMs and is only used for services agents like `quantum-dhcp-agent` or `quantum-lbaas-agent`, it should be added to NVP as a Hypervisor).

After following the NVP Administrator Guide, use the page for this Hypervisor in the NVP Manager GUI to confirm that the node is properly connected to the NVP Controller Cluster and that the NVP Controller Cluster is seeing the integration bridge "br-int".

Node Setup: Ryu Plugin

The Ryu plugin requires Open vSwitch and `ryu`. Please install `ryu` and `openvswitch` in addition to `ryu` agent package.

Install `ryu`. There isn't `ryu` package for ubuntu yet.

```
pip install ryu
```

Install the Ryu agent package and openvswitch package:

```
apt-get -y install quantum-plugin-ryu-agent openvswitch-switch python-  
openvswitch openvswitch-datapath-dkms
```

The `ovs_ryu_plugin.ini` and `quantum.conf` created in the above step must be replicated on all nodes `quantum-plugin-ryu-agent`.

Then restart the agent

```
$ sudo service quantum-plugin-ryu-agent restart
```

Then restart Open vSwitch to properly load the kernel module:

```
$ sudo service openvswitch-switch restart
```

And restart the agent:

```
$ sudo service quantum-plugin-ryu-agent restart
```

All hosts running `quantum-plugin-ryu-agent` also requires that an OVS bridge named "br-int" exists. To create it, run:

```
ovs-vsctl add-br br-int
```

Install DHCP Agent

The DHCP service agent is compatible with all existing plugins and is required for all deployments where VMs should automatically receive IP addresses via DHCP.

The host running the `quantum-dhcp-agent` must be configured as a "data forwarding node" according to your plugin's requirements (see section above).

In addition, you must install the DHCP agent:

```
apt-get -y install quantum-dhcp-agent
```

Some options in `/etc/quantum/dhcp_agent.ini` must have certain values that depend on the plugin in use. The sub-sections below will indicate those values for certain plugins.

DHCP Agent Setup: OVS Plugin

The following DHCP agent options are required for the OVS plugin:

```
[DEFAULT]  
ovs_use_veth = True  
enable_isolated_metadata = True  
use_namespaces = True  
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver
```

DHCP Agent Setup: NVP Plugin

The following DHCP agent options are required for the NVP plugin:


```
[DEFAULT]
ovs_use_veth = True
enable_metadata_network = True
enable_isolated_metadata = True
use_namespaces = True
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver
```

DHCP Agent Setup: Ryu Plugin

The following DHCP agent options are required for the Ryu plugin:

```
[DEFAULT]
ovs_use_veth = True
use_namespace = True
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver
```

Install L3 Agent

Quantum has a widely used API extension to allow administrators and tenants to create "routers" that connect to L2 networks.

Many plugins rely on the L3 service agent to implement this L3 functionality. However, the following plugins have built in L3 capabilities:

- Nicira NVP Plugin
- Floodlight/BigSwitch Plugin (L3 functionality with BigSwitch only)
- PLUMgrid Plugin



Warning

Do NOT configure or use `quantum-l3-agent` if you are using one of the above plugins.



Note

The Floodlight/BigSwitch plugin supports both the open source [Floodlight](#) controller and the proprietary BigSwitch controller. However, only the proprietary BigSwitch controller implements L3 functionality. When using Floodlight as your OpenFlow controller, L3 functionality is not available.

For all other plugins, install the `quantum-l3-agent` binary on the network node.

```
apt-get -y install quantum-l3-agent
```

Create a bridge "br-ex" that will be used to uplink this node running `quantum-l3-agent` to the external network, then attach the NIC attached to the external network to this bridge.



Warning

OpenStack does not manage this routing for you, so you need to make sure that your host running the metadata service always has a route to reach

each private network's subnet via the external network IP of that subnet's OpenStack Networking router. To do this, you can run OpenStack Networking without namespaces, and run the quantum-l3-agent on the same host as nova-api. Alternatively, you can identify an IP prefix that includes all private network subnet's (e.g., 10.0.0.0/8) and then make sure that your metadata server has a route for that prefix with the OpenStack Networking router's external IP address as the next hop. For more validation information, refer to [Advanced configuration](#)

For example, with Open vSwitch and NIC eth1 connect to the external network, run:

```
ovs-vsctl add-br br-ex
ovs-vsctl add-port br-ex eth1
```

The node running quantum-l3-agent should not have an IP address manually configured on the NIC connected to the external network. 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.

The quantum-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, quantum-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 router's network namespace. The namespace will have the name "qrouter-<UUID of the router>". The following commands are examples of running commands in the namespace of a router 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>
```

Install OpenStack Networking CLI Client

Install the OpenStack Networking CLI client:

```
apt-get -y install python-pyparsing python-cliff python-quantumclient
```

Init, Config, and Log File Locations

Services can be started and stopped using the 'service' command. For example:

```
service quantum-server stop
service quantum-server status
service quantum-server start
service quantum-server restart
```

Log files are found in /var/log/quantum.

Configuration files are in /etc/quantum.

Installing Packages (Fedora)

The OpenStack packages for Fedora can be retrieved from: <https://apps.fedoraproject.org/packages/s/openstack>. Additional information can be found at <https://fedoraproject.org/wiki/OpenStack>

RPC Setup

OpenStack Networking uses RPC to allow DHCP agents and any plugin agents to communicate with the main quantum-server process. Commonly, this can use the same RPC mechanism used by other OpenStack components like Nova.

To use QPID AMQP as the message bus for RPC, make sure that QPID is installed on a host reachable via the management network (if this is already the case because of deploying another service like Nova, this existing QPID setup is sufficient):

```
sudo yum -y install qpid-cpp-server qpid-cpp-server-daemon
sudo chkconfig qpid on
sudo service qpid start
```

Then update `/etc/quantum/quantum.conf` with these values:

```
rpc_backend = quantum.openstack.common.rpc.impl_qpid
qpid_hostname = <mgmt-IP-of-qpid-host>
```



Important

The Fedora packaging has a number of utility scripts that configure all of the necessary configuration files. The scripts can also be used to understand what needs to be configured for the specific OpenStack Networking services. The scripts will be described below. Please note that the scripts make use of the package `openstack-utils`. Please install:

```
sudo yum install -y openstack-utils
```

Install quantum-server and plugin

Install `quantum-server` and plugin. **Note** the client is installed as a dependency for the OpenStack Networking service. Each plugin has its own package, named `openstack-quantum-<plugin>`. `openvswitch` will be used in the examples below. A complete list of the supported plugins can be seen at: https://fedoraproject.org/wiki/Quantum#Quantum_Plugins.

```
sudo yum install -y openstack-quantum
sudo yum install -y openstack-quantum-openvswitch
```

Most plugins require a database to be installed and configured in a plugin configuration file. The Fedora packaging for OpenStack Networking a server setup utility scripts that will take care of this. For example:

```
sudo quantum-server-setup --plugin openvswitch
```

Enable and start the service:

```
sudo chkconfig quantum-server on
sudo service quantum-server start
```

Install quantum-plugin-*-agent

Some plugins utilize an agent that runs on each node that handles data packets. This includes any node running nova-compute, as well as nodes running dedicated OpenStack Networking agents like quantum-dhcp-agent and quantum-l3-agent (see below). If your plugin uses an agent, this section describes how to run the agent for this plugin, as well as the basic configuration options.

Open vSwitch Agent

Install the OVS agent:

```
sudo yum install -y openstack-quantum-openvswitch
```

Run the agent setup script:

```
sudo quantum-node-setup --plugin openvswitch
```

All hosts running quantum-plugin-openvswitch-agent also requires that an OVS bridge named "br-int" exists. To create it, run:

```
ovs-vsctl add-br br-int
```

Enable and start the agent:

```
sudo chkconfig quantum-openvswitch-agent on
sudo service quantum-openvswitch-agent start
```

Enable the ovs cleanup utility:

```
sudo chkconfig quantum-ovs-cleanup on
```

Install quantum-dhcp-agent

The DHCP agent is part of the openstack-quantum package.

```
sudo yum install -y openstack-quantum
```

Run the agent setup script:

```
sudo quantum-dhcp-setup --plugin openvswitch
```

Enable and start the agent:

```
sudo chkconfig quantum-dhcp-agent on
sudo service quantum-dhcp-agent start
```

Install quantum-l3-agent

The L3 agent is part of the openstack-quantum package.

Create a bridge "br-ex" that will be used to uplink this node running quantum-l3-agent to the external network, then attach the NIC attached to the external network to this bridge. For example, with Open vSwitch and NIC eth1 connect to the external network, run:

```
sudo ovs-vsctl add-br br-ex
sudo ovs-vsctl add-port br-ex eth1
```

The node running quantum-l3-agent should not have an IP address manually configured on the NIC connected to the external network. 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.

```
sudo yum install -y openstack-quantum
```

Run the agent setup script:

```
sudo quantum-l3-setup --plugin openvswitch
```

Enable and start the agent:

```
sudo chkconfig enable quantum-l3-agent on
sudo start quantum-l3-agent
```

Enable and start the meta data agent:

```
sudo chkconfig quantum-metadata-agent on
sudo service quantum-metadata-agent start
```

Install OpenStack Networking CLI client

Install the OpenStack Networking CLI client:

```
sudo yum install -y python-quantumclient
```

Init, Config, and Log File Locations

Services can be started and stopped using the 'service' command. For example:

```
sudo service quantum-server stop
sudo service quantum-server status
sudo service quantum-server start
sudo service quantum-server restart
```

Log files are found in `/var/log/quantum`.

Configuration files are in `/etc/quantum`.

3. Required Configuration for OpenStack Identity & Compute

OpenStack Identity	23
OpenStack Compute	24

Using OpenStack Networking requires particular configuration and setup steps for both the OpenStack Identity Service and the OpenStack Compute Service. This chapter describes these steps.

OpenStack Identity

Procedure 3.1. To configure OpenStack Identity for use with OpenStack Networking

1. To Create an OpenStack Networking Service Entry

OpenStack Networking needs to be available in the OpenStack Compute service catalog. The steps for this depend on whether you are using the SQL catalog driver or the template catalog driver.

With the *SQL driver*, for a given region (\$REGION), IP address of the OpenStack Networking server (\$IP), and service ID (\$ID) returned by the OpenStack Compute service catalog, run:

```
keystone service-create --name quantum --type network --description
'OpenStack Networking Service'
```

Make a note of the ID returned by the command and put it in the \$ID location.

```
keystone endpoint-create --region $REGION --service-id $ID --publicurl
'http://$IP:9696/' --adminurl 'http://$IP:9696/' --internalurl 'http://
$IP:9696/'
```

Here's an example with real values:

```
$ keystone service-create --name quantum --type network --description
'OpenStack Networking Service'
```

Property	Value
description	OpenStack Networking Service
id	26a55b340e254ad5bb78c0b14391e153
name	quantum
type	network

```
$ keystone endpoint-create --region myregion --service-id
26a55b340e254ad5bb78c0b14391e153 \
--publicurl "http://10.211.55.17:9696/" --adminurl "http://10.211.55.
17:9696/" --internalurl "http://10.211.55.17:9696/"
```

With the *template driver*, for a given region (\$REGION) and IP address of the OpenStack Networking server (\$IP), add the following content to your OpenStack Compute catalog template file (default_catalog.templates).

```
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
```

Here is an example with real values:

```
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
```

2. Create OpenStack Networking Service User

For OpenStack Compute to speak to the OpenStack Networking API, and for some internal components of OpenStack Networking to communicate with the OpenStack Networking API, you need to provide them with admin user credentials that they can use when accessing the OpenStack Networking API. The suggested approach is to create a special 'service' tenant, create a 'quantum' user within this tenant, and to assign this user an 'admin' role. Kindly check the ID for user, role and tenant.

For example:

```
$ ADMIN_ROLE=$(get_id keystone role-create --name=admin)

$ QUANTUM_USER=$(get_id keystone user-create --name=quantum --pass=
"$QUANTUM_PASSWORD" --email=demo@example.com --tenant-id service)

$ keystone user-role-add --user_id $QUANTUM_USER --role_id $ADMIN_ROLE --
tenant_id service
```

See the OpenStack Installation Guides for more details about creating service entries and service users.

OpenStack Compute

Unlike traditional OpenStack Compute deployments, when OpenStack Networking is in use, OpenStack Compute should not run a nova-network. Instead, OpenStack Compute delegates almost all of the network-related decisions to OpenStack Networking. Tenant-facing API calls to manage objects like security groups + floating IPs are proxied by OpenStack Compute to OpenStack Network APIs. However, operator-facing tools (e.g., nova-manage) are not proxied and therefore should NOT be used as these calls are not proxied.



Warning

Therefore, it is very important that you refer to this guide when configuring networking, rather than relying on OpenStack Compute networking documentation or past experience with OpenStack Compute. If a Nova CLI command or configuration option related to networking is not mentioned in

this guide, it is likely not supported for use with OpenStack Networking. In particular, using CLI tools like 'nova-manage' and 'nova' to manage networks or IP addressing, including both fixed and floating IPs, is not supported with OpenStack Networking.



Note

It is strongly recommended that you uninstall nova-network and reboot any physical nodes that had been running nova-network before using them to run OpenStack Networking. Inadvertently running the nova-network process while using OpenStack Networking can cause problems, as can stale iptables rules pushed down by previously running nova-network.

The next section describes nova.conf settings that are required for OpenStack Compute to work properly with OpenStack Networking, rather than the legacy nova-network mechanism.

Networking API & and Credential Configuration

Each time a VM is provisioned or deprovisioned in OpenStack Compute, nova-* services communicate with OpenStack Networking via the standard API. To do so, it requires the following items in the nova.conf used by each nova-compute and nova-api instance:

- `network_api_class`: must be modified from default to 'nova.network.quantumv2.api.API' to indicate that OpenStack Networking should be used rather than the traditional nova-network networking model.
- `quantum_url`: must include the hostname/IP and port of the quantum-server instance for this deployment.
- `quantum_auth_strategy`: should be kept as default 'keystone' for all production deployments.
- `quantum_admin_tenant_name`: must be modified to be the name of the service tenant created in the above section on OpenStack Identity configuration.
- `quantum_admin_username`: must be modified to be the name of the user created in the above section on OpenStack Identity configuration.
- `quantum_admin_password`: must be modified to be the password of the user created in the above section on OpenStack Identity configuration.
- `quantum_admin_auth_url`: must be modified to point to the OpenStack 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.

Security Group Configuration

The OpenStack Networking Service provides security group functionality using a mechanism that is more flexible and powerful than the security group capabilities built into OpenStack Compute. Thus, when using OpenStack Networking, nova.conf should always disable built-in security groups and proxy all security group calls to the OpenStack Networking API.

Failure to do so will result in conflicting security policies being simultaneously applied by both services. To proxy security groups to Quantum, use the following configuration values:

- `firewall_driver` : must be set to 'nova.virt.firewall.NoopFirewallDriver' so that nova-compute does not perform iptables-based filtering itself.
- `security_group_api` : must be set to 'quantum' so that all security group requests are proxied to the OpenStack Network Service.

Metadata Configuration

The OpenStack Compute service allows VMs to query metadata associated with a VM by making a web request to a special 169.254.169.254 address. Quantum 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. Enabling this requires setting the following fields in nova.conf:

- `service_quantum_metadata_proxy`: must be set to 'true', otherwise nova-api will not properly respond to requests from the quantum-metadata-agent.
- `quantum_metadata_proxy_shared_secret`: a string "password" value that should also be configured in the `metadata_agent.ini` file in order to authenticate requests made for metadata. The default value of the empty string in both files will allow metadata to function, but will be insecure if any non-trusted entities have access to the metadata APIs exposed by nova-api.



Note

As a precaution, even when using `quantum_metadata_proxy_shared_secret`, it is recommended that you do not expose metadata using the same nova-api instances that are used for tenants. Instead, run a dedicated set of nova-api instances for metadata 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.

Vif-plugging Configuration

When nova-compute creates a VM, it must "plug" each of the VM's vNICs into an OpenStack Networking controlled virtual switch, and inform the virtual switch about the OpenStack Networking port-id associated with each vNIC. Different OpenStack Networking plugins may require different types of vif-plugging. The type of vif-plugging to be used is specified in the nova.conf for each nova-compute instance.

The following plugins support the "port bindings" API extension that allows Nova to query for the type of vif-plugging required.

- OVS plugin
- Linux Bridge Plugin
- NEC Plugin
- Big Switch Plugin

- Hyper-V Plugin
- Brocade Plugin

For these plugins, the default values in `nova.conf` are sufficient. For other plugins, see the sub-sections below for vif-plugging configuration, or consult external plugin documentation.



Note

It is possible that the vif-plugging configuration required for nova-compute will vary even within a single deployment if your deployment includes heterogeneous compute platforms (e.g., some computes hosts are KVM while others are ESX).

Vif-plugging with Nicira NVP Plugin

The choice of vif-plugging for the NVP Plugin depends on what version of libvirt is in use.



Checking your libvirt version

To check your libvirt version, use `libvirtd --version`.

- When using libvirt (version \geq 0.9.11):
`libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtOpenVswitchVirtualPortDriver`
- When using libvirt (version $<$ 0.9.11):
`libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtOpenVswitchDriver`
- When using ESX: no vif-plugging configuration required.
- When using XenServer: `xenapi_vif_driver=nova.virt.xenapi.vif.XenAPIOpenVswitchDriver`



Note

When using libvirt $<$ 0.9.11, one must also edit `/etc/libvirt/qemu.conf`, uncomment the entry for 'cgroup_device_acl', add the value `'/dev/net/tun'` to the list of items for the configuration entry, and then restart libvirtd.

Example nova.conf (for nova-compute and nova-api)

Example values for the above settings, assuming a cloud controller node running OpenStack Compute and OpenStack Networking with an IP address of 192.168.1.2 and vif-plugging using the LibvirtHybridOVSBridgeDriver.

```
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://192.168.1.2:9696
quantum_auth_strategy=keystone
quantum_admin_tenant_name=service
quantum_admin_username=quantum
quantum_admin_password=password
quantum_admin_auth_url=http://192.168.1.2:35357/v2.0
security_group_api=quantum
```

```
firewall_driver=nova.virt.firewall.NoopFirewallDriver

service_quantum_metadata_proxy=true
quantum_metadata_proxy_shared_secret=foo

# needed only for nova-compute and only for some plugins
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

4. Using OpenStack Networking

Core OpenStack Networking API Features	29
Using OpenStack Compute with OpenStack Networking	32

There are two main approaches to using OpenStack Networking. The first is to expose the OpenStack Networking API to cloud tenants, allowing them to build rich network topologies. The second is to have the cloud administrator, or another tool run the cloud admin, create network connectivity on behalf of tenants. In this document, we commonly describe operations as being performed by a tenant, but they might also be performed by the cloud admin on behalf of the tenant.

Core OpenStack Networking API Features

Once OpenStack Networking is installed and running, both tenants and admins primarily interact with the service via create-read-update-delete (CRUD) API operations performed either directly against the API, or more commonly via the 'quantum' CLI tool. Like other OpenStack CLI tools, the 'quantum' tool is just a basic wrapper around the OpenStack Networking API, so any operation that can be performed via the CLI has an equivalent API call that can be performed programmatically.

The CLI supports many options for filtering results, limiting fields show, etc. For details, refer to the OpenStack Networking CLI documentation.

API Abstractions

The OpenStack Networking v2.0 API provides control over both L2 network topologies and the IP addresses used on those networks (IP Address Management or IPAM). There is also an extension to cover basic L3 forwarding and NAT, providing capabilities similar to nova-network.

In the OpenStack Networking API, an isolated L2 network segment (similar to a VLAN) is called a 'Network' and forms the basis for describing the L2 network topology available in a given OpenStack Networking deployment.

The OpenStack Networking API uses the notion of a 'Subnet', which associates a block of IP addresses and other network configuration (e.g., default gateway, dns-servers) with an OpenStack Networking network. Each subnet represents an IPv4 or IPv6 address block and each OpenStack Networking network can have multiple subnets, if desired.

A 'Port' represents an attachment port to a L2 OpenStack Networking network. When a port is created on the network, by default it will be allocated an available fixed IP address out of one of the designated Subnets for each IP version (if one exists). When the Port is destroyed, the allocated addresses return to the pool of available IPs on the subnet(s). Users of the OpenStack Networking API can either choose a specific IP address from the block, or let OpenStack Networking choose the first available IP address.

The table below summarizes the attributes available for each of these abstractions. For more operations about API abstraction and operations, please refer to the [OpenStack Networking v2.0 API Developer Guide](#).

Table 4.1. Network

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the network.
name	String	None	Human-readable name for the network. Might not be unique.
admin_state_up	Bool	True	The administrative state of network. If false (down), the network does not forward packets.
status	String	N/A	Indicates whether network is currently operational.
subnets	list(uuid-str)	Empty list	Subnets associated with this network.
shared	Bool	False	Specifies whether the network resource can be accessed by any tenant or not.
tenant_id	uuid-str	N/A	Owner of network. Only admin users can specify a tenant_id other than its own.

Table 4.2. Subnet

Attribute	Type	Default Value	Description
id	uuid-str	generated	UUID representing the subnet
network_id	uuid-str	N/A	network this subnet is associated with.
name	String	None	Human-readable name for the subnet. Might not be unique.
ip_version	int	4	IP version
cidr	string	N/A	cidr representing IP range for this subnet, based on IP version
gateway_ip	string	first address in <i>cidr</i>	default gateway used by devices in this subnet
dns_nameservers	list(str)	Empty list	DNS name servers used by hosts in this subnet.
allocation_pools	list(dict)	Every address in <i>cidr</i> , excluding <i>gateway_ip</i> if configured	Sub-ranges of cidr available for dynamic allocation to ports [{ "start": "10.0.0.2", "end": "10.0.0.254" }]
host_routes	list(dict)	Empty List	Routes that should be used by devices with IPs from this subnet (not including local subnet route).
enable_dhcp	Bool	True	Specifies whether DHCP is enabled for this subnet or not.
tenant_id	uuid-str	N/A	Owner of network. Only admin users can specify a tenant_id other than its own.

Table 4.3. Port

Attribute	Type	Default Value	Description
id	uuid-str	generated	UUID for the port.
network_id	uuid-str	N/A	Network that this port is associated with.
name	String	None	Human-readable name for the port. Might not be unique.
admin_state_up	bool	true	Administrative state of port. If false (down), port does not forward packets.
status	string	N/A	Indicates whether network is currently operational.
mac_address	string	generated	Mac address to use on this port.

Attribute	Type	Default Value	Description
fixed_ips	list(dict)	automatically allocated from pool	Specifies IP addresses for the port thus associating the port itself with the subnets where the IP addresses are picked from
device_id	str	None	identifies the device (e.g., virtual server) using this port.
device_owner	str	None	Identifies the entity (e.g.: dhcp agent) using this port.
tenant_id	uuid-str	N/A	Owner of network. Only admin users can specify a tenant_id other than its own.

OpenStack Networking CLI Guide

Before going further, we STRONGLY suggest that you quickly read the few pages in the [OpenStack CLI Guide](#) that are specific to OpenStack Networking. OpenStack Networking's CLI has some advanced capabilities that are described only in that guide.

Basic Operations

Create a network

```
$ quantum net-create net1
```

Create a subnet associated with net1

```
$ quantum subnet-create net1 10.0.0.0/24
```

List ports on a tenant

```
$ quantum port-list
```

device_owner field describes who owns the port. A port whose *device_owner* begins with "network:" is created by OpenStack Networking and a port whose *device_owner* begins with "compute:" is created by OpenStack Compute (compute service).

```
$ quantum port-list -c id -c fixed_ips -c device_owner
```

port-show shows a detail of a specified port.

```
$ quantum port-show <port-id>
```

Admin API configuration

These same calls can be performed by the cloud admin on behalf of the tenants by specifying a *tenant_id* in the request, for example:

```
quantum net-create --tenant-id=<tenant-id> net1
```

This *tenant_id* should be the tenant ID from OpenStack Identity. To view all OpenStack Identity tenant IDs, run the following command as an OpenStack Identity (keystone) admin user:

```
keystone tenant-list
```

Advanced API Operations Examples

Network, Subnet & Port Creation

- Create a "shared" network (i.e., a network that can be used by all tenants)

```
quantum net-create --shared public-net
```

- Create a subnet that has a specific gateway IP address.

```
quantum subnet-create --gateway 10.0.0.254 net1 10.0.0.0/24
```

- Create a subnet that has no gateway IP address.

```
quantum subnet-create --no-gateway net1 10.0.0.0/24
```

- Create a subnet for which DHCP is disabled.

```
quantum subnet-create net1 10.0.0.0/24 --enable_dhcp False
```

- Create subnet with a specific set of host routes:

```
quantum subnet-create test-net1 40.0.0.0/24 --host_routes type=dict list=true destination=40.0.1.0/24,nexthop=40.0.0.2
```

- Create subnet with a specific set of dns nameserver:

```
quantum subnet-create test-net1 40.0.0.0/24 --dns_nameservers list=true 8.8.8.7 8.8.8.8
```

Searches

- Find all Ports/IPs allocated on a network.

```
quantum port-list -- --network_id <net-id>
```

Using OpenStack Compute with OpenStack Networking

Basic Workflow

Check available networks:

```
$ quantum net-list
```

Boot the VM with a single NIC on the selected network (net1):

```
$ nova boot --image <img> --flavor <flavor> --nic net-id=<net-id> <vm-name>
```

Congrats, you have booted a VM on an OpenStack Networking network. You may need to configure [security group rules](#) to allow accesses to the VM.

There is now an OpenStack Networking port on 'net1' that corresponds to the VM Nic. You can view it with the following command, which searches for all ports with a "device_id" corresponding to the OpenStack Compute instance UUID:

```
$ quantum port-list -- --device_id=<vm-id>
```

To view only a few fields of the port, you can limit output using `-c`. For example to see only the `mac_address` of the port, use:

```
$ quantum port-list -c mac_address -- --device_id=<vm-id>
```

You could temporarily disable the port from sending traffic by updating it to have `admin_state_up=False`:

```
$ quantum port-update <port-id> --admin_state_up=False
```

When we delete the OpenStack Compute VM, the underlying OpenStack Networking port is automatically deleted:

```
$ quantum port-list -c mac_address -- --device_id=<vm-id>
```

Advanced VM creation

- Booting a VM with multiple NICs

```
$ nova boot --image <img> --flavor <flavor> \
--nic net-id=<net1-id> --nic net-id=<net2-id> <vm-name>
```

- Booting a VM with a specific IP address. To do this, we need to create an OpenStack Networking port with a specific IP address first, and then boot a VM specifying a `port-id` rather than a `net-id`.

```
$ quantum port-create --fixed-ip subnet_id=<subnet-id>,ip_address=192.168.
57.101 <net-id>
nova boot --image <img> --flavor <flavor> --nic port-id=<port-id> <vm-name>
```



Note

`v4-fixed-ip` parameter of `--nic` option in `nova` command is not supported with OpenStack Networking at the moment.

- booting a VM with no `--nic` option specified. In this case the launched VM connects to all networks that are accessible to the tenant who submits the request.

```
$ nova boot --image <img> --flavor <flavor> <vm-name>
```

Security Groups (Enabling Ping and SSH on VMs)

If using a plugin that implements quantum security groups you can configure security group rules directly by using `quantum security-group-rule-create` to enable access to your VMs. The example below allows `ping` and `ssh` to your VMs.

```
$ quantum security-group-rule-create --protocol icmp --direction ingress
default
$ quantum security-group-rule-create --protocol tcp --port-range-min 22
--port-range-max 22 --direction ingress default
```

If your plugin does not implement quantum security group, security groups can still be leveraged via OpenStack Compute. This can be done using `nova secgroup-add-rule` or

euca-authorize command to enable accesses to your VMs. Below are the nova commands to allow **ping** and **ssh** to your VMs.

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```



Note

If your plugin implements quantum security groups you can still leverage nova security groups by setting `security_group_api = quantum` in `nova.conf`. After setting this all nova security group commands will be proxied to quantum.

5. Under the Hood

Open vSwitch	35
Linux bridge	45

This chapter describes two networking scenarios and how the Open vSwitch plugin and the Linux bridging plugin implement these scenarios.

Open vSwitch

This section describes how the Open vSwitch plugin implements the OpenStack Networking abstractions.

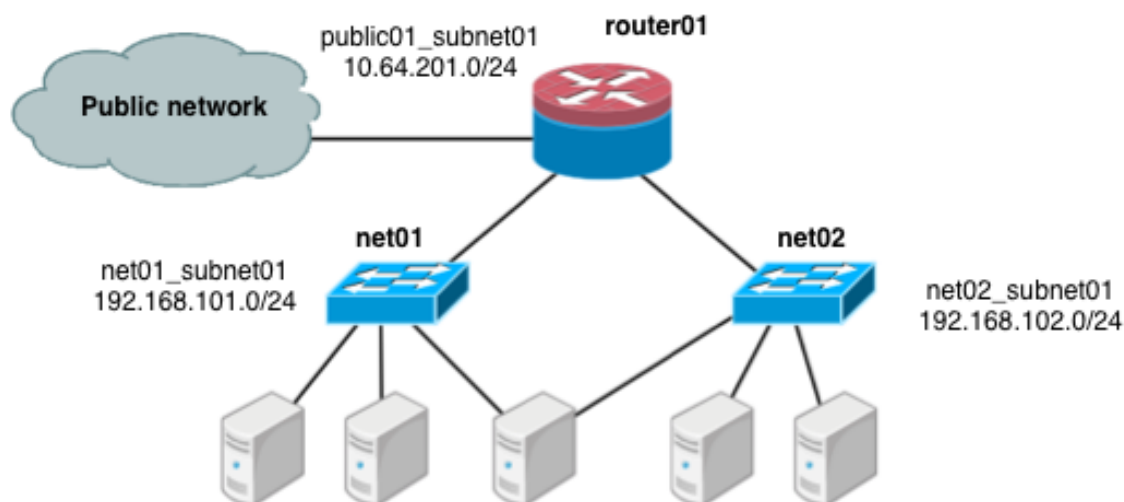
Configuration

This example uses VLAN isolation on the switches to isolate tenant networks. This configuration labels the physical network associated with the public network as `physnet1`, and the physical network associated with the data network as `physnet2`, which leads to the following configuration options in `ovs_quantum_plugin.ini`:

```
[OVS]
tenant_network_type = vlan
network_vlan_ranges = physnet2:100:110
integration_bridge = br-int
bridge_mappings = physnet2:br-eth1
```

Scenario 1: one tenant, two networks, one router

The first scenario has two private networks (`net01`, and `net02`), each with one subnet (`net01_subnet01`: 192.168.101.0/24, `net02_subnet01`, 192.168.102.0/24). Both private networks are attached to a router that contains them to the public network (10.64.201.0/24).



Under the `service` tenant, create the shared router, define the public network, and set it as the default gateway of the router

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ quantum router-create router01
$ quantum net-create --tenant-id $tenant public01 \
    --provider:network_type flat \
    --provider:physical_network physnet1 \
    --router:external=True
$ quantum subnet-create --tenant-id $tenant --name public01_subnet01 \
    --gateway 10.64.201.254 public01 10.64.201.0/24 --enable_dhcp False
$ quantum router-gateway-set router01 public01
```

Under the `demo` user tenant, create the private network `net01` and corresponding subnet, and connect it to the `router01` router. Configure it to use VLAN ID 101 on the physical switch.

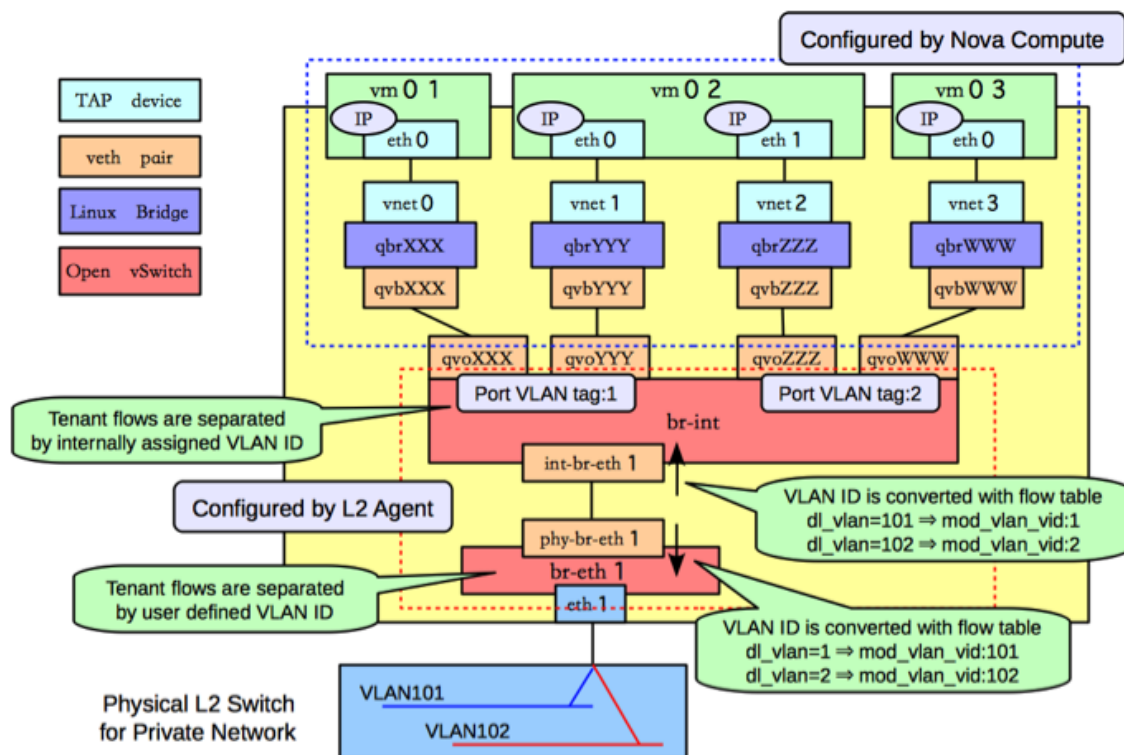
```
$ tenant=$(keystone tenant-list|awk '/demo/ {print $2}')
$ quantum net-create --tenant-id $tenant net01 \
    --provider:network_type vlan \
    --provider:physical_network physnet2 \
    --provider:segmentation_id 101
$ quantum subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.
168.101.0/24
$ quantum router-interface-add router01 net01_subnet01
```

Similarly, for `net02`, using VLAN ID 102 on the physical switch:

```
$ quantum net-create --tenant-id $tenant net02 \
    --provider:network_type vlan \
    --provider:physical_network physnet2 \
    --provider:segmentation_id 102
$ quantum subnet-create --tenant-id $tenant --name net02_subnet01
net02 192.168.102.0/24
$ quantum router-interface-add router01 net02_subnet01
```

Scenario 1: Compute host config

The following figure shows how to configure various Linux networking devices on the compute host:



Types of network devices



Note

There are four distinct type of virtual networking devices: TAP devices, veth pairs, Linux bridges, and Open vSwitch bridges. For an ethernet frame to travel from `eth0` of virtual machine `vm01`, to the physical network, it must pass through nine devices inside of the host: TAP `vnet0`, Linux bridge `qbrXXX`, veth pair (`qcbXXX`, `qvoXXX`), Open vSwitch bridge `br-int`, veth pair (`int-br-eth1`, `phy-br-eth1`), and, finally, the physical network interface card `eth1`.

A **TAP device**, such as `vnet0` is how hypervisors such as KVM and Xen implement a virtual network interface card (typically called a VIF or vNIC). An ethernet frame sent to a TAP device is received by the guest operating system.

A **veth pair** is a pair of virtual network interfaces correctly directly together. An ethernet frame sent to one end of a veth pair is received by the other end of a veth pair. OpenStack networking makes use of veth pairs as virtual patch cables in order to make connections between virtual bridges.

A **Linux bridge** behaves like a hub: you can connect multiple (physical or virtual) network interfaces devices to a Linux bridge. Any ethernet frames that come in from one interface attached to the bridge is transmitted to all of the other devices.

An **Open vSwitch bridge** behaves like a virtual switch: network interface devices connect to Open vSwitch bridge's ports, and the ports can be configured much like a physical switch's ports, including VLAN configurations.

Integration bridge

The `br-int` OpenvSwitch bridge is the integration bridge: all of the guests running on the compute host connect to this bridge. OpenStack Networking implements isolation across these guests by configuring the `br-int` ports.

Physical connectivity bridge

The `br-eth1` bridge provides connectivity to the physical network interface card, `eth1`. It connects to the integration bridge by a veth pair: (`int-br-eth1`, `phy-br-eth1`).

VLAN translation

In this example, `net01` and `net02` have VLAN ids of 1 and 2, respectively. However, the physical network in our example only supports VLAN IDs in the range 101 through 110. The Open vSwitch agent is responsible for configuring flow rules on `br-int` and `br-eth1` to do VLAN translation. When `br-eth1` receives a frame marked with VLAN ID 1 on the port associated with `phy-br-eth1`, it modifies the VLAN ID in the frame to 101. Similarly, when `br-int` receives a frame marked with VLAN ID 101 on the port associated with `int-br-eth1`, it modifies the VLAN ID in the frame to 1.

Security groups: iptables and Linux bridges

Ideally, the TAP device `vnet0` would be connected directly to the integration bridge, `br-int`. Unfortunately, this isn't possible because of how OpenStack security groups are currently implemented. OpenStack uses iptables rules on the TAP devices such as `vnet0` to implement security groups, and Open vSwitch is not compatible with iptables rules that are applied directly on TAP devices that are connected to an Open vSwitch port.

OpenStack Networking uses an extra Linux bridge and a veth pair as a workaround for this issue. Instead of connecting `vnet0` to an Open vSwitch bridge, it is connected to a Linux bridge, `qbrXXX`. This bridge is connected to the integration bridge, `br-int`, via the (`qvbXXX`, `qvoXXX`) veth pair.

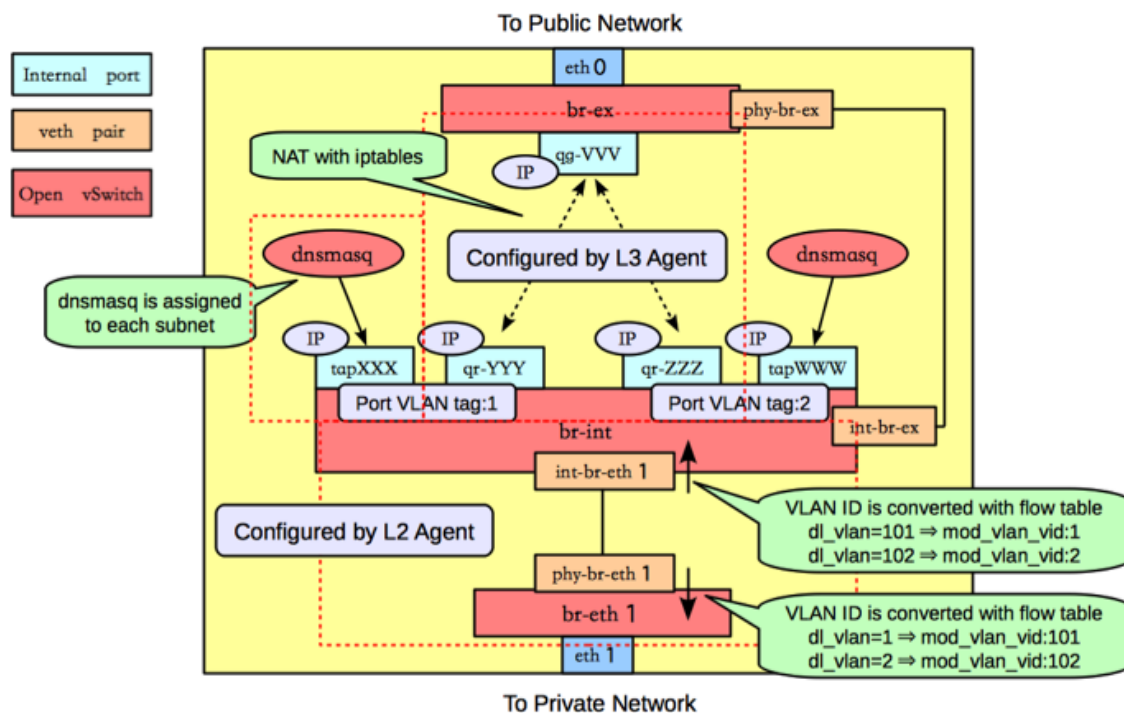
Scenario 1: Network host config

Recall that the network host runs the `quantum-openvswitch-plugin-agent`, the `quantum-dhcp-agent`, `quantum-l3-agent`, and `quantum-metadata-agent` services.

On the network host, assume that `eth0` is connected to the external network, and `eth1` is connected to the data network, which leads to the following configuration options in `ovs_quantum_plugin.ini`:

```
[OVS]
tenant_network_type = vlan
network_vlan_ranges = physnet2:101:110
integration_bridge = br-int
bridge_mappings = physnet1:br-ex,physnet2:br-eth1
```

The following figure shows the network devices on the network host:



As on the compute host, there is an Open vSwitch integration bridge (`br-int`) and an Open vSwitch bridge connected to the data network (`br-eth1`), and the two are connected by a veth pair, and the quantum-openvswitch-plugin-agent configures the ports on both switches to do VLAN translation.

There is also an additional Open vSwitch bridge, `br-ex`, which connects to the physical interface that is connected to the external network. In this example, that physical interface is `eth0`.



Note

While the integration bridge and the external bridge are connected by a veth pair (`int-br-ex`, `phy-br-ex`), this example uses layer 3 connectivity to route packets from the internal networks to the public network: no packets traverse that veth pair in this example.

Open vSwitch internal ports

The network host uses Open vSwitch *internal ports*. Internal ports are a mechanism that allows you to assign one or more IP addresses to an Open vSwitch bridge. In previous example, the `br-int` bridge has four internal ports: `tapXXX`, `qr-YYY`, `qr-ZZZ`, `tapWWW`. Each internal port has a separate IP address associated with it. There is also an internal port, `qg-VVV`, on the `br-ex` bridge.

DHCP agent

By default, The OpenStack Networking DHCP agent uses a program called `dnsmasq` to provide DHCP services to guests. OpenStack Networking must create an internal port for

each network that requires DHCP services and attach a `dnsmasq` process to that port. In the previous example, the interface `tapXXX` is on subnet `net01_subnet01`, and the interface `tapWWW` is on `net02_subnet01`.

L3 agent (routing)

The OpenStack Networking L3 agent implements routing through the use of Open vSwitch internal ports and relies on the network host to route the packets across the interfaces. In this example: interface `qr-YYY`, which is on subnet `net01_subnet01`, has an IP address of `192.168.101.1/24`, interface `qr-ZZZ`, which is on subnet `net02_subnet01`, has an IP address of `192.168.102.1/24`, and interface `qg-VVV`, which has an IP address of `10.64.201.254/24`. Because each of these interfaces is visible to the network host operating system, it will route the packets appropriately across the interfaces, as long as an administrator has enabled IP forwarding.

The L3 agent uses `iptables` to implement floating IPs to do the network address translation (NAT).

Overlapping subnets and network namespaces

One problem with using the host to implement routing is that there is a chance that one of the OpenStack Networking subnets might overlap with one of the physical networks that the host uses. For example, if the management network is implemented on `eth2` (not shown in the previous example), by coincidence happens to also be on the `192.168.101.0/24` subnet, then this will cause routing problems because it is impossible to determine whether a packet on this subnet should be sent to `qr-YYY` or `eth2`. In general, if end-users are permitted to create their own logical networks and subnets, then the system must be designed to avoid the possibility of such collisions.

OpenStack Networking uses Linux *network namespaces* to prevent collisions between the physical networks on the network host, and the logical networks used by the virtual machines. It also prevents collisions across different logical networks that are not routed to each other, as you will see in the next scenario.

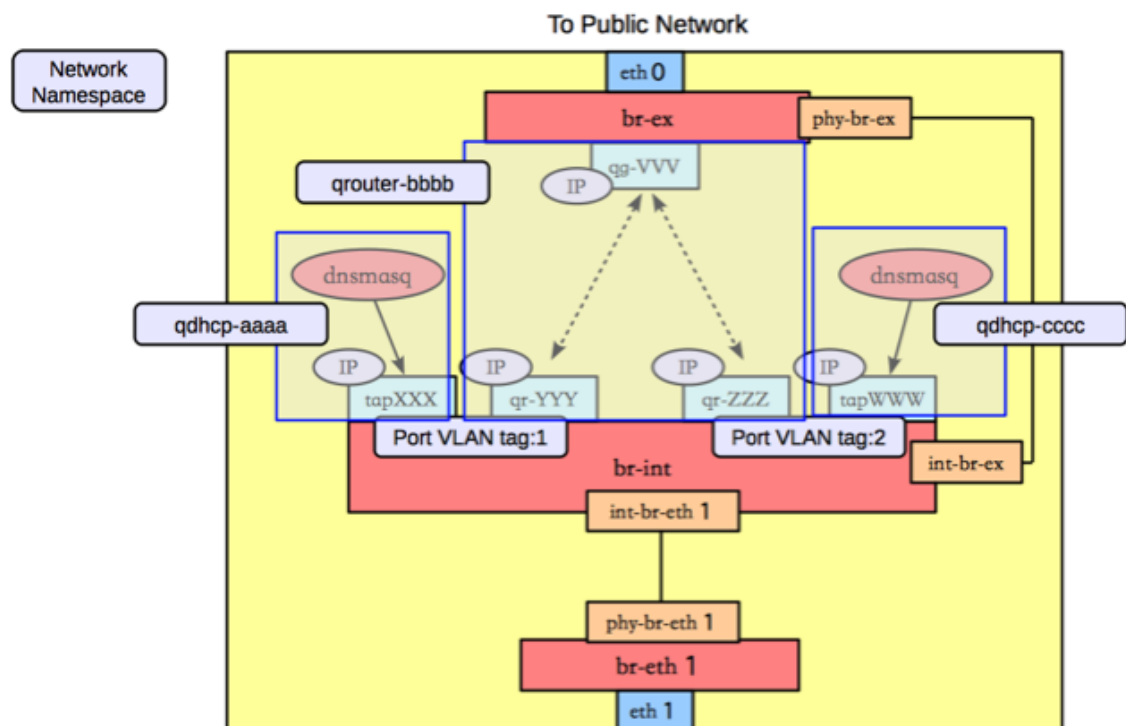
A network namespace can be thought of as an isolated environment that has its own networking stack. A network namespace has its own network interfaces, routes, and `iptables` rules. You can think of it like a chroot jail, except for networking instead of a file system. As an aside, LXC (Linux containers) use network namespaces to implement networking virtualization.

OpenStack Networking creates network namespaces on the network host in order to avoid subnet collisions.

In this example, there are three network namespaces, as depicted in the following figure.

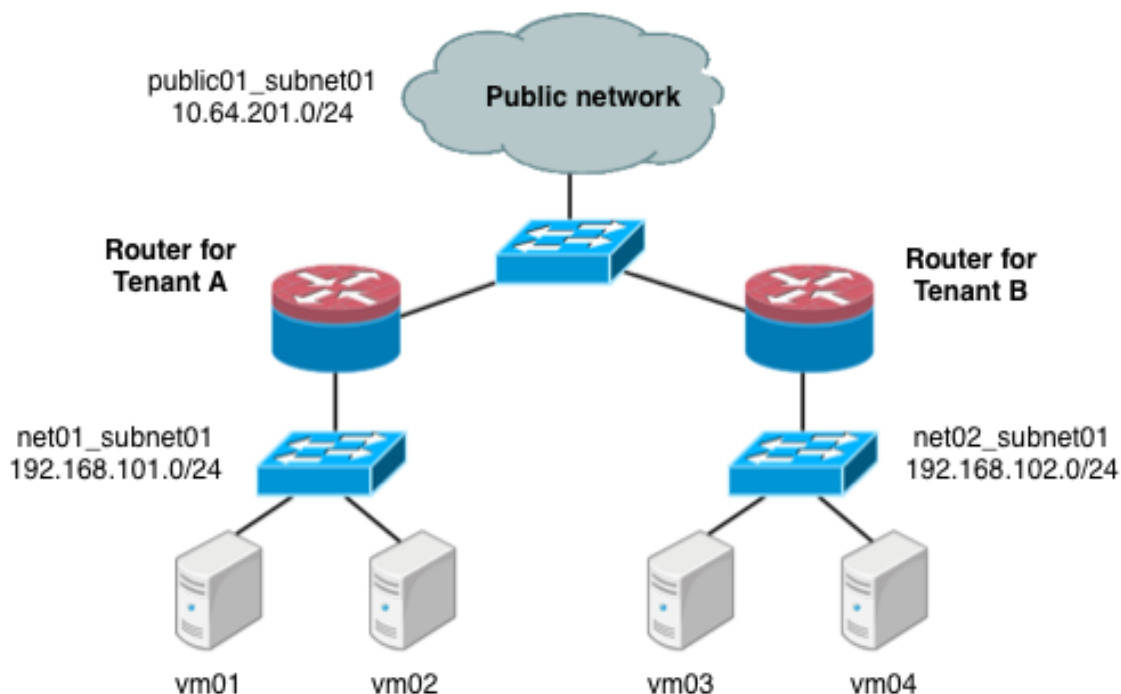
- `qdhcp-aaa`: contains the `tapXXX` interface and the `dnsmasq` process that listens on that interface, to provide DHCP services for `net01_subnet01`. This allows overlapping IPs between `net01_subnet01` and any other subnets on the network host.
- `qrouter-bbbb`: contains the `qr-YYY`, `qr-ZZZ`, and `qg-VVV` interfaces, and the corresponding routes. This namespace implements `router01` in our example.

- `qdhcp-ccc`: contains the `tapWWW` interface and the `dnsmasq` process that listens on that interface, to provide DHCP services for `net02_subnet01`. This allows overlapping IPs between `net02_subnet01` and any other subnets on the network host.



Scenario 2: two tenants, two networks, two routers

The second scenario has two tenants (A, B). Each tenant has a network with one subnet, and each one has a router that connects them to the public Internet.



Under the `service` tenant, define the public network:

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ quantum net-create --tenant-id $tenant public01 \
  --provider:network_type flat \
  --provider:physical_network physnet1 \
  --router:external=True
$ quantum subnet-create --tenant-id $tenant --name public01_subnet01 \
  --gateway 10.64.201.254 public01 10.64.201.0/24 --enable_dhcp False
```

Under the `tenantA` user tenant, create the tenant router and set its gateway for the public network.

```
$ tenant=$(keystone tenant-list|awk '/tenantA/ {print $2}')
$ quantum router-create --tenant-id $tenant router01
$ quantum router-gateway-set router01 public01
```

Then, define private network `net01` using VLAN ID 102 on the physical switch, along with its subnet, and connect it to the router.

```
$ quantum net-create --tenant-id $tenant net01 \
  --provider:network_type vlan \
  --provider:physical_network physnet2 \
  --provider:segmentation_id 101
$ quantum subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.168.101.0/24
$ quantum router-interface-add router01 net01_subnet01
```

Similarly, for `tenantB`, create a router and another network, using VLAN ID 102 on the physical switch:

```
$ tenant=$(keystone tenant-list|awk '/tenantB/ {print $2}')
```

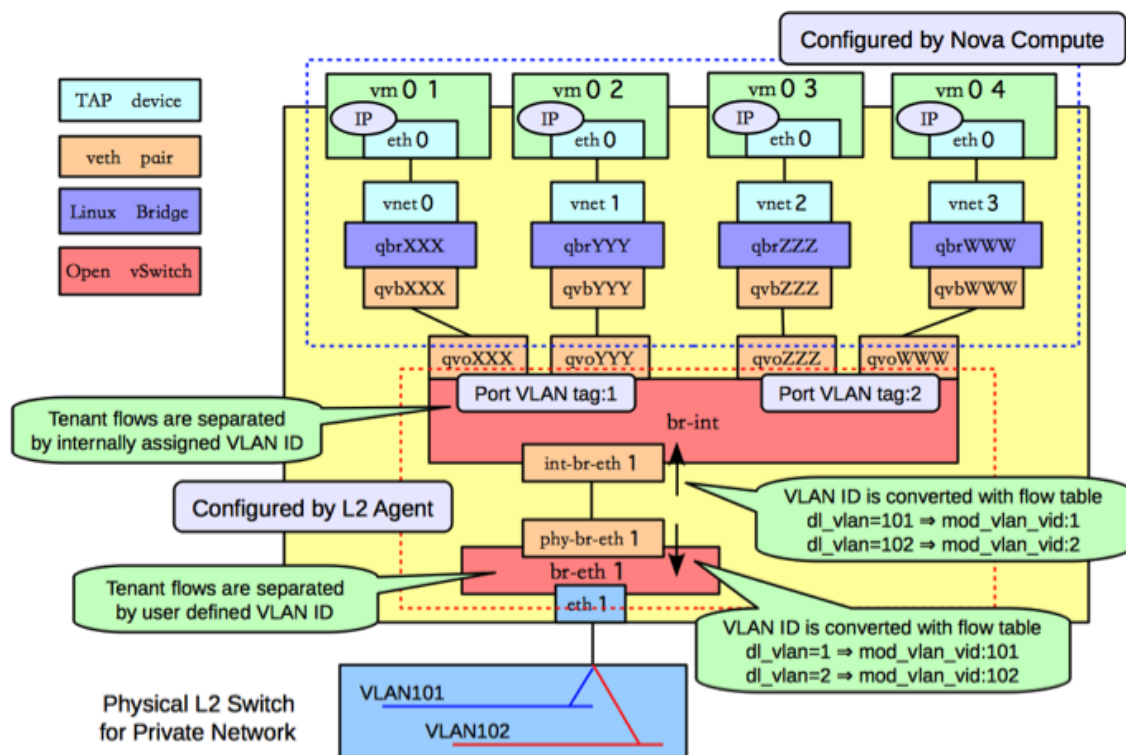
```

$ quantum router-create --tenant-id $tenant router02
$ quantum router-gateway-set router02 public01
$ quantum net-create --tenant-id $tenant net02 \
--provider:network_type vlan \
--provider:physical_network physnet2 \
--provider:segmentation_id 102
$ quantum subnet-create --tenant-id $tenant --name net02_subnet01 net01 192.168.101.0/24
$ quantum router-interface-add router02 net02_subnet01

```

Scenario 2: Compute host config

The following figure shows how the various Linux networking devices would be configured on the compute host under this scenario.

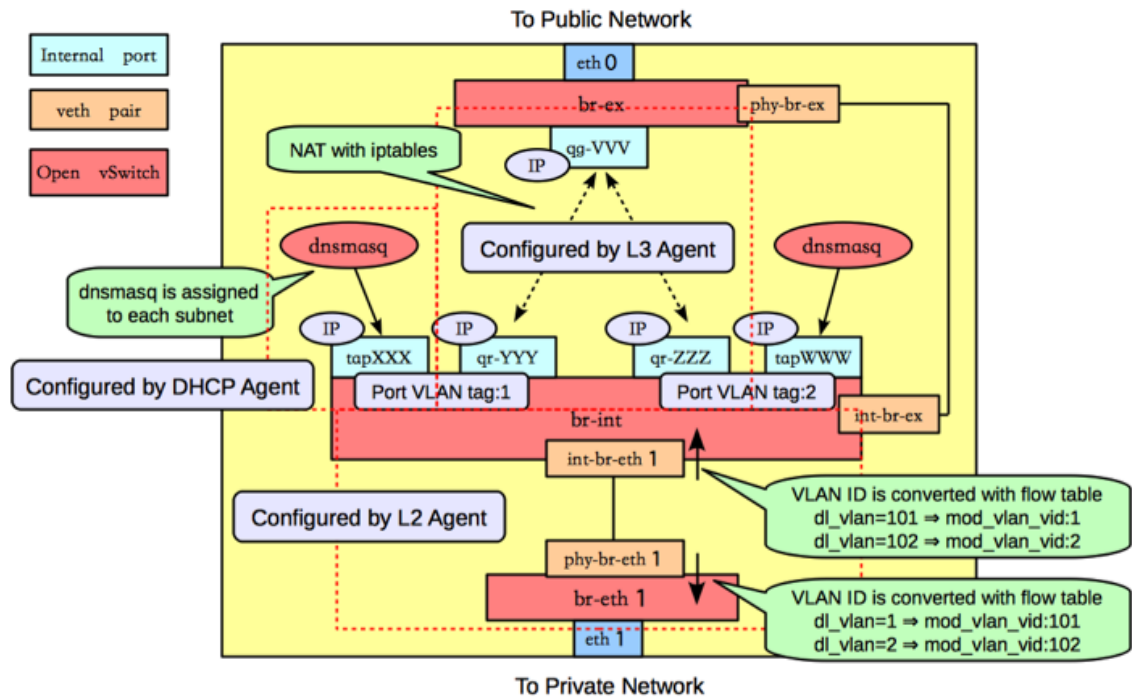


Note

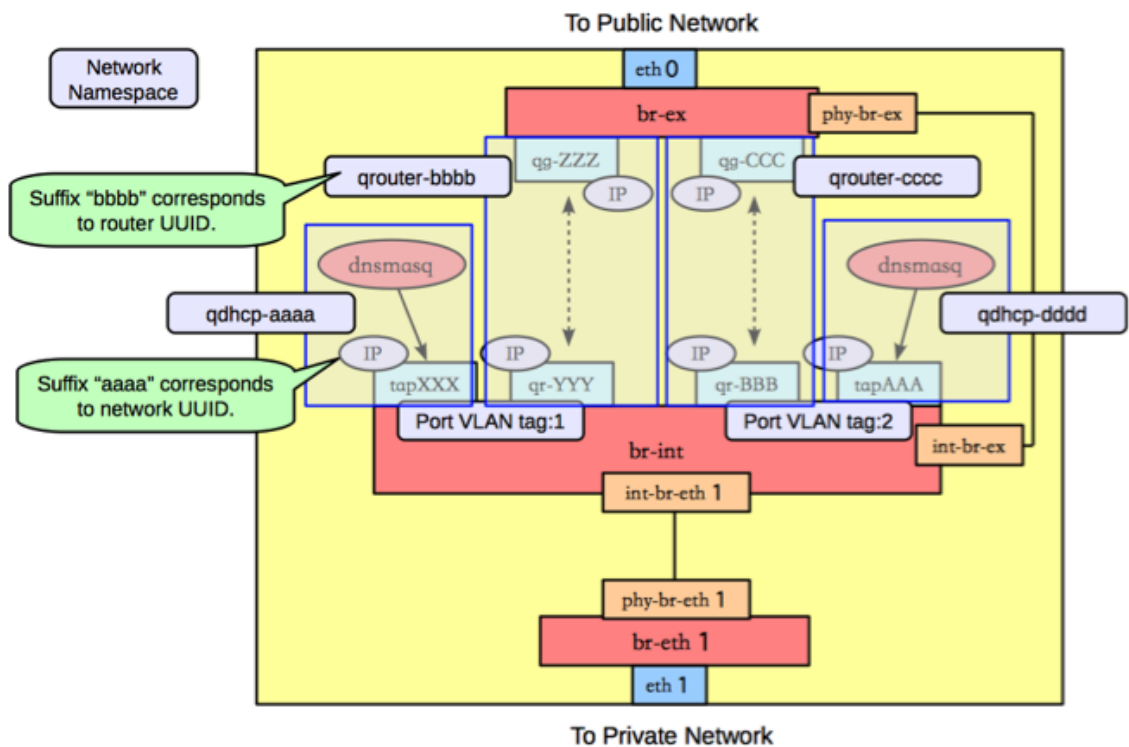
The configuration on the compute host is very similar to the configuration in scenario 1. The only real difference is that scenario 1 had a guest that was connected to two subnets, and in this scenario the subnets belong to different tenants.

Scenario 2: Network host config

The following figure shows the network devices on the network host for the second scenario.



The main difference between the configuration in this scenario and the previous one is the organization of the network namespaces, in order to provide isolation across the two subnets, as shown in the following figure.



In this scenario, there are four network namespaces (`qdhcp-aaa`, `qrouter-bbbb`, `qrouter-cccc`, and `qdhcp-dddd`), instead of three. Since there is no connectivity between the two networks, and so each router is implemented by a separate namespace.

Linux bridge

This section describes how the Linux bridge plugin implements the OpenStack Networking abstractions. It does not discuss the DHCP or L3 agents in detail in this section, because they are covered in the previous section.

Configuration

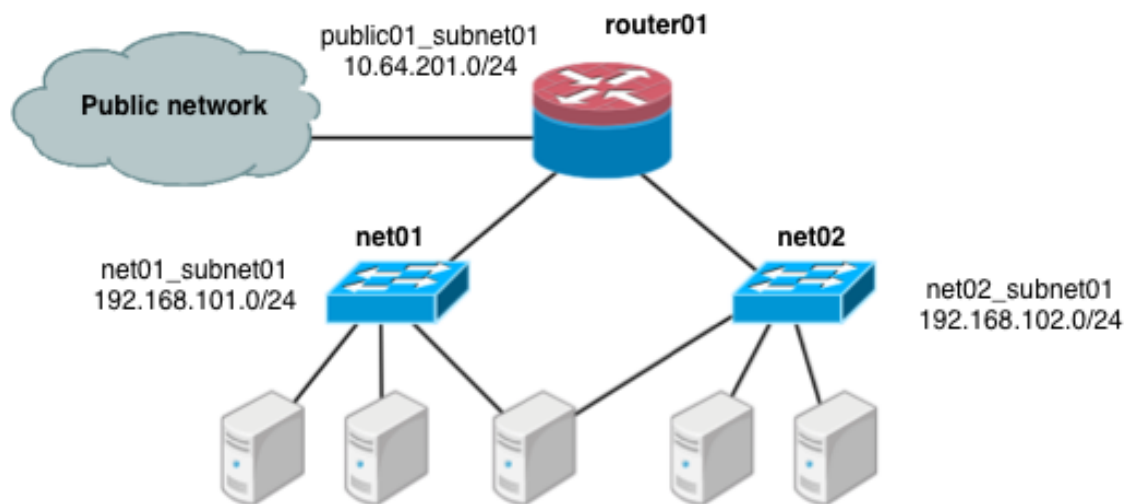
This example uses VLAN isolation on the switches to isolate tenant networks. This configuration labels the physical network associated with the public network as `physnet1`, and the physical network associated with the data network as `physnet2`, which leads to the following configuration options in `linuxbridge_conf.ini`:

```
[VLAN]
tenant_network_type = vlan
network_vlan_ranges = physnet2:100:110

[LINUX_BRIDGE]
physical_interface_mappings: physnet2:eth1
```

Scenario 1: one tenant, two networks, one router

The first scenario has two private networks (`net01`, and `net02`), each with one subnet (`net01_subnet01`: `192.168.101.0/24`, `net02_subnet01`, `192.168.102.0/24`). Both private networks are attached to a router that contains them to the public network (`10.64.201.0/24`).



Under the `service` tenant, create the shared router, define the public network, and set it as the default gateway of the router

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ quantum router-create router01
$ quantum net-create --tenant-id $tenant public01 \
```

```

--provider:network_type flat \
--provider:physical_network physnet1 \
--router:external=True
$ quantum subnet-create --tenant-id $tenant --name public01_subnet01 \
--gateway 10.64.201.254 public01 10.64.201.0/24 --enable_dhcp False
$ quantum router-gateway-set router01 public01

```

Under the `demo` user tenant, create the private network `net01` and corresponding subnet, and connect it to the `router01` router. Configure it to use VLAN ID 101 on the physical switch.

```

$ tenant=$(keystone tenant-list|awk '/demo/ {print $2}')
$ quantum net-create --tenant-id $tenant net01 \
--provider:network_type vlan \
--provider:physical_network physnet2 \
--provider:segmentation_id 101
$ quantum subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.168.101.0/24
$ quantum router-interface-add router01 net01_subnet01

```

Similarly, for `net02`, using VLAN ID 102 on the physical switch:

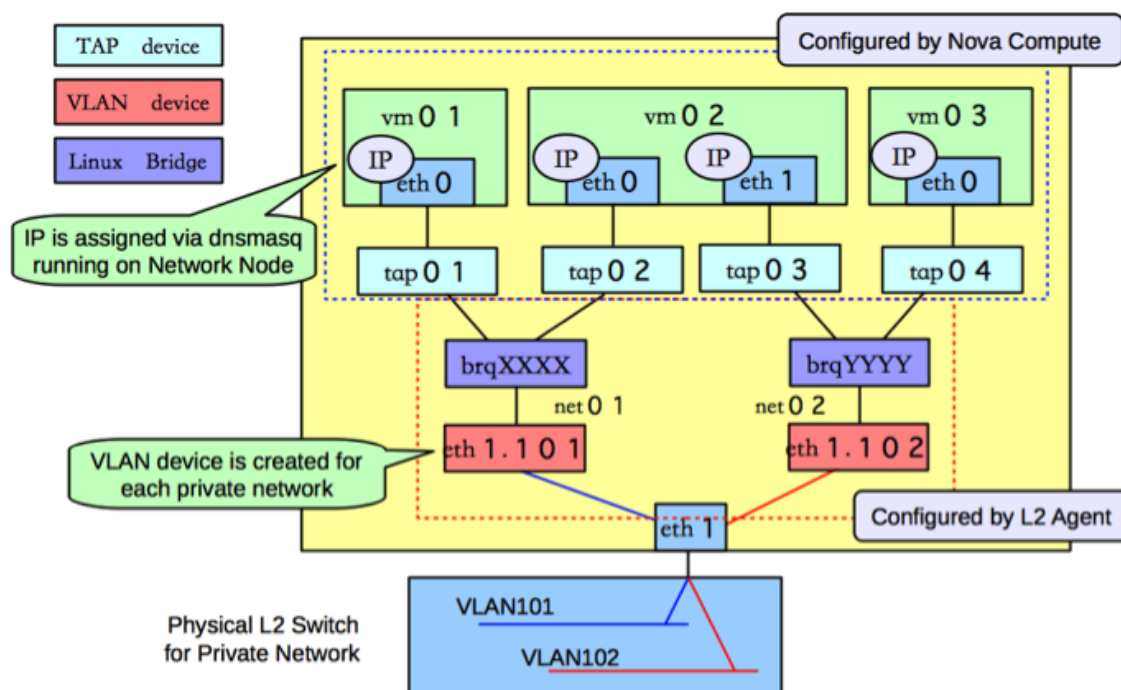
```

$ quantum net-create --tenant-id $tenant net02 \
--provider:network_type vlan \
--provider:physical_network physnet2 \
--provider:segmentation_id 102
$ quantum subnet-create --tenant-id $tenant --name net02_subnet01 net02 192.168.102.0/24
$ quantum router-interface-add router01 net02_subnet01

```

Scenario 1: Compute host config

The following figure shows how to configure the various Linux networking devices on the compute host.



Types of network devices



Note

There are three distinct type of virtual networking devices: TAP devices, VLAN devices, and Linux bridges. For an ethernet frame to travel from `eth0` of virtual machine `vm01`, to the physical network, it must pass through four devices inside of the host: TAP `vnet0`, Linux bridge `brqXXX`, VLAN `eth1.101`), and, finally, the physical network interface card `eth1`.

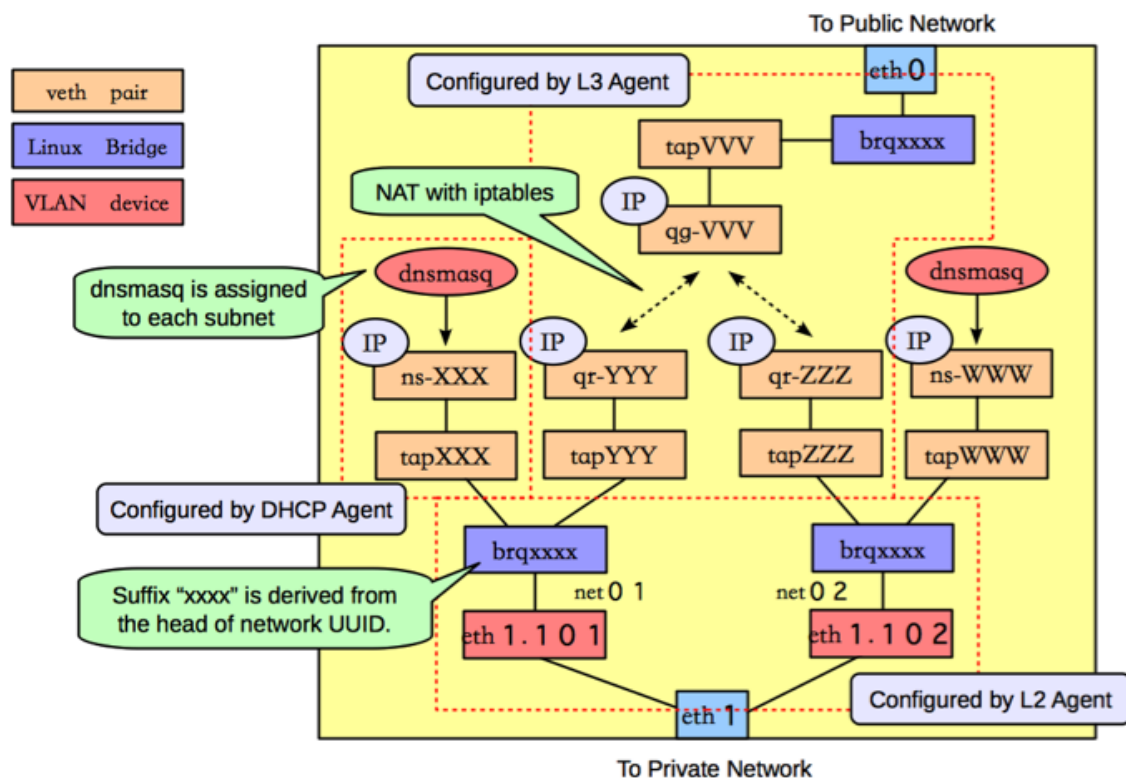
A *TAP device*, such as `vnet0` is how hypervisors such as KVM and Xen implement a virtual network interface card (typically called a VIF or vNIC). An ethernet frame sent to a TAP device is received by the guest operating system.

A *VLAN device* is associated with a VLAN tag attaches to an existing interface device and adds or removes VLAN tags. In the preceding example, VLAN device `eth1.101` is associated with VLAN ID 101 and is attached to interface `eth1`. Packets received from the outside by `eth1` with VLAN tag 101 will be passed to device `eth1.101`, which will then strip the tag. In the other direction, any ethernet frame sent directly to `eth1.101` will have VLAN tag 101 added and will be forward to `eth1` for sending out to the network.

A *Linux bridge* behaves like a hub: you can connect multiple (physical or virtual) network interfaces devices to a Linux bridge. Any ethernet frames that come in from one interface attached to the bridge is transmitted to all of the other devices.

Scenario 1: Network host config

The following figure shows the network devices on the network host.

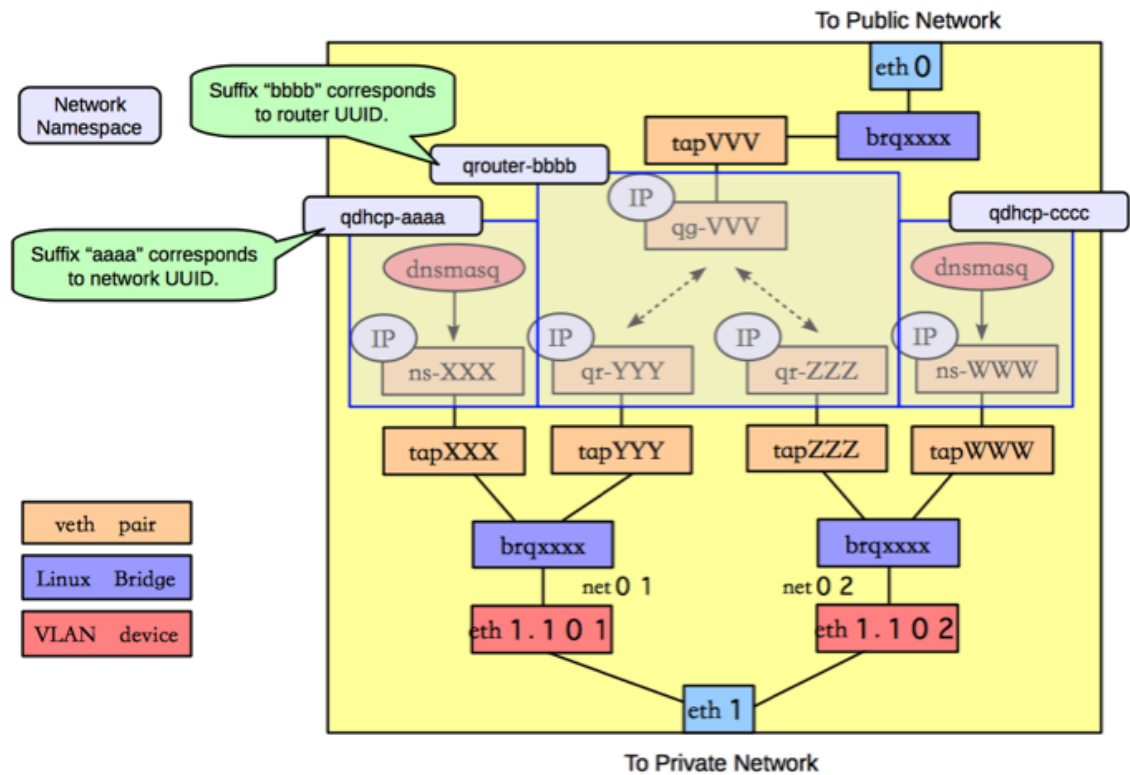


The following figure shows how the Linux bridge plugin uses network namespaces to provide isolation.



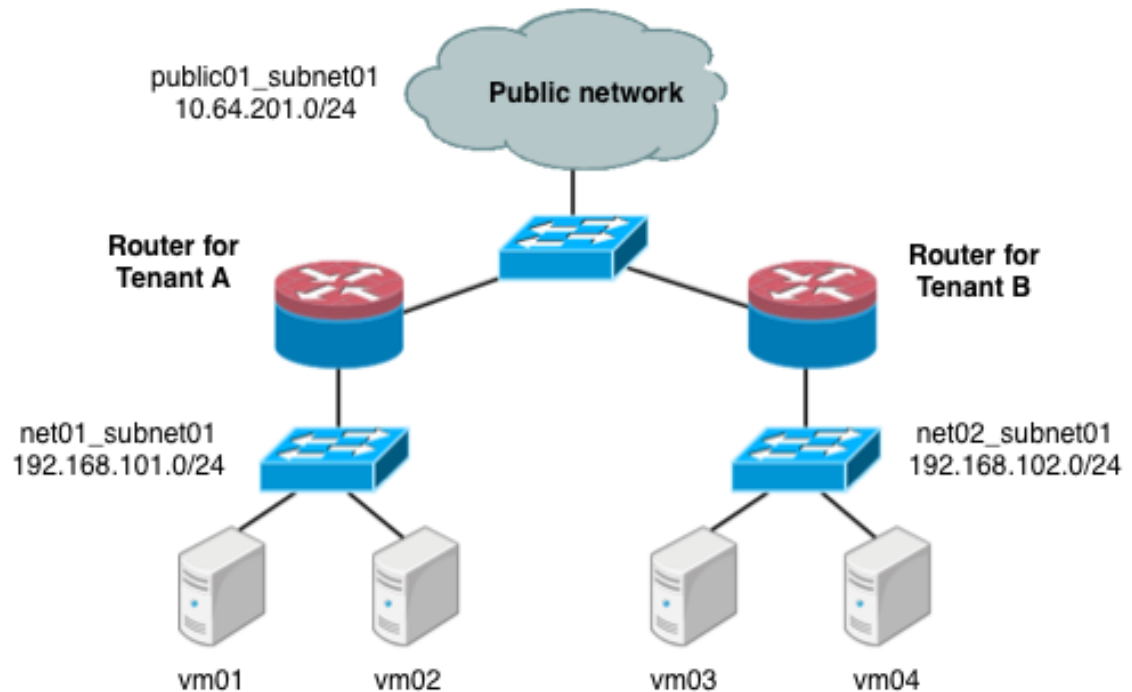
Note

veth pairs form connections between the Linux bridges and the network namespaces.



Scenario 2: two tenants, two networks, two routers

The second scenario has two tenants (A, B). Each tenant has a network with one subnet, and each one has a router that connects them to the public Internet.



Under the `service` tenant, define the public network:

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ quantum net-create --tenant-id $tenant public01 \
  --provider:network_type flat \
  --provider:physical_network physnet1 \
  --router:external=True
$ quantum subnet-create --tenant-id $tenant --name public01_subnet01 \
  --gateway 10.64.201.254 public01 10.64.201.0/24 --enable_dhcp False
```

Under the `tenantA` user tenant, create the tenant router and set its gateway for the public network.

```
$ tenant=$(keystone tenant-list|awk '/tenantA/ {print $2}')
$ quantum router-create --tenant-id $tenant router01
$ quantum router-gateway-set router01 public01
```

Then, define private network `net01` using VLAN ID 102 on the physical switch, along with its subnet, and connect it to the router.

```
$ quantum net-create --tenant-id $tenant net01 \
  --provider:network_type vlan \
  --provider:physical_network physnet2 \
  --provider:segmentation_id 101
$ quantum subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.168.101.0/24
$ quantum router-interface-add router01 net01_subnet01
```

Similarly, for `tenantB`, create a router and another network, using VLAN ID 102 on the physical switch:

```
$ tenant=$(keystone tenant-list|awk '/tenantB/ {print $2}')
```

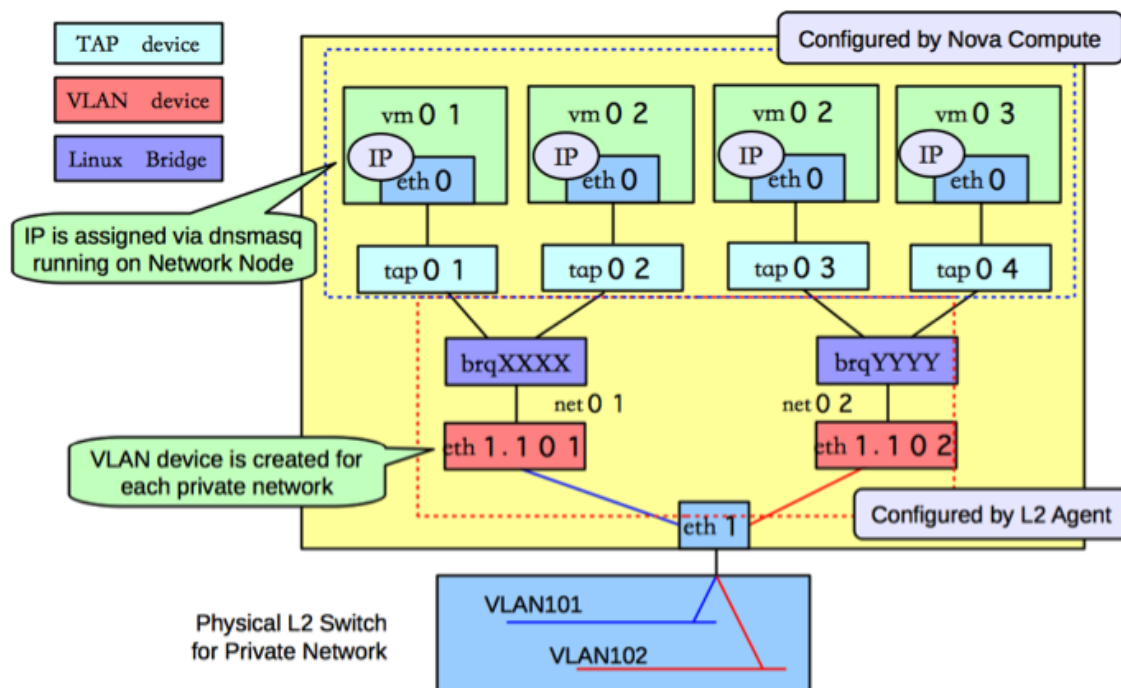
```

$ quantum router-create --tenant-id $tenant router02
$ quantum router-gateway-set router02 public01
$ quantum net-create --tenant-id $tenant net02 \
  --provider:network_type vlan \
  --provider:physical_network physnet2 \
  --provider:segmentation_id 102
$ quantum subnet-create --tenant-id $tenant --name net02_subnet01 net01 192.168.101.0/24
$ quantum router-interface-add router02 net02_subnet01

```

Scenario 2: Compute host config

The following figure shows how the various Linux networking devices would be configured on the compute host under this scenario.

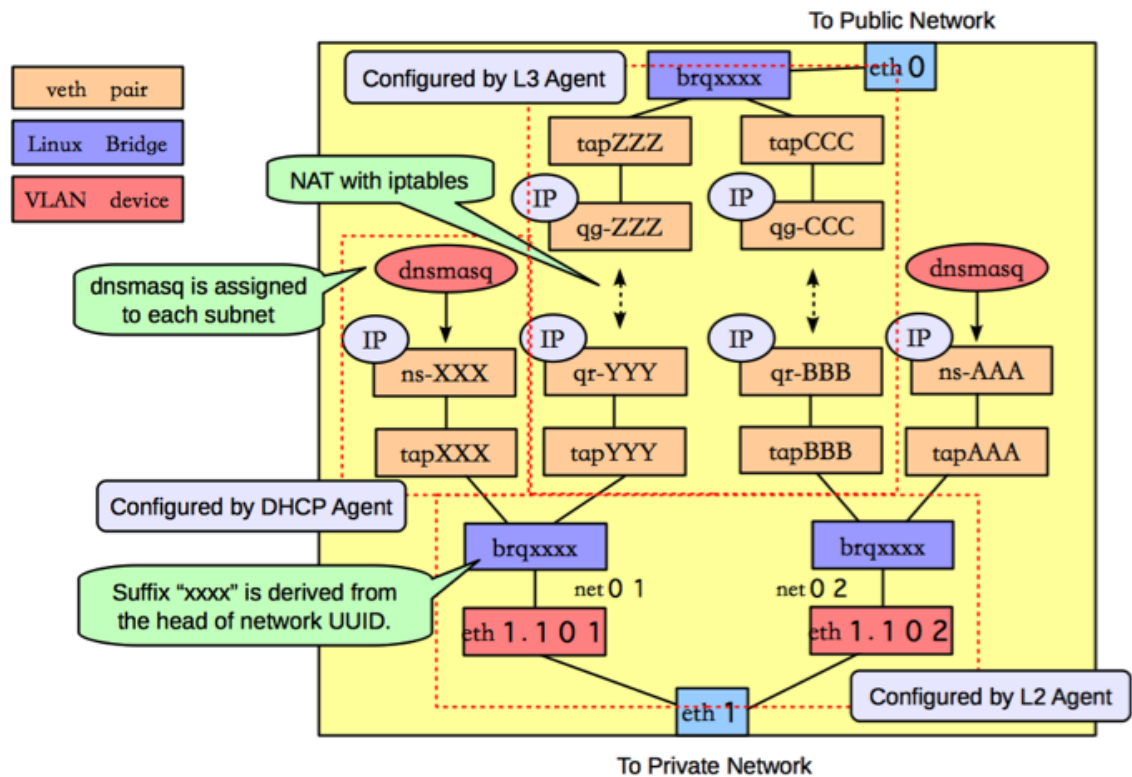


Note

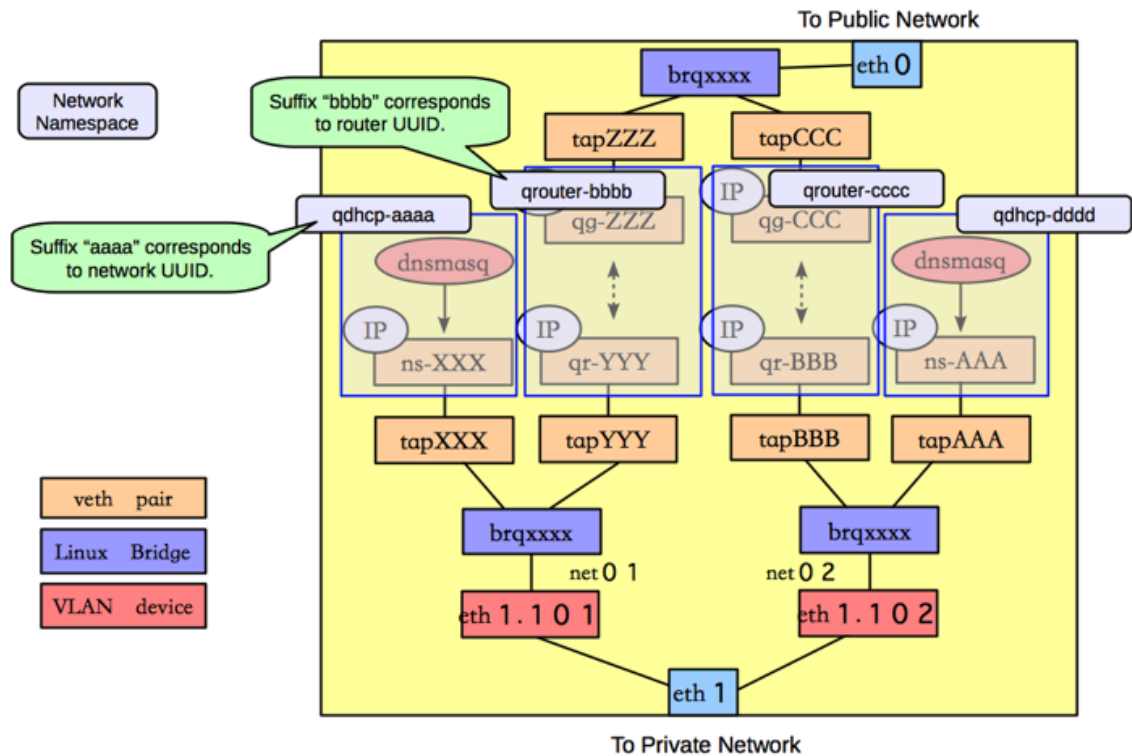
The configuration on the compute host is very similar to the configuration in scenario 1. The only real difference is that scenario 1 had a guest that was connected to two subnets, and in this scenario, the subnets belong to different tenants.

Scenario 2: Network host config

The following figure shows the network devices on the network host for the second scenario.



The main difference between the configuration in this scenario and the previous one is the organization of the network namespaces, in order to provide isolation across the two subnets, as shown in the following figure.



In this scenario, there are four network namespaces (*qdhcp-aaa*, *qrouter-bbbb*, *qrouter-cccc*, and *qdhcp-dddd*), instead of three. Since there is no connectivity between the two networks, and so each router is implemented by a separate namespace.

6. Advanced Features through API Extensions

Provider Networks	54
L3 Routing and NAT	56
Security Groups	59

This section discusses two API extensions implemented by several plugins. We include them in this guide as they provide capabilities similar to what was available in nova-network and are thus likely to be relevant to a large portion of the OpenStack community.

Provider Networks

Provider networks allow cloud administrators to create OpenStack Networking networks that map directly to physical networks in the data center. This is commonly used to give tenants direct access to a "public" network that can be used to reach the Internet. It may also be used to integrate with VLANs in the network that already have a defined meaning (e.g., allow 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 OpenStack Networking virtual networks and underlying physical mechanisms such as VLANs and tunnels. When this extension is supported, OpenStack 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 openvswitch and linuxbridge plugins. Configuration of these plugins requires familiarity with this extension.

Terminology

A number of terms are used in the provider extension and in the configuration of plugins supporting the provider extension:

- **virtual network** - An OpenStack Networking L2 network (identified by a UUID and optional name) whose ports can be attached as vNICs to OpenStack Compute instances and to various OpenStack Networking agents. The openvswitch and linuxbridge plugins each support several different mechanisms to realize virtual networks.
- **physical network** - A network connecting virtualization hosts (i.e. OpenStack Compute nodes) with each other and with other network resources. Each physical network may support multiple virtual networks. The provider extension and the plugin configurations identify physical networks using simple string names.
- **tenant network** - A "normal" virtual network created by/for a tenant. The tenant is not aware of how that network is physically realized.
- **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. Tenants can be given access to provider networks.

- **VLAN network** - A virtual network realized 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 realized 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 may have other uses.
- **GRE network** - A virtual network realized as network packets encapsulated using GRE. GRE networks are also referred to as "tunnels". GRE tunnel packets are routed by the host's IP routing table, so GRE networks are not associated by OpenStack Networking with specific physical networks.

Both the openvswitch and linuxbridge plugins support VLAN networks, flat networks, and local networks. Only the openvswitch plugin currently supports GRE networks, provided that the host's Linux kernel supports the required Open vSwitch features.

Provider Attributes

The provider extension extends the OpenStack Networking network resource with the following three additional attributes:

Table 6.1. Provider Network Attributes

Attribute name	Type	Default Value	Description
provider:network_type	String	N/A	The physical mechanism by which the virtual network is realized. Possible values are "flat", "vlan", "local", and "gre", corresponding to flat networks, VLAN networks, local networks, and GRE networks as defined above. All types of provider networks can be created by administrators, while tenant networks can be realized as "vlan", "gre", or "local" network types depending on plugin 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 realized for flat and VLAN networks. Not applicable to the "local" 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.

The provider attributes are returned by OpenStack Networking API operations when the client is authorized for the `extension:provider_network:view`

action via the OpenStack Networking policy configuration. The provider attributes are only accepted for network API operations if the client is authorized for the `extension:provider_network:set` action. The default OpenStack Networking API policy configuration authorizes both actions for users with the admin role. See [Chapter 8, "Authentication and Authorization" \[68\]](#) for details on policy configuration.

Provider API Workflow

Show all attributes of a network, including provider attributes when invoked with the admin role:

```
quantum net-show <name or net-id>
```

Create a local provider network (admin-only):

```
quantum net-create <name> --tenant_id <tenant-id> --provider:network_type  
local
```

Create a flat provider network (admin-only):

```
quantum net-create <name> --tenant_id <tenant-id> --provider:network_type flat  
--provider:physical_network <phys-net-name>
```

Create a VLAN provider network (admin-only):

```
quantum net-create <name> --tenant_id <tenant-id> --provider:network_type vlan  
--provider:physical_network <phys-net-name> --provider:segmentation_id <VID>
```

Create a GRE provider network (admin-only):

```
quantum net-create <name> --tenant_id <tenant-id> --provider:network_type gre  
--provider:segmentation_id <tunnel-id>
```

When creating flat networks or VLAN networks, `<phys-net-name>` must be known to the plugin. See [the section called "ovs_quantum_plugin.ini" \[124\]](#) and [the section called "linuxbridge_conf.ini" \[126\]](#) for details on configuring `network_vlan_ranges` to identify all physical networks. When creating VLAN networks, `<VID>` can fall either within or outside any configured ranges of VLAN IDs from which tenant networks are allocated. Similarly, when creating GRE networks, `<tunnel-id>` can fall either within or outside any tunnel ID ranges from which tenant networks are allocated.

Once provider networks have been created, subnets can be allocated and they can be used similarly to other virtual networks, subject to authorization policy based on the specified `<tenant_id>`.

L3 Routing and NAT

Just like the core OpenStack Networking API provides abstract L2 network segments that are decoupled from the technology used to implement the L2 network, OpenStack Networking includes an API extension that provides abstract L3 routers that API users can dynamically provision and configure. These OpenStack Networking routers can connect multiple L2 OpenStack Networking networks, and can also provide a "gateway" that connects one or more private L2 networks to a shared "external" network (e.g., a public network for access to the Internet). See [the section called "Use Case: Provider Router with](#)

[Private Networks](#) [9] and [the section called "Use Case: Per-tenant Routers with Private Networks"](#) [10] for details on common models of deploying OpenStack 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 tenant to selectively expose VMs on private networks to other hosts on the external network (and often to all hosts on the Internet). Floating IPs can be allocated and then mapped from one OpenStack Networking port to another, as needed.

L3 API Abstractions

Table 6.2. Router

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the router.
name	String	None	Human-readable name for the router. Might not be unique.
admin_state_up	Bool	True	The administrative state of router. If false (down), the router does not forward packets.
status	String	N/A	Indicates whether router is currently operational.
tenant_id	uuid-str	N/A	Owner of the router. Only admin users can specify a tenant_id other than its own.
external_gateway_info	dict contain 'network_id' key-value pair	Null	External network that this router connects to for gateway services (e.g., NAT)

Table 6.3. Floating IP

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the floating IP.
floating_ip_address	string (IP address)	allocated by OpenStack Networking	The external network IP address available to be mapped to an internal IP address.
floating_network_id	uuid-str	N/A	The network indicating the set of subnets from which the floating IP should be allocated
router_id	uuid-str	N/A	Read-only value indicating the router that connects the external network to the associated internal port, if a port is associated.
port_id	uuid-str	Null	Indicates the internal OpenStack Networking port associated with the external floating IP.
fixed_ip_address	string (IP address)	Null	Indicates the IP address on the internal port that is mapped to by the floating IP (since an OpenStack Networking port might have more than one IP address).
tenant_id	uuid-str	N/A	Owner of the Floating IP. Only admin users can specify a tenant_id other than its own.

Common L3 Workflow

Create external networks (admin-only)


```
quantum net-create public --router:external=True
quantum subnet-create public 172.16.1.0/24
```

Viewing external networks:

```
quantum net-list -- --router:external=True
```

Creating routers

Internal-only router to connect multiple L2 networks privately.

```
quantum net-create net1
quantum subnet-create net1 10.0.0.0/24
quantum net-create net2
quantum subnet-create net2 10.0.1.0/24
quantum router-create router1
quantum router-interface-add router1 <subnet1-uuid>
quantum router-interface-add router1 <subnet2-uuid>
```

The router will get an interface with the gateway_ip address of the subnet, and this interface will be attached to a port on the L2 OpenStack Networking network associated with the subnet. The router will also get an gateway interface to the specified external network. This will provide SNAT connectivity to the external network as well as support for floating IPs allocated on that external networks (see below). Commonly an external network maps to a network in the provider

A router can also be connected to an “external network”, allowing that router to act as a NAT gateway for external connectivity.

```
quantum router-gateway-set router1 <ext-net-id>
```

Viewing routers:

List all routers:

```
quantum router-list
```

Show a specific router:

```
quantum router-show <router_id>
```

Show all internal interfaces for a router:

```
quantum port-list -- --device_id=<router_id>
```

Associating / Disassociating Floating IPs:

First, identify the port-id representing the VM NIC that the floating IP should map to:

```
quantum port-list -c id -c fixed_ips -- --device_id=ZZZ
```

This port must be on an OpenStack Networking subnet that is attached to a router uplinked to the external network that will be used to create the floating IP. Conceptually, this is because the router must be able to perform the Destination NAT (DNAT) rewriting of packets from the Floating IP address (chosen from a subnet on the external network) to the internal Fixed IP (chosen from a private subnet that is “behind” the router).

Create floating IP unassociated, then associate

```
quantum floatingip-create <ext-net-id>  
quantum floatingip-associate <floatingip-id> <internal VM port-id>
```

create floating IP and associate in a single step

```
quantum floatingip-create --port_id <internal VM port-id> <ext-net-id>
```

Viewing Floating IP State:

```
quantum floatingip-list
```

Find floating IP for a particular VM port:

```
quantum floatingip-list -- --port_id=ZZZ
```

Disassociate a Floating IP:

```
quantum floatingip-disassociate <floatingip-id>
```

L3 Tear Down

Delete the Floating IP:

```
quantum floatingip-delete <floatingip-id>
```

Then clear the gateway:

```
quantum router-gateway-clear router1
```

Then remove the interfaces from the router (deleting the network and subnet will do this as well):

```
quantum router-interface-delete router1 <subnet-id>
```

Finally, delete the router:

```
quantum router-delete router1
```

Security Groups

Security groups and security group rules allows administrators and tenants the ability 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 OpenStack Networking it is associated with a security group. If a security group is not specified the port will be associated with a 'default' security group. By default this group will drop all ingress traffic and allow all egress. Rules can be added to this group in order to change the behaviour.

If one desires to use the OpenStack Compute security group APIs and/or have OpenStack Compute orchestrate the creation of new ports for instances on specific security groups, additional configuration is needed. To enable this, one must configure the following file `/etc/nova/nova.conf` and set the config option `security_group_api=quantum` on every node running nova-compute and nova-api. After this change is made restart nova-api and nova-compute in order to pick up this change. After this change is made one will be able to

use both the OpenStack Compute and OpenStack Network security group API at the same time.



Note

In order to use the OpenStack Compute security group API with OpenStack Networking the OpenStack Networking plugin must implement the security group api. The following plugins currently implement this: Nicira NVP, Open vSwitch, Linux Bridge, NEC, and Ryu.



Note

When using the security group API through OpenStack Compute, security groups are applied to all ports on an instance. The reason for this is that OpenStack Compute security group APIs are instances based and not port based as OpenStack Networking.

Security Group API Abstractions

Table 6.4. Security Group Attributes

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the security group.
name	String	None	Human-readable name for the security group. Might not be unique. Cannot be named default as that is automatically created for a tenant.
description	String	None	Human-readable description of a security group.
tenant_id	uuid-str	N/A	Owner of the security group. Only admin users can specify a tenant_id other than their own.

Table 6.5. Security Group Rules

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the security group rule.
security_group_id	uuid-str or Integer	allocated by OpenStack Networking	The security group to associate rule with.
direction	String	N/A	The direction the traffic is allow (ingress/egress) from a VM.
protocol	String	None	IP Protocol (icmp, tcp, udp, etc).
port_range_min	Integer	None	Port at start of range
port_range_max	Integer	None	Port at end of range
ethertype	String	None	ethertype in L2 packet (IPv4, IPv6, etc)
remote_ip_prefix	string (IP cidr)	None	CIDR for address range
remote_group_id	uuid-str or Integer	allocated by OpenStack Networking or OpenStack Compute	Source security group to apply to rule.
tenant_id	uuid-str	N/A	Owner of the security group rule. Only admin users can specify a tenant_id other than its own.

Common Security Group Commands

Create a security group for our web servers:

```
quantum security-group-create webserver --description "security group for webserver"
```

Viewing security groups:

```
quantum security-group-list
```

Creating security group rule to allow port 80 ingress:

```
quantum security-group-rule-create --direction ingress --protocol tcp --port_range_min 80 --port_range_max 80 <security_group_uuid>
```

List security group rules:

```
quantum security-group-rule-list
```

Delete a security group rule:

```
quantum security-group-rule-delete <security_group_rule_uuid>
```

Delete security group:

```
quantum security-group-delete <security_group_uuid>
```

Create a port and associated two security groups:

```
quantum port-create --security-group <security_group_id1> --security-group <security_group_id2> <network_id>
```

Remove security groups from a port:

```
quantum port-update --no-security-groups <port_id>
```

7. Advanced Configuration Options

OpenStack Networking Server with Plugin	62
DHCP Agent	63
L3 Agent	64

This section describes advanced configurations options for various system components (i.e. config options where the default is usually ok, but that the user may want to tweak). After installing from packages, \$QUANTUM_CONF_DIR is /etc/quantum.

OpenStack Networking Server with Plugin

This is the web server that runs the OpenStack Networking API Web Server. It is responsible for loading a plugin and passing the API calls to the plugin for processing. The quantum-server should receive one of more configuration files as its input, for example:

```
quantum-server --config-file <quantum config> --config-file <plugin config>
```

The quantum config contains the common quantum configuration parameters. The plugin config contains the plugin specific flags. The plugin that is run on the service is loaded via the configuration parameter 'core_plugin'. In some cases a plugin may have an agent that performs the actual networking. Specific configuration details can be seen in the Appendix - Configuration File Options.

Most plugins require a SQL database. After installing and starting the database server, set a password for the root account and delete the anonymous accounts:

```
$> mysql -u root
mysql> update mysql.user set password = password('iamroot') where user =
'root';
mysql> delete from mysql.user where user = '';
```

Create a database and user account specifically for plugin:

```
mysql> create database <database-name>;
mysql> create user '<user-name>'@'localhost' identified by '<user-name>';
mysql> create user '<user-name>'@'%' identified by '<user-name>';
mysql> grant all on <database-name>.* to '<user-name>'@'%';
```

Once the above is done you can update the settings in the relevant plugin configuration files. The plugin specific configuration files can be found at \$QUANTUM_CONF_DIR/plugins.

Some plugins have a L2 agent that performs the actual networking. That is, the agent will attach the virtual machine NIC to the OpenStack Networking network. Each node should have an L2 agent running on it. Note that the agent receives the following input parameters:

```
quantum-plugin-agent --config-file <quantum config> --config-file <plugin
config>
```

Two things need to be done prior to working with the plugin:

1. Ensure that the core plugin is updated.

2. Ensure that the database connection is correctly set.

The table below contains examples for these settings. Some Linux packages may provide installation utilities that configure these.

Table 7.1. Settings

Parameter	Value
Open vSwitch	
core_plugin (\$QUANTUM_CONF_DIR/quantum.conf)	quantum.plugins.openvswitch.ovs_quantum_plugin.OVSQuantumPluginV2
sql_connection (in the plugin configuration file)	mysql://<username>:<password>@localhost/ovs_quantum?charset=utf8
Plugin Configuration File	\$QUANTUM_CONF_DIR/plugins/openvswitch/ovs_quantum_plugin.ini
Agent	quantum-openvswitch-agent
Linux Bridge	
core_plugin (\$QUANTUM_CONF_DIR/quantum.conf)	quantum.plugins.linuxbridge.lb_quantum_plugin.LinuxBridgePluginV2
sql_connection (in the plugin configuration file)	mysql://<username>:<password>@localhost/quantum_linux_bridge?charset=utf8
Plugin Configuration File	\$QUANTUM_CONF_DIR/plugins/linuxbridge/linuxbridge_conf.ini
Agent	quantum-linuxbridge-agent

All of the plugin configuration files options can be found in the [Appendix - Configuration File Options](#).

DHCP Agent

There is an option to run a DHCP server that will allocate IP addresses to virtual machines running on the network. When a subnet is created, by default, the subnet has DHCP enabled.

The node that runs the DHCP agent should run:

```
quantum-dhcp-agent --config-file <quantum config>
--config-file <dhcp config>
```

Currently the DHCP agent uses dnsmasq to perform that static address assignment.

A driver needs to be configured that matches the plugin running on the service.

Table 7.2. Basic settings

Parameter	Value
Open vSwitch	
interface_driver (\$QUANTUM_CONF_DIR/dhcp_agent.ini)	quantum.agent.linux.interface.OVSInterfaceDriver
Linux Bridge	
interface_driver (\$QUANTUM_CONF_DIR/dhcp_agent.ini)	quantum.agent.linux.interface.BridgeInterfaceDriver

All of the DHCP agent configuration options can be found in the [Appendix - Configuration File Options](#).

Namespace

By default the DHCP agent makes use of Linux network namespaces in order to support overlapping IP addresses. Requirements for network namespaces support are described in the [Limitation](#) section.

If the Linux installation does not support network namespace, you must disable using network namespace in the DHCP agent config file (The default value of use_namespaces is True).

```
use_namespaces = False
```

L3 Agent

There is an option to run a L3 agent that will give enable layer 3 forwarding and floating IP support. The node that runs the L3 agent should run:

```
quantum-l3-agent --config-file <quantum config>
--config-file <l3 config>
```

A driver needs to be configured that matches the plugin running on the service. The driver is used to create the routing interface.

Table 7.3. Basic settings

Parameter	Value
Open vSwitch	
interface_driver (\$QUANTUM_CONF_DIR/l3_agent.ini)	quantum.agent.linux.interface.OVSInterfaceDriver
external_network_bridge (\$QUANTUM_CONF_DIR/l3_agent.ini)	br-ex
Linux Bridge	
interface_driver (\$QUANTUM_CONF_DIR/l3_agent.ini)	quantum.agent.linux.interface.BridgeInterfaceDriver
external_network_bridge (\$QUANTUM_CONF_DIR/l3_agent.ini)	This field must be empty (or the bridge name for the external network).

The L3 agent communicates with the OpenStack Networking server via the OpenStack Networking API, so the following configuration is required:

1. OpenStack Identity authentication:

```
auth_url="$KEYSTONE_SERVICE_PROTOCOL://$KEYSTONE_AUTH_HOST:
$KEYSTONE_AUTH_PORT/v2.0"
```

For example,

```
http://10.56.51.210:5000/v2.0
```

2. Admin user details:

```
admin_tenant_name $SERVICE_TENANT_NAME
admin_user $Q_ADMIN_USERNAME
admin_password $SERVICE_PASSWORD
```

All of the L3 agent configuration options can be found in the [Appendix - Configuration File Options](#).

Namespace

By default the L3 agent makes use of Linux network namespaces in order to support overlapping IP addresses. Requirements for network namespaces support are described in the [Limitation](#) section.

If the Linux installation does not support network namespace, you must disable using network namespace in the L3 agent config file (The default value of `use_namespaces` is `True`).

```
use_namespaces = False
```

When `use_namespaces` is set to `False`, only one router ID can be supported per node. This must be configured via the configuration variable `router_id`.

```
# If use_namespaces is set to False then the agent can only configure one
router.
# This is done by setting the specific router_id.
router_id = 1064ad16-36b7-4c2f-86f0-daa2bcbd6b2a
```

To configure it, you need to run the OpenStack Networking service and create a router, and then set an ID of the router created to `router_id` in the L3 agent configuration file.

```
$ quantum router-create myrouter1
Created a new router:
```

Field	Value
admin_state_up	True
external_gateway_info	
id	338d42d7-b22e-42c5-9df6-f3674768fe75
name	myrouter1
status	ACTIVE
tenant_id	0c236f65baa04e6f9b4236b996555d56

Multiple Floating IP Pools

The L3 API in OpenStack Networking supports multiple floating IP pools. In OpenStack Networking, a floating IP pool is represented as an external network and a floating IP is allocated from a subnet associated with the external network. Since each L3 agent can be associated with at most one external network, we need to invoke multiple L3 agent to define multiple floating IP pools. '`gateway_external_network_id`' in L3 agent configuration file indicates the external network that the L3 agent handles. You can run multiple L3 agent instances on one host.

In addition, when you run multiple L3 agents, make sure that `handle_internal_only_routers` is set to `True` only for one L3 agent in an OpenStack Networking deployment and set to `False` for all other L3 agents. Since the default value of this parameter is `True`, you need to configure it carefully.

Before starting L3 agents, you need to create routers and external networks, then update the configuration files with UUID of external networks and start L3 agents.

For the first agent, invoke it with the following l3_agent.ini where handle_internal_only_routers is True.

```
handle_internal_only_routers = True
gateway_external_network_id = 2118b11c-011e-4fa5-a6f1-2ca34d372c35
external_network_bridge = br-ex
```

```
python /opt/stack/quantum/bin/quantum-l3-agent
--config-file /etc/quantum/quantum.conf
--config-file=/etc/quantum/l3_agent.ini
```

For the second (or later) agent, invoke it with the following l3_agent.ini where handle_internal_only_routers is False.

```
handle_internal_only_routers = False
gateway_external_network_id = e828e54c-850a-4e74-80a8-8b79c6a285d8
external_network_bridge = br-ex-2
```

OpenStack Compute Metadata Server Support

To use OpenStack Compute metadata service, metadata_ip and metadata_port in the L3 agent configuration file need to be configured. Accessing from VMs to OpenStack Compute metadata service is forwarded to an external network through OpenStack Networking L3 router. OpenStack Compute metadata service must be reachable from the external network. As the [Limitations section](#) says, note that OpenStack Networking overlapping IPs support and OpenStack Compute metadata service cannot be used together.

Allowing VMs to reach the metadata service is a big point of confusion with OpenStack Networking. We need to make sure instructions for how to set this up are displayed more prominently than they already are, and that there are instructions for how to validate and troubleshoot in this scenario.

Example validation includes:

VALIDATION STEP #1

- on network node(l3_agent running) ping to nova_metadata_ip specified in metadata_agent.ini

if you are not using namespace, just run:

```
ping <metadata_ip>
```

for example, if the metadata server IP is 172.16.10.5, run:

```
$ping 172.16.10.5
```

if you are using namespaces, identify the UUID of the router and run:

```
ip netns exec qrouter-<router uuid> ping <metadata_ip>
```

for example, if the router uuid is d7e9ec57-77c2-4046-aebf-d978ed4a4f83 and the metadata server IP is 172.16.10.5, run:

```
$ ip netns exec qrouter-d7e9ec57-77c2-4046-aebf-d978ed4a4f83 ping 172.16.10.5
```

VALIDATION STEP #2

- on metadata server(nova_api server) check connection to vm's subnets using an un-NATed IP address of the VM, such as 10.0.0.2:

```
$ping 10.0.0.2
```

NOTE

OpenStack does not manage this routing for you, so you need to make sure that your host running the metadata service always has a route to reach each private network's subnet via the external network IP of that subnet's OpenStack Networking router. To do this, you can either run OpenStack Networking without namespaces, and run the quantum-l3-agent on the same host as nova-api. Otherwise, you can identify an IP prefix that includes all private network subnet's (e.g., 10.0.0.0/8) and then make sure that your metadata server has a route for that prefix with the OpenStack Networking router's external IP address as the next hop.

8. Authentication and Authorization

OpenStack Networking uses the OpenStack Identity service (project name keystone) as the default authentication service. When OpenStack Identity is enabled Users submitting requests to the OpenStack Networking service must provide an authentication token in X-Auth-Token request header. The aforementioned token should have been obtained by authenticating with the OpenStack Identity endpoint. For more information concerning authentication with OpenStack Identity, please refer to the OpenStack Identity documentation. When OpenStack Identity is enabled, it is not mandatory to specify `tenant_id` for resources in create requests, as the tenant identifier will be derived from the Authentication token. Please note that the default authorization settings only allow administrative users to create resources on behalf of a different tenant. OpenStack Networking uses information received from OpenStack Identity to authorize user requests. OpenStack 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:** whether access to specific resource might be granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in OpenStack 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. That is to say, every time the policy file is updated, the policies will be automatically reloaded. Currently the only way of updating such policies is to edit the policy file. Please note that in this section we will use both the terms "policy" and "rule" to refer to objects which are specified in the same way in the policy file; in other words, there are no syntax differences between a rule and a policy. We will define a policy something which is matched directly from the OpenStack Networking policy engine, whereas we will define a rule as the elements of such policies which are then evaluated. For instance in `create_subnet: [["admin_or_network_owner"]]`, `create_subnet` is regarded as a policy, whereas `admin_or_network_owner` is regarded as a rule.

Policies are triggered by the OpenStack Networking policy engine whenever one of them matches an OpenStack 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 OpenStack 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 be also related to specific API extensions; for instance `extension:provider_network:set` will be 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, evaluation policy will be successful 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 OpenStack Networking policy engine currently defines the following kinds of terminal rules:

- **Role-based rules:** evaluate successfully if the user submitting the request has the specified role. For instance `"role:admin"` is successful if the user submitting 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 attribute *shared* of the *network* resource is set to true.
- **Generic rules:** compare an attribute in the resource with an attribute extracted from the user's security credentials and evaluates successfully if the comparison is successful. For instance `"tenant_id:%(tenant_id)s"` is successful if the tenant identifier in the resource is equal to the tenant identifier of the user submitting the request.

The following is an extract from the default policy.json file:

```
{
  "admin_or_owner": [["role:admin"], ["tenant_id:%(tenant_id)s"]],
  [1]
  "admin_or_network_owner": [["role:admin"], ["tenant_id:
%(network_tenant_id)s"]],
  "admin_only": [["role:admin"]], "regular_user": [],
  "shared": [["field:networks:shared=True"]],
  "default": [["rule:admin_or_owner"]], [2]
  "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"], ["rule:shared"]], [3]
  "create_network:shared": [["rule:admin_only"]], [4]
  "update_network": [["rule:admin_or_owner"]],
  "delete_network": [["rule:admin_or_owner"]],
  "create_port": [],
  "create_port:mac_address": [["rule:admin_or_network_owner"]], [5]
  "create_port:fixed_ips": [["rule:admin_or_network_owner"]],
```

```
"get_port": [ ["rule:admin_or_owner"] ],  
"update_port": [ ["rule:admin_or_owner"] ],  
"delete_port": [ ["rule:admin_or_owner"] ]  
}
```

[1] is a rule which evaluates successfully if the current user is an administrator or the owner of the resource specified in the request (tenant identifier is equal).

[2] is the default policy which is always evaluated if an API operation does not match any of the policies in policy.json.

[3] This policy will evaluate successfully if either *admin_or_owner*, or *shared* evaluates successfully.

[4] This policy will restrict the ability of manipulating the *shared* attribute for a network to administrators only.

[5] This policy will restrict the ability of manipulating the *mac_address* attribute for a port only to administrators and the owner of the network where the port is attached.

In some cases, some operations should be restricted to administrators only; therefore, as a further example, let us consider how this sample policy file should be modified in a scenario where tenants are allowed only to define networks and see their resources, and all the other operations can be performed only in an administrative context:

```
{  
  "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"}]  
}
```

9. Advanced Operational Features

Logging Settings	72
Notifications	72
Quotas	74

Logging Settings

OpenStack Networking components use Python logging module to do logging. Logging configuration can be provided in `quantum.conf` or as command line options. Command options will override ones in `quantum.conf`.

Two ways to specify the logging configuration for OpenStack Networking components:

1. Provide logging settings in a logging configuration file.

Please see [Python Logging HOWTO](#) for logging configuration file.

2. Provide logging setting in `quantum.conf`

```
[DEFAULT]
# Default log level is WARNING
# Show debugging output in logs (sets DEBUG log level output)
# debug = False

# Show more verbose log output (sets INFO log level output) if debug is
False
# verbose = False

# log_format = %(asctime)s %(levelname)8s [%(name)s] %(message)s
# 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

Notification Options

Notifications can be sent when OpenStack Networking resources such as network, subnet and port are created, updated or deleted. To support DHCP agent, `rpc_notifier` driver must be set. To set up the notification, edit notification options in `quantum.conf`:

```
# ===== Notification System Options =====

# Notifications can be sent when network/subnet/port are create, updated or
# deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = quantum.openstack.common.notifier.no_op_notifier
# Logging driver
# notification_driver = quantum.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = quantum.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic names or to set
# logging level
# default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications
```

Setting Cases

Logging and RPC

The options below will make OpenStack Networking server send notifications via logging and RPC. The logging options are described in [Logging Settings](#). RPC notifications will go to 'notifications.info' queue bound to a topic exchange defined by 'control_exchange' in quantum.conf.

```
# ===== Notification System Options =====

# Notifications can be sent when network/subnet/port are create, updated or
# deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = quantum.openstack.common.notifier.no_op_notifier
# Logging driver
notification_driver = quantum.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = quantum.openstack.common.notifier.rpc_notifier
```



```
# default_notification_level is used to form actual topic names or to set
logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications
```

Multiple RPC Topics

The options below will make OpenStack Networking server send notifications to multiple RPC topics. RPC notifications will go to 'notifications_one.info' and 'notifications_two.info' queues bound to a topic exchange defined by 'control_exchange' in quantum.conf.

```
# ===== Notification System Options =====

# Notifications can be sent when network/subnet/port are create, updated or
deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = quantum.openstack.common.notifier.no_op_notifier
# Logging driver
# notification_driver = quantum.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = quantum.openstack.common.notifier.rabbit_notifier

# default_notification_level is used to form actual topic names or to set
logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications_one,notifications_two
```

Quotas

Quota is a function to limit number of resources. You can enforce default quota for all tenants. You will get error when you try to create more resources than the limit.

```
$ quantum net-create test_net
Quota exceeded for resources: ['network']
```

Per-tenant quota configuration is also supported by quota extension API. See [Per-tenant quota configuration](#) for details.

Basic quota configuration

In OpenStack Networking default quota mechanism, all tenants have a same quota value, i.e., a number of resources that a tenant can create. This is enabled by default.

The value of quota is defined in the OpenStack Networking configuration file (`quantum.conf`). If you want to disable quotas for a specific resource (e.g., network, subnet, port), remove a corresponding item from `quota_items`. Each of the quota values in the example below is the default value.

```
[QUOTAS]
# resource name(s) that are supported in quota features
quota_items = network,subnet,port

# 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 = quantum.quota.ConfDriver
```

OpenStack Networking also supports quotas for L3 resources: router and floating IP. You can configure them by adding the following lines to `QUOTAS` section in `quantum.conf`. (Note that `quota_items` does not affect these quotas.)

```
[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 per security group. You can configure them by adding the following lines to `QUOTAS` section in `quantum.conf`. (Note that `quota_items` does not affect these quotas.)

```
[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
```

Per-tenant quota configuration

OpenStack Networking also supports per-tenant quota limit by quota extension API. To enable per-tenant quota, you need to set `quota_driver` in `quantum.conf`.

```
quota_driver = quantum.db.quota_db.DbQuotaDriver
```

When per-tenant quota is enabled, the output of the following command contains quotas.

```
$ quantum ext-list -c alias -c name
```

alias	name
agent_scheduler	Agent Schedulers
security-group	security-group
binding	Port Binding
quotas	Quota management support
agent	agent
provider	Provider Network
router	Quantum L3 Router
lbaas	LoadBalancing service
extraroute	Quantum Extra Route

```
$ quantum ext-show quotas
```

Field	Value
alias	quotas
description	Expose functions for quotas management per tenant
links	
name	Quota management support
namespace	http://docs.openstack.org/network/ext/quotas-sets/api/v2.0
updated	2012-07-29T10:00:00-00:00



Note

Per-tenant quotas are supported only supported by some plugins. At least Open vSwitch, Linux Bridge, and Nicira NVP are known to work but new versions of other plugins may bring additional functionality - consult the documentation for each plugin.

There are four CLI commands to manage per-tenant quota.

quota-delete	Delete defined quotas of a given tenant.
quota-list	List defined quotas of all tenants.
quota-show	Show quotas of a given tenant
quota-update	Define tenant's quotas not to use defaults.

Only users with 'admin' role can change a quota value. Note that the default set of quotas are enforced for all tenants by default, so there is no `quota-create` command.

`quota-list` displays a list of tenants for which per-tenant quota is enabled. The tenants who have the default set of quota limits are not listed. This command is permitted to only 'admin' users.

```
$ quantum quota-list
```

	floatingip	network	port	router	subnet	tenant_id

```

+-----+-----+-----+-----+-----+
+-----+
|      20 |      5 |    20 |    10 |      5 |
| 6f88036c45344d9999a1f971e4882723 |
|      25 |     10 |    30 |    10 |     10 |
| bff5c9455ee24231b5bc713c1b96d422 |
+-----+-----+-----+-----+
+-----+

```

`quota-show` reports the current set of quota limits for the specified tenant. Regular (non-admin) users can call this command (without `--tenant_id` parameter). If per-tenant quota limits are not defined for the tenant, the default set of quotas are displayed.

```
$ quantum quota-show --tenant_id 6f88036c45344d9999a1f971e4882723
```

```

+-----+-----+
| Field      | Value |
+-----+-----+
| floatingip  | 20    |
| network     | 5     |
| port        | 20    |
| router      | 10    |
| subnet      | 5     |
+-----+-----+

```

The below is an example called by a non-admin user.

```
$ quantum quota-show
```

```

+-----+-----+
| Field      | Value |
+-----+-----+
| floatingip  | 20    |
| network     | 5     |
| port        | 20    |
| router      | 10    |
| subnet      | 5     |
+-----+-----+

```

You can update a quota of the given tenant by `quota-update` command.

Update the limit of network quota.

```
$ quantum quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --network 5
```

```

+-----+-----+
| Field      | Value |
+-----+-----+
| floatingip  | 50    |
| network     | 5     |
| port        | 50    |
| router      | 10    |
| subnet      | 10    |
+-----+-----+

```

You can update quotas of multiple resources in one command.

```
$ quantum quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --subnet 5 --port 20
```

```

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

```

floatingip	50	
network	5	
port	20	
router	10	
subnet	5	
+-----+		

To update the limits of L3 resource (router, floating IP), we need to specify new values of the quotas after '-'. The example below updates the limit of the number of floating IPs for the given tenant.

```
$ quantum quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 -- -- floatingip 20
```

Field	Value	
+-----+		
floatingip	20	
network	5	
port	20	
router	10	
subnet	5	
+-----+		

You can update the limits of multiple resources including L2 resources and L3 resource in one command.

```
$ quantum quota-update --tenant_id 6f88036c45344d9999a1f971e4882723 --network 3 --subnet 3 --port 3 -- --floatingip 3 --router 3
```

Field	Value	
+-----+		
floatingip	3	
network	3	
port	3	
router	3	
subnet	3	
+-----+		

To clear per-tenant quota limits, use `quota-delete`. After `quota-delete`, quota limits enforced to the tenant are reset to the default set of quotas.

```
$ quantum quota-delete --tenant_id 6f88036c45344d9999a1f971e4882723
```

```
Deleted quota: 6f88036c45344d9999a1f971e4882723
```

```
$ quantum quota-show --tenant_id 6f88036c45344d9999a1f971e4882723
```

Field	Value	
+-----+		
floatingip	50	
network	10	
port	50	
router	10	
subnet	10	
+-----+		

10. High Availability

OpenStack Networking High Availability with Pacemaker 79

Several aspects of an OpenStack Networking deployment benefit from high-availability to withstand individual node failures. In general, quantum-server and quantum-dhcp-agent can be run in an active-active fashion. The quantum-l3-agent service can be run only as active/passive, to avoid IP conflicts with respect to gateway IP addresses.

OpenStack Networking High Availability with Pacemaker

You can run some OpenStack Networking services into a cluster (Active / Passive or Active / Active for OpenStack Networking Server only) with Pacemaker.

Here you can download the latest Resources Agents :

- quantum-server: <https://github.com/madkiss/openstack-resource-agents/blob/master/ocf/quantum-server>
- quantum-dhcp-agent : <https://github.com/madkiss/openstack-resource-agents/blob/master/ocf/quantum-agent-dhcp>
- quantum-l3-agent : <https://github.com/madkiss/openstack-resource-agents/blob/master/ocf/quantum-agent-l3>



Note

If you need more informations about "*How to build a cluster*", please refer to [Pacemaker documentation](#).

11. Limitations

- *No equivalent for nova-network --multi_host flag:* Nova-network has a model where the L3, NAT, and DHCP processing happen on the compute node itself, rather than a dedicated networking node. OpenStack Networking now support running multiple l3-agent and dhcp-agents with load being split across those agents, but the tight coupling of that scheduling with the location of the VM is not supported in Grizzly. The Havana release is expected to include an exact replacement for the --multi_host flag in nova-network.
- *Linux network namespace required on nodes running quantum-l3-agent or quantum-dhcp-agent if overlapping IPs are in use:* . In order to support overlapping IP addresses, the OpenStack Networking DHCP and L3 agents use Linux network namespaces by default. The hosts running these processes must support network namespaces. To support network namespaces, the following are required:
 - Linux kernel 2.6.24 or newer (with CONFIG_NET_NS=y in kernel configuration) and
 - iproute2 utilities ('ip' command) version 3.1.0 (aka 20111117) or newer

To check whether your host supports namespaces try running the following as root:

```
ip netns create test-ns
ip netns exec test-ns ifconfig
```

If the preceding commands do not produce errors, your platform is likely sufficient to use the dhcp-agent or l3-agent with namespace. In our experience, Ubuntu 12.04 or later support namespaces as does Fedora 17 and new, but some older RHEL platforms do not by default. It may be possible to upgrade the iproute2 package on a platform that does not support namespaces by default.

If you need to disable namespaces, make sure the `quantum.conf` used by quantum-server has the following setting:

```
allow_overlapping_ips=False
```

and that the `dhcp_agent.ini` and `l3_agent.ini` have the following setting:

```
use_namespaces=False
```



Note

If the host does not support namespaces then the `quantum-l3-agent` and `quantum-dhcp-agent` should be run on different hosts. This is due to the fact that there is no isolation between the IP addresses created by the L3 agent and by the DHCP agent. By manipulating the routing the user can ensure that these networks have access to one another.

If you run both L3 + DHCP services on the same node, you should enable namespaces to avoid conflicts with routes :

```
use_namespaces=True
```

- *No IPv6 support for L3 agent:* The quantum-l3-agent, used by many plugins to implement L3 forwarding, supports only IPv4 forwarding. Currently, There are no errors provided if you configure IPv6 addresses via the API.
- *ZeroMQ support is experimental:* Some agents, including quantum-dhcp-agent, quantum-openvswitch-agent, and quantum-linuxbridge-agent use RPC to communicate. ZeroMQ is an available option in the configuration file, but has not been tested and should be considered experimental. In particular, there are believed to be issues with ZeroMQ and the dhcp agent.
- *MetaPlugin is experimental:* This release includes a "MetaPlugin" that is intended to support multiple plugins at the same time for different API requests, based on the content of those API requests. This functionality has not been widely reviewed or tested by the core team, and should be considered experimental until further validation is performed.

Appendix A. Demos Setup

Table of Contents

Single Flat Network	82
Provider Router with Private Networks	88
Per-tenant Routers with Private Networks	96
Scalable and Highly Available DHCP Agents	110

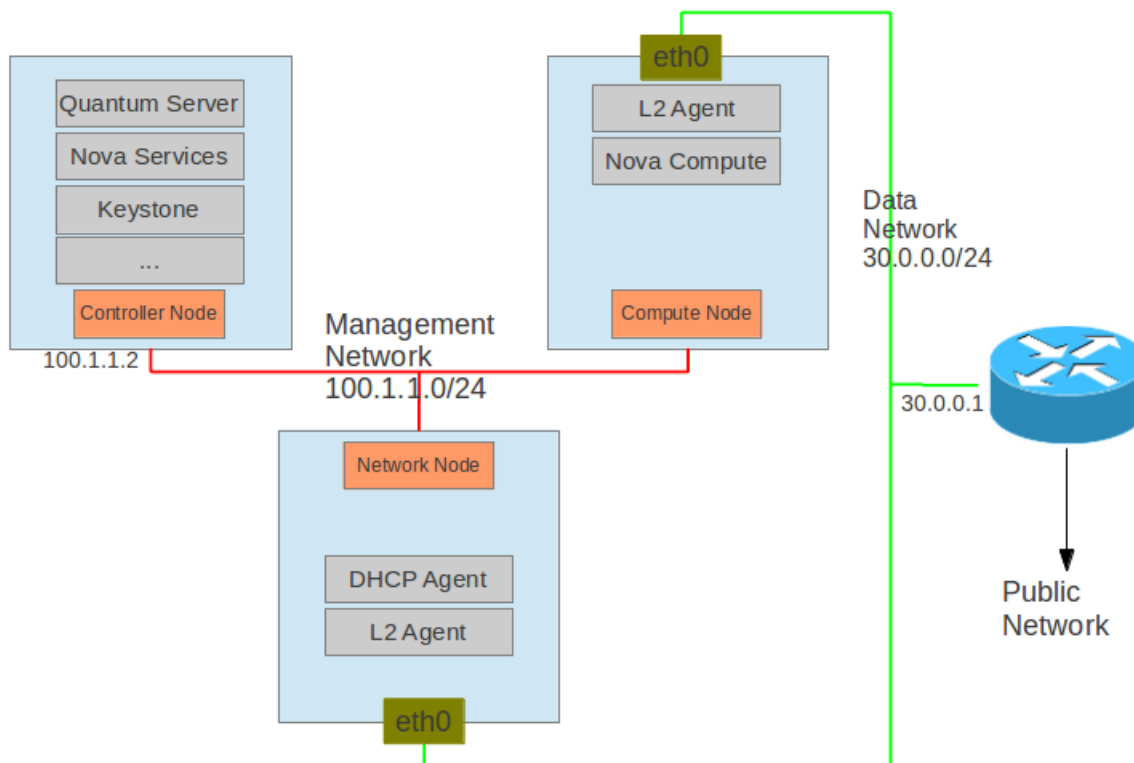
This section describes how to configure the OpenStack Networking service and its components for some typical use cases.

Single Flat Network

This section describes how to install the OpenStack Networking service and its components for the "Use Case: Single Flat Network".

The diagram below shows the setup. For simplicity all of the nodes should have one interface for management traffic and one or more interfaces for traffic to and from VMs. The management network is 100.1.1.0/24 with controller node at 100.1.1.2. The example uses the Open vSwitch plugin and agent.

Note the setup can be tweaked to make use of another supported plugin and its agent.



Here are some nodes in the setup.

Node	Description
Controller Node	Runs the OpenStack Networking service, OpenStack Identity and all of the OpenStack Compute services that are required to deploy VMs (nova-api, nova-scheduler, for example). The node must have at least one network interface, which is connected to the "Management Network". The hostname is 'controlnode', which every other node resolve to the controller node's IP. Note The nova-network service should not be running. This is replaced by OpenStack Networking.
Compute Node	Runs the OpenStack Networking L2 agent and the OpenStack Compute services that run VMs (nova-compute specifically, and optionally other nova-* services depending on configuration). The node must have at least two network interfaces. The first is used to communicate with the controller node via the management network. The second interface is used for the VM traffic on the Data network. The VM will be able to receive its IP address from the DHCP agent on this network.
Network Node	Runs OpenStack Networking L2 agent and the DHCP agent. The DHCP agent will allocate IP addresses to the VMs on the network. The node must have at least two network interfaces. The first is used to communicate with the controller node via the management network. The second interface will be used for the VM traffic on the data network.
Router	Router has IP 30.0.0.1, which is the default gateway for all VMs. The router should have ability to access public networks.

The demo assumes the following:

Controller Node

1. Relevant OpenStack Compute services are installed, configured and running.
2. Glance is installed, configured and running. In addition to this there should be an image.
3. OpenStack Identity is installed, configured and running. An OpenStack Networking user **quantum** should be created on tenant **servicetenant** with password **servicepassword**.
4. Additional services
 - RabbitMQ is running with default guest and its password
 - MySQL server (user is **root** and password is **root**)

Compute Node

1. OpenStack Compute compute is installed and configured

Installation

• Controller Node - OpenStack Networking Server

1. Install the OpenStack Networking server.
2. Create database **ovs_quantum**. See the section on the [Core Plugins](#) for the exact details.
3. Update the OpenStack Networking configuration file, **/etc/quantum/quantum.conf** setting plugin choice and Identity Service user as necessary:

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
```

```
notification_driver = quantum.openstack.common.notifier.rabbit_notifier

[keystone_authtoken]
admin_tenant_name=servicetenant
admin_user=quantum
admin_password=servicepassword
```

4. Update the plugin configuration file, `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini`:

```
[DATABASE]
sql_connection = mysql://root:root@controlnode:3306/ovs_quantum?charset=utf8

[OVS]
network_vlan_ranges = physnet1
bridge_mappings = physnet1:br-eth0
```

5. Start the OpenStack Networking service

- **Compute Node - OpenStack Compute**

1. Install the nova-compute service.
2. Update the OpenStack Compute configuration file, `/etc/nova/nova.conf`. **Make sure the following is at the end of this file:**

```
network_api_class=nova.network.quantumv2.api.API

quantum_admin_username=quantum
quantum_admin_password=servicepassword
quantum_admin_auth_url=http://controlnode:35357/v2.0/
quantum_auth_strategy=keystone
quantum_admin_tenant_name=servicetenant
quantum_url=http://controlnode:9696/

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

3. Restart the OpenStack Compute service

- **Compute and Network Node - L2 Agent**

1. Install the L2 agent.
2. Add the integration bridge to the Open vSwitch:

```
sudo ovs-vsctl add-br br-int
```

3. Update the OpenStack Networking configuration file, `/etc/quantum/quantum.conf`:

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
notification_driver = quantum.openstack.common.notifier.rabbit_notifier
```

4. Update the plugin configuration file, `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini`:

```
[DATABASE]
sql_connection = mysql://root:root@controlnode:3306/ovs_quantum?charset=utf8
[OVS]
network_vlan_ranges = physnet1
bridge_mappings = physnet1:br-eth0
```

5. Create the network bridge **br-eth0** (All VM communication between the nodes will be done via eth0):

```
sudo ovs-vsctl add-br br-eth0
sudo ovs-vsctl add-port br-eth0 eth0
```

6. Start the OpenStack Networking L2 agent

- **Network Node - DHCP Agent**

1. Install the DHCP agent.
2. Update the OpenStack Networking configuration file, **/etc/quantum/quantum.conf**:

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
notification_driver = quantum.openstack.common.notifier.rabbit_notifier
```

3. Update the DHCP configuration file **/etc/quantum/dhcp_agent.ini**:

```
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver
```

4. Start the DHCP agent

Logical Network Configuration

All of the commands below can be executed on the network node.

Note please ensure that the following environment variables are set. These are used by the various clients to access OpenStack Identity.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://127.0.0.1:5000/v2.0/
```

1. Get the tenant ID (Used as \$TENANT_ID later):

```
keystone tenant-list
```

id	name	enabled
247e478c599f45b5bd297e8ddb9b6a	TenantA	True
2b4fec24e62e4ff28a8445ad83150f9d	TenantC	True
3719a4940bf24b5a8124b58c9b0a6ee6	TenantB	True
5fcfb3283a142a5bb6978b549a511ac	demo	True

```
| b7445f221cda4f4a8ac7db6b218b1339 | admin | True |
+-----+-----+-----+
```

2. Get the User information:

```
keystone user-list
```

```
+-----+-----+-----+
| id | name | enabled | email |
+-----+-----+-----+
| 5a9149ed991744fa85f71e4aa92eb7ec | demo | True | |
| 5b419c74980d46a1ab184e7571a8154e | admin | True | admin@example.com |
| 8e37cb8193cb4873a35802d257348431 | UserC | True | |
| c11f6b09ed3c45c09c21cbbc23e93066 | UserB | True | |
| ca567c4f6c0942bdac0e011e97bddbe3 | UserA | True | |
+-----+-----+-----+
```

3. Create a internal shared network on the demo tenant (\$TENANT_ID will be b7445f221cda4f4a8ac7db6b218b1339):

```
quantum net-create --tenant-id $TENANT_ID sharednet1 --shared --
provider:network_type flat --provider:physical_network physnet1
```

Created a new network:

```
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| id | 04457b44-e22a-4a5c-be54-a53a9b2818e7 |
| name | sharednet1 |
| provider:network_type | flat |
| provider:physical_network | physnet1 |
| provider:segmentation_id | |
| router:external | False |
| shared | True |
| status | ACTIVE |
| subnets | |
| tenant_id | b7445f221cda4f4a8ac7db6b218b1339 |
+-----+-----+
```

4. Create a subnet on the network:

```
quantum subnet-create --tenant-id $TENANT_ID sharednet1 30.0.0.0/24
```

Created a new subnet:

```
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | {"start": "30.0.0.2", "end": "30.0.0.254"} |
| cidr | 30.0.0.0/24 |
| dns_nameservers | |
| enable_dhcp | True |
| gateway_ip | 30.0.0.1 |
| host_routes | |
| id | b8e9a88e-ded0-4e57-9474-e25fa87c5937 |
| ip_version | 4 |
| name | |
| network_id | 04457b44-e22a-4a5c-be54-a53a9b2818e7 |
+-----+-----+
```

```
| tenant_id | 5fcfbc3283a142a5bb6978b549a511ac |
+-----+
```

5. Create a server for tenant A:

```
nova --os-tenant-name TenantA --os-username UserA --os-password password
--os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1 --nic
net-id=04457b44-e22a-4a5c-be54-a53a9b2818e7 TenantA_VM1
```

```
nova --os-tenant-name TenantA --os-username UserA --os-password password --
os-auth-url=http://localhost:5000/v2.0 list
```

ID	Name	Status	Networks
09923b39-050d-4400-99c7-e4b021cdc7c4	TenantA_VM1	ACTIVE	sharednet1=30.0.0.3

6. Ping the server of tenant A:

```
sudo ip addr flush eth0
sudo ip addr add 30.0.0.201/24 dev br-eth0
ping 30.0.0.3
```

7. Ping the public network within the server of tenant A:

```
ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data:
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=1.74 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=1.50 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.23 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.234/1.495/1.745/0.211 ms
```

Note: The 192.168.1.1 is an IP on public network that the router is connecting.

8. Create servers for other tenants

We can create servers for other tenants with similar commands. Since all these VMs share the same subnet, they will be able to access each other.

Provider Router with Private Networks

This section describes how to install the OpenStack Networking service and its components for the "Use Case: Provider Router with Private Networks".

We will follow the [Basic Install](#) document except for the Quantum, Open-vSwitch and Virtual Networking sections on each of the nodes.

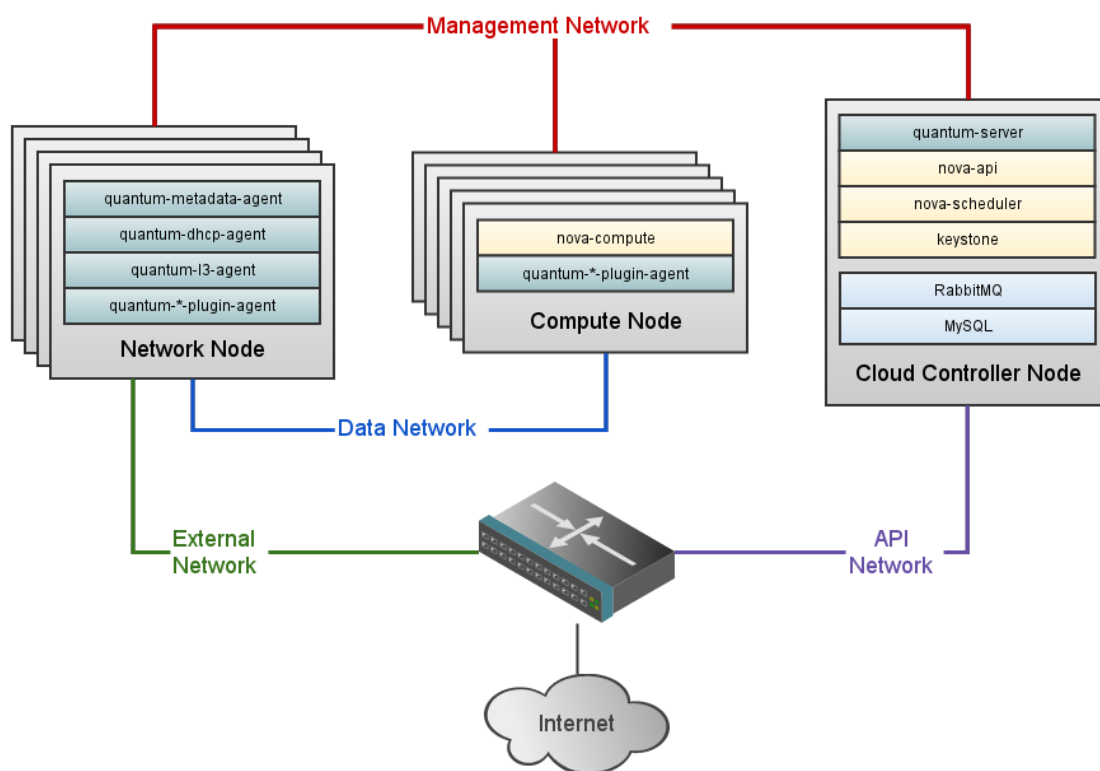
The **Basic Install** document uses gre tunnels. This document will describe how to use vlans for separation instead.

The diagram below shows the setup:



Note

Since we are running DHCP agent and L3 agent on one node, we should set `use_namespaces` to `True` (which is True by default) in both agents' configuration file. Please refer to [Limitations](#).



There will be three nodes in the setup.

Table A.1. Nodes for Demo

Node	Description
Controller Node	Runs the OpenStack Networking service, OpenStack Identity and all of the OpenStack Compute services that

Node	Description
	are required to deploy a VM. The service must have at least two network interfaces The first should be connected to the "Management Network". This will be used to communicate with the compute and network nodes. The second interface will be connected to the API/public network
Compute Node	Runs OpenStack Compute and the OpenStack Networking L2 agent. This node will not have access the public network. The node must have at least two network interfaces. The first is used to communicate with the controller node, via the management network. The VM will be able to receive its IP address from the DHCP agent on this network.
Network Node	Runs OpenStack Networking L2 agent, the DHCP agent and the L3 agent. This node will have access to the public network. The DHCP agent will allocate IP addresses to the VMs on the network. The L3 agent will perform NAT and enable the VMs to access the public network. The node must have at least three network interfaces. The first is used to communicate with the controller node, this is via the management network. The second interface will be used for the VM traffic, this is on the data network. The third interface will be used to connect to the external gateway on the network.

Installations

Controller

1. Install the packages :

```
apt-get install -y quantum-server
```

2. Configure Quantum services :

- Edit `/etc/quantum/quantum.conf` file and modify :

```
core_plugin = \
    quantum.plugins.openvswitch.ovs_quantum_plugin.OVSQuantumPluginV2
auth_strategy = keystone
fake_rabbit = False
rabbit_password = password
```

- Edit `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini` file and modify :

```
[DATABASE]
sql_connection = mysql://quantum:password@localhost:3306/quantum
[OVS]
tenant_network_type = vlan
network_vlan_ranges = physnet1:100:2999
```

- Edit `/etc/quantum/api-paste.ini` file and modify :

```
admin_tenant_name = service
admin_user = quantum
admin_password = password
```

3. Start the services :


```
service quantum-server restart
```

Network Node

1. Install the packages:

```
apt-get install -y quantum-plugin-openvswitch-agent \
quantum-dhcp-agent quantum-l3-agent
```

2. Start Open vSwitch:

```
service openvswitch-switch start
```

3. Add the integration bridge to the Open vSwitch:

```
sudo ovs-vsctl add-br br-int
```

4. Update the OpenStack Networking configuration file, `/etc/quantum/quantum.conf`:

```
rabbit_password = password
rabbit_host = 192.168.0.1
```

5. Update the plugin configuration file, `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini`:

```
[DATABASE]
sql_connection = mysql://quantum:password@192.168.0.1:3306/quantum
[OVS]
tenant_network_type=vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-eth1
```

6. Create the network bridge **br-eth1** (All VM communication between the nodes will be done via eth1):

```
sudo ovs-vsctl add-br br-eth1
sudo ovs-vsctl add-port br-eth1 eth1
```

7. Create the external network bridge to the Open vSwitch:

```
sudo ovs-vsctl add-br br-ex
sudo ovs-vsctl add-port br-ex eth2
```

8. Edit `/etc/quantum/l3_agent.ini` file and modify:

```
[DEFAULT]
auth_url = http://192.168.0.1:35357/v2.0
admin_tenant_name = service
admin_user = quantum
admin_password = password
metadata_ip = 192.168.0.1
use_namespaces = True
```

9. Edit `/etc/quantum/api-paste.ini` file and modify:

```
[DEFAULT]
auth_host = 192.168.0.1
admin_tenant_name = service
admin_user = quantum
```

```
admin_password = password
```

10. Edit `/etc/quantum/dhcp_agent.ini` file and modify:

```
use_namespaces = True
```

11. Restart networking services

```
service quantum-plugin-openvswitch-agent start
service quantum-dhcp-agent restart
service quantum-l3-agent restart
```

Compute Node

1. Install the packages.

```
apt-get install -y openvswitch-switch quantum-plugin-openvswitch-agent
```

2. Start open vSwitch Service

```
service openvswitch-switch start
```

3. Configure Virtual Bridging

```
sudo ovs-vsctl add-br br-int
```

4. Update the OpenStack Networking configuration file, `/etc/quantum/quantum.conf`:

```
rabbit_password = password
rabbit_host = 192.168.0.1
```

5. Update `/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini`:

```
[DATABASE]
sql_connection = mysql://quantum:password@192.168.0.1:3306/quantum
[OVS]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-eth1
```

6. Start the DHCP agent

```
service quantum-plugin-openvswitch-agent restart
```

Logical Network Configuration

All of the commands below can be done on the Network node.

Note please ensure that the following environment variables are set. These are used by the various clients to access OpenStack Identity.

- Create `novarc` file :

```
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=password
export OS_AUTH_URL="http://192.168.0.1:5000/v2.0/"
export SERVICE_ENDPOINT="http://192.168.0.1:35357/v2.0"
```

```
export SERVICE_TOKEN=password
```

- Export the variables :

```
source novarc
echo "source novarc">>.bashrc
```

• Internal Networking Configuration

1. Get the tenant ID (Used as \$TENANT_ID later).

```
keystone tenant-list
```

id	name	enabled
48fb81ab2f6b409bafac8961a594980f	admin	True
cbb574ac1e654a0a992bfc0554237abf	service	True
e371436fe2854ed89cca6c33ae7a83cd	invisible_to_admin	True
e40fa60181524f9f9ee7aa1038748f08	demo	True

2. Create a internal network named **net1** on the demo tenant (\$TENANT_ID will be e40fa60181524f9f9ee7aa1038748f08):

```
quantum net-create --tenant-id $TENANT_ID net1 --provider:network_type
vlan \
--provider:physical_network physnet1 --provider:segmentation_id 1024
```

Field	Value
admin_state_up	True
id	e99a361c-0af8-4163-9feb-8554d4c37e4f
name	net1
provider:network_type	vlan
provider:physical_network	physnet1
provider:segmentation_id	1024
router:external	False
shared	False
status	ACTIVE
subnets	
tenant_id	e40fa60181524f9f9ee7aa1038748f08

3. Create a subnet on the network **net1** (ID field below is used as \$SUBNET_ID later):

```
quantum subnet-create --tenant-id $TENANT_ID net1 10.5.5.0/24
```

Field	Value
allocation_pools	{"start": "10.5.5.2", "end": "10.5.5.254"}
cidr	10.5.5.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	10.5.5.1
host_routes	
id	c395cb5d-ba03-41ee-8a12-7e792d51a167
ip_version	4
name	

network_id	e99a361c-0af8-4163-9feb-8554d4c37e4f
tenant_id	e40fa60181524f9f9ee7aa1038748f08

- **External Networking Configuration**

1. Create a router named **router1** (ID is used as \$ROUTER_ID later):

```
quantum router-create --tenant_id $TENANT_ID router1
```

Field	Value
admin_state_up	True
external_gateway_info	
id	685f64e7-a020-4fdf-a8ad-e41194ae124b
name	router1
status	ACTIVE
tenant_id	e40fa60181524f9f9ee7aa1038748f08

2. Add an interface to **router1** and attach the **net1** subnet:

```
quantum router-interface-add $ROUTER_ID $SUBNET_ID
```

```
Added interface to router 685f64e7-a020-4fdf-a8ad-e41194ae124b
```

3. Create the external network named **ext_net**. Environment variables will attach this to the **admin** tenant (Used as \$EXTERNAL_NETWORK_ID). **Note** this is a different \$TENANT_ID than used previously:

```
quantum net-create ext_net --router:external=True
```

Field	Value
admin_state_up	True
id	8858732b-0400-41f6-8e5c-25590e67ffeb
name	ext_net
provider:network_type	vlan
provider:physical_network	physnet1
provider:segmentation_id	1
router:external	True
shared	False
status	ACTIVE
subnets	
tenant_id	48fb81ab2f6b409bafac8961a594980f

4. Create the subnet for floating IPs. **Note** the DHCP service is disabled for this subnet:

```
quantum subnet-create ext_net --allocation-pool start=7.7.7.130,end=7.7.7.150 \
--gateway 7.7.7.1 7.7.7.0/24 -- --enable_dhcp=False
```

Field	Value
allocation_pools	{"start": "7.7.7.130", "end": "7.7.7.150"}
cidr	7.7.7.0/24

dns_nameservers	
enable_dhcp	False
gateway_ip	7.7.7.1
host_routes	
id	aef60b55-cbff-405d-a81d-406283ac6cff
ip_version	4
name	
network_id	8858732b-0400-41f6-8e5c-25590e67ffeb
tenant_id	48fb81ab2f6b409bafac8961a594980f

5. Set the router gateway towards the external network:

```
quantum router-gateway-set $ROUTER_ID $EXTERNAL_NETWORK_ID
```

```
Set gateway for router 685f64e7-a020-4fdf-a8ad-e41194ae124b
```

• Floating IP Allocation

1. After a VM is deployed a floating IP address can be associated to the VM. A VM that is created will be allocated an OpenStack Networking port (\$PORT_ID). The port ID for the VM can be retrieved as follows (VIA the Controller or Compute Node) :

```
nova list
```

ID	Name	Status	Networks
1cdc671d-a296-4476-9a75-f9ca1d92fd26	testvm	ACTIVE	net1=10.5.5.3

```
quantum port-list -- --device_id 1cdc671d-a296-4476-9a75-f9ca1d92fd26
```

id	name	mac_address
fixed_ips		
9aa47099-b87b-488c-8c1d-32f993626a30		fa:16:3e:b4:d6:6c
{ "subnet_id": "c395cb5d-ba03-41ee-8a12-7e792d51a167", "ip_address": "10.0.0.3" }		

2. Allocate a floating IP (Used as \$FLOATING_ID):

```
quantum floatingip-create ext_net
```

Field	Value
fixed_ip_address	
floating_ip_address	7.7.7.131
floating_network_id	8858732b-0400-41f6-8e5c-25590e67ffeb
id	40952c83-2541-4d0c-b58e-812c835079a5

port_id	
router_id	
tenant_id	e40fa60181524f9f9ee7aa1038748f08

3. Associate a floating IP to a VM (in the case of the example it is 9aa47099-b87b-488c-8c1d-32f993626a30):

```
quantum floatingip-associate $FLOATING_ID $PORT_ID
```

```
Associated floatingip 40952c83-2541-4d0c-b58e-812c835079a5
```

4. Show the floating IP:

```
quantum floatingip-show $FLOATING_ID
```

Field	Value
fixed_ip_address	10.5.5.3
floating_ip_address	7.7.7.131
floating_network_id	8858732b-0400-41f6-8e5c-25590e67ffeb
id	40952c83-2541-4d0c-b58e-812c835079a5
port_id	9aa47099-b87b-488c-8c1d-32f993626a30
router_id	685f64e7-a020-4fdf-a8ad-e41194ae124b
tenant_id	e40fa60181524f9f9ee7aa1038748f08

5. Test the floating IP:

```
ping 7.7.7.131
```

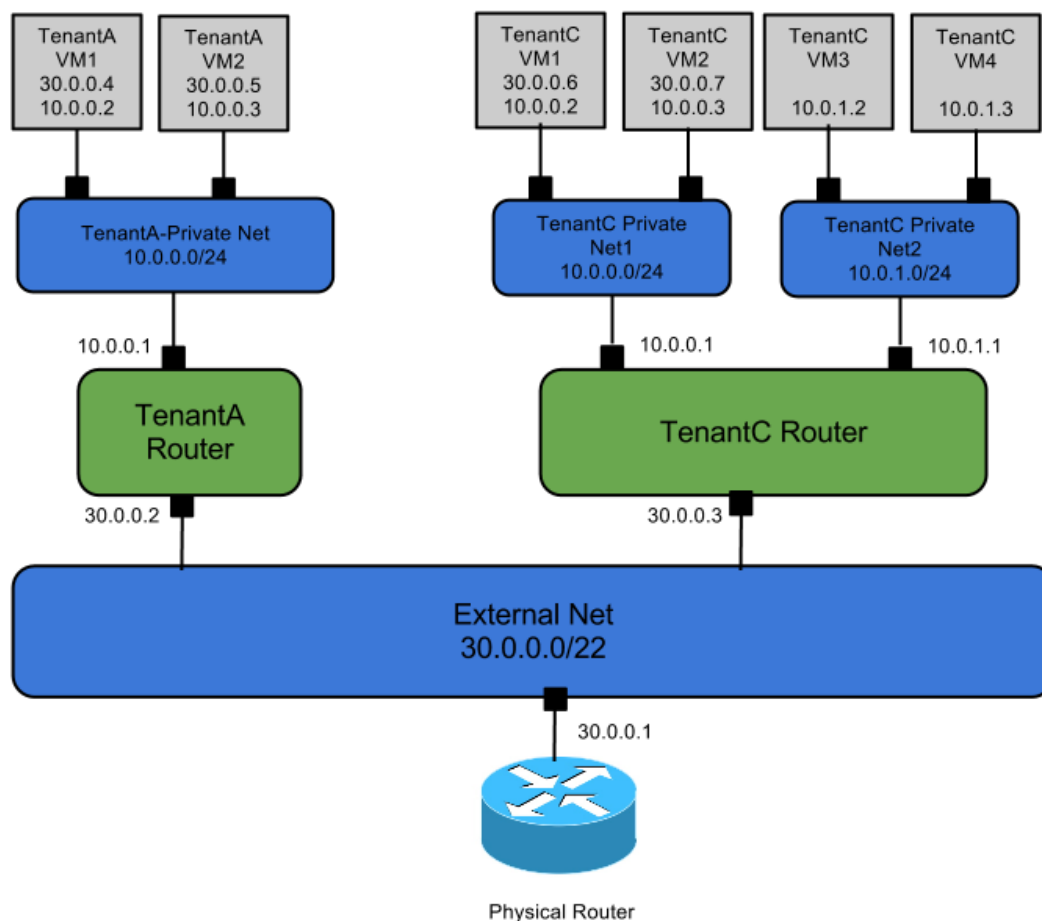
```
PING 7.7.7.131 (7.7.7.131) 56(84) bytes of data.
```

```
64 bytes from 7.7.7.131: icmp_req=2 ttl=64 time=0.152 ms
```

```
64 bytes from 7.7.7.131: icmp_req=3 ttl=64 time=0.049 ms
```

Per-tenant Routers with Private Networks

This section describes how to install the OpenStack Networking service and its components for the "Use Case: Per-tenant Routers with Private Networks".

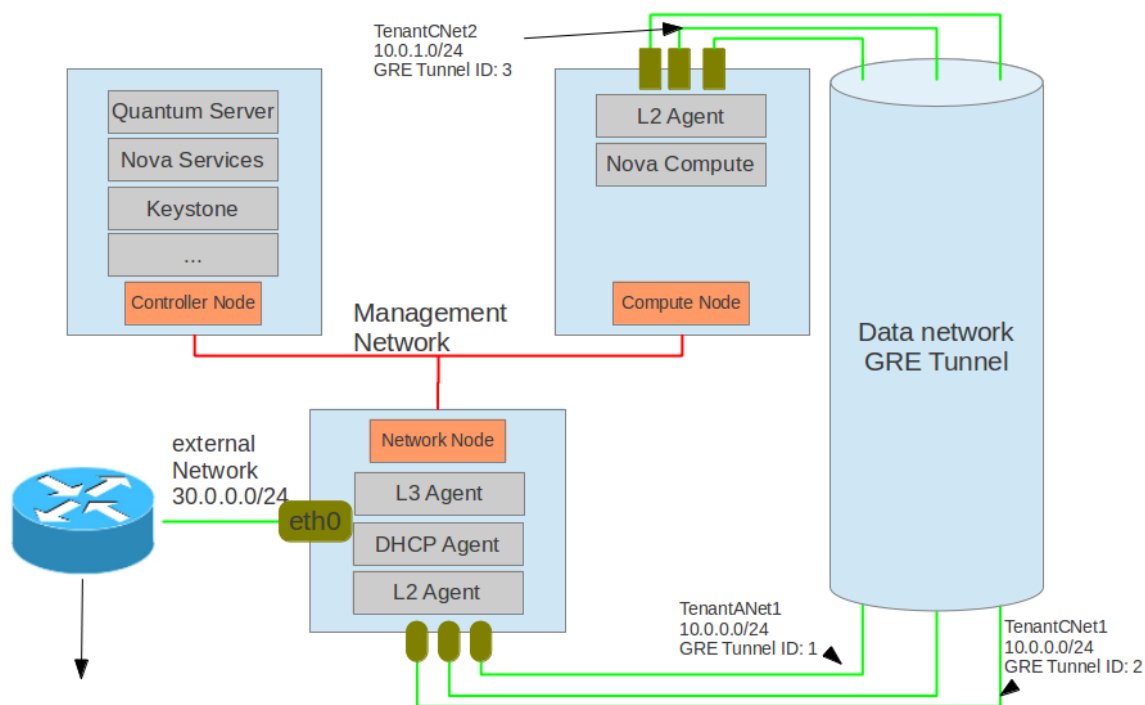


The diagram below shows the setup. All of the nodes should have one interface for management traffic. The example uses the Open vSwitch plugin. The agents are configured to use GRE for data transport. There is an external network where we will have floating IPs and router gateway ports. A physical router is connecting these floating IPs and router gateway ports to outside world.



Note

Since we are running DHCP agent and L3 agent on one node, we should set `use_namespaces` to `True` (which is `True` by default) in both agents' configuration file. Please refer to [Limitations](#).



Below are nodes in the setup:

Node	Description
Controller Node	Runs the OpenStack Networking service, OpenStack Identity and all of the OpenStack Compute services that are required to deploy VMs (nova-api, nova-scheduler, for example). The node must have at least one network interface, which is connected to the "Management Network". The hostname is 'controlnode', which every other node resolve to the controller node's IP. Note The nova-network service should not be running. This is replaced by OpenStack Networking.
Compute Node	Runs the OpenStack Networking L2 agent and the OpenStack Compute services that run VMs (nova-compute specifically, and optionally other nova-* services depending on configuration). The node must have at least two network interfaces. The first is used to communicate with the controller node via the management network. The second interface is used for the VM traffic on the Data network. The VM will be able to receive its IP address from the DHCP agent on this network.
Network Node	Runs OpenStack Networking L2 agent, DHCP agent and L3 agent. This node will have access to the external network. The DHCP agent will allocate IP addresses to the VMs on data network (Technically, the addresses are allocated by the OpenStack Networking server, and distributed by the dhcp agent). The node must have at least two network interfaces. The first is used to communicate with the controller node via the management network. The second interface will be used as external network. GRE tunnels will be set up as data network.
Router	Router has IP 30.0.0.1, which is the default gateway for all VMs. The router should have ability to access public networks.

The demo assumes the following:

Controller Node

1. Relevant OpenStack Compute services are installed, configured and running.
2. Glance is installed, configured and running. In addition to this there should be an image named tty.

3. OpenStack Identity is installed, configured and running. A OpenStack Networking user named **quantum** should be created on tenant **servicetenant** with password **servicepassword**.
4. Additional services
 - RabbitMQ is running with default guest and its password
 - MySQL server (user is **root** and password is **root**)

Compute Node

1. OpenStack Compute is installed and configured

Installation

• Controller Node - OpenStack Networking Server

1. Install the OpenStack Networking server.
2. Create database **ovs_quantum**. See the section on the [Core Plugins](#) for the exact details.
3. Update the OpenStack Networking configuration file, **/etc/quantum/quantum.conf**, with plugin choice and Identity Service user as necessary:

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
notification_driver = quantum.openstack.common.notifier.rabbit_notifier

[keystone_authtoken]
admin_tenant_name=servicetenant
admin_user=quantum
admin_password=servicepassword
```

4. Update the plugin configuration file, **/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini**:

```
[DATABASE]
sql_connection = mysql://root:root@controlnode:3306/ovs_quantum?charset=
utf8
[OVS]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
```

5. Start the OpenStack Networking server

The OpenStack Networking server can be a service of the operating system. The command may be different to start the service on different operating systems. One example of the command to run the OpenStack Networking server directly is:

```
sudo quantum-server --config-file /etc/quantum/plugins/openvswitch/
ovs_quantum_plugin.ini --config-file /etc/quantum/quantum.conf
```

- **Compute Node - OpenStack Compute**

1. Install OpenStack Compute services.
2. Update the OpenStack Compute configuration file, **/etc/nova/nova.conf**. **Make sure the following is at the end of this file:**

```
network_api_class=nova.network.quantumv2.api.API

quantum_admin_username=quantum
quantum_admin_password=servicepassword
quantum_admin_auth_url=http://controlnode:35357/v2.0/
quantum_auth_strategy=keystone
quantum_admin_tenant_name=servicetenant
quantum_url=http://controlnode:9696/

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

3. Restart relevant OpenStack Compute services

- **Compute and Network Node - L2 Agent**

1. Install the L2 agent.
2. Add the integration bridge to the Open vSwitch

```
sudo ovs-vsctl add-br br-int
```

3. Update the OpenStack Networking configuration file, **/etc/quantum/quantum.conf**

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
notification_driver = quantum.openstack.common.notifier.rabbit_notifier
```

4. Update the plugin configuration file, **/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini**.

Compute Node:

```
[DATABASE]
sql_connection = mysql://root:root@controlnode:3306/ovs_quantum?charset=
utf8
[OVS]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
local_ip = 9.181.89.202
```

Network Node:

```
[DATABASE]
```

```
sql_connection = mysql://root:root@controlnode:3306/ovs_quantum?charset=utf8
[OVS]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
local_ip = 9.181.89.203
```

5. Create the integration bridge **br-int**:

```
sudo ovs-vsctl --may-exist add-br br-int
```

6. Start the OpenStack Networking L2 agent

The OpenStack Networking Open vSwitch L2 agent can be a service of operating system. The command may be different to start the service on different operating systems. However the command to run it directly is kind of like:

```
sudo quantum-openvswitch-agent --config-file /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini --config-file /etc/quantum/quantum.conf
```

- **Network Node - DHCP Agent**

1. Install the DHCP agent.

2. Update the OpenStack Networking configuration file, **/etc/quantum/quantum.conf**

```
[DEFAULT]
core_plugin = quantum.plugins.openvswitch.ovs_quantum_plugin.
OVSQuantumPluginV2
control_exchange = quantum
rabbit_host = controlnode
notification_driver = quantum.openstack.common.notifier.rabbit_notifier
allow_overlapping_ips = True
```

We set `allow_overlapping_ips` because we have overlapping subnets for TenantA and TenantC.

3. Update the DHCP configuration file **/etc/quantum/dhcp_agent.ini**

```
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver
```

4. Start the DHCP agent

The OpenStack Networking DHCP agent can be a service of operating system. The command may be different to start the service on different operating systems. However the command to run it directly is kind of like:

```
sudo quantum-dhcp-agent --config-file /etc/quantum/quantum.conf --config-file /etc/quantum/dhcp_agent.ini
```

- **Network Node - L3 Agent**

1. Install the L3 agent.

2. Add the external network bridge

```
sudo ovs-vsctl add-br br-ex
```

3. Add the physical interface, for example eth0, that is connected to the outside network to this bridge

```
sudo ovs-vsctl add-port br-ex eth0
```

4. Update the L3 configuration file `/etc/quantum/l3_agent.ini`:

```
[DEFAULT]
interface_driver=quantum.agent.linux.interface.OVSInterfaceDriver
use_namespaces=True
```

We set `use_namespaces` (It is True by default.) because we have overlapping subnets for TenantA and TenantC and we are going to host the routers with one l3 agent network node.

5. Start the L3 agent

The OpenStack Networking L3 agent can be a service of operating system. The command may be different to start the service on different operating systems. However the command to run it directly is kind of like:

```
sudo quantum-l3-agent --config-file /etc/quantum/quantum.conf --config-file /etc/quantum/l3_agent.ini
```

Logical Network Configuration

All of the commands below can be executed on the network node.

Note please ensure that the following environment variables are set. These are used by the various clients to access OpenStack Identity.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://127.0.0.1:5000/v2.0/
```

1. Get the tenant ID (Used as \$TENANT_ID later)

```
keystone tenant-list
```

id	name	enabled
247e478c599f45b5bd297e8ddbbc9b6a	TenantA	True
2b4fec24e62e4ff28a8445ad83150f9d	TenantC	True
3719a4940bf24b5a8124b58c9b0a6ee6	TenantB	True
5fcfb3283a142a5bb6978b549a511ac	demo	True
b7445f221cda4f4a8ac7db6b218b1339	admin	True

2. Get the user information

```
keystone user-list
```

id	name	enabled	email
5a9149ed991744fa85f71e4aa92eb7ec	demo	True	
5b419c74980d46a1ab184e7571a8154e	admin	True	admin@example.com
8e37cb8193cb4873a35802d257348431	UserC	True	
c11f6b09ed3c45c09c21cbbc23e93066	UserB	True	
ca567c4f6c0942bdac0e011e97bddbe3	UserA	True	

3. Create the external network and its subnet by admin user:

```
quantum net-create Ext-Net --provider:network_type local --router:external true
```

Created a new network:

Field	Value
admin_state_up	True
id	2c757c9e-d3d6-4154-9a77-336eb99bd573
name	Ext-Net
provider:network_type	local
provider:physical_network	
provider:segmentation_id	
router:external	True
shared	False
status	ACTIVE
subnets	
tenant_id	b7445f221cda4f4a8ac7db6b218b1339

```
quantum subnet-create Ext-Net 30.0.0.0/24
```

Created a new subnet:

Field	Value
allocation_pools	{"start": "30.0.0.2", "end": "30.0.0.254"}
cidr	30.0.0.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	30.0.0.1
host_routes	
id	ba754a55-7ce8-46bb-8d97-aa83f4ffa5f9
ip_version	4
name	
network_id	2c757c9e-d3d6-4154-9a77-336eb99bd573
tenant_id	b7445f221cda4f4a8ac7db6b218b1339

provider:network_type local means we don't need OpenStack Networking to realize this network through provider network. **router:external true** means we are creating an external network, on which we can create floating ip and router gateway port.

4. Add an IP on external network to br-ex

Since we are using br-ex as our external network bridge, we will add an IP 30.0.0.100/24 to br-ex and then ping our VM's floating IP from our network node.

```
sudo ip addr add 30.0.0.100/24 dev br-ex
sudo ip link set br-ex up
```

5. Serve TenantA

For TenantA, we will create a private network, a subnet, a server, a router and a floating IP.

a. Create a network for TenantA

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 net-create TenantA-Net
Created a new network:
```

Field	Value
admin_state_up	True
id	7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68
name	TenantA-Net
router:external	False
shared	False
status	ACTIVE
subnets	
tenant_id	247e478c599f45b5bd297e8ddbbc9b6a

After that we can use admin user to query the network's provider network information:

```
quantum net-show TenantA-Net
```

Field	Value
admin_state_up	True
id	7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68
name	TenantA-Net
provider:network_type	gre
provider:physical_network	
provider:segmentation_id	1
router:external	False
shared	False
status	ACTIVE
subnets	
tenant_id	247e478c599f45b5bd297e8ddbbc9b6a

We can see that it has GRE tunnel ID (I.E. provider:segmentation_id) 1.

b. Create a subnet on the network TenantA-Net

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantA-
Net 10.0.0.0/24
```

Created a new subnet:

Field	Value
allocation_pools	{"start": "10.0.0.2", "end": "10.0.0.254"}
cidr	10.0.0.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	10.0.0.1
host_routes	
id	51e2c223-0492-4385-b6e9-83d4e6d10657
ip_version	4
name	
network_id	7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68
tenant_id	247e478c599f45b5bd297e8ddb9b6a

c. Create a server for TenantA

```
nova --os-tenant-name TenantA --os-username UserA --os-password password
--os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1 --
nic net-id=7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68 TenantA_VM1
```

```
nova --os-tenant-name TenantA --os-username UserA --os-password password
--os-auth-url=http://localhost:5000/v2.0 list
```

ID	Name	Status	Networks
7c5e6499-7ef7-4e36-8216-62c2941d21ff	TenantA_VM1	ACTIVE	TenantA-Net=10.0.0.3

d. Create and configure a router for TenantA:

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 router-create TenantA-
R1
```

Created a new router:

Field	Value
admin_state_up	True
external_gateway_info	
id	59cd02cb-6ee6-41e1-9165-d251214594fd
name	TenantA-R1
status	ACTIVE
tenant_id	247e478c599f45b5bd297e8ddb9b6a

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 router-interface-add
TenantA-R1 51e2c223-0492-4385-b6e9-83d4e6d10657
Added interface to router TenantA-R1
```

```
quantum router-gateway-set TenantA-R1 Ext-Net
```

We are using admin user to run last command since our external network is owned by admin tenant.

e. Associate a floating IP for TenantA_VM1

1. Create a floating IP

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 floatingip-create Ext-
Net
Created a new floatingip:
```

Field	Value
fixed_ip_address	
floating_ip_address	30.0.0.2
floating_network_id	2c757c9e-d3d6-4154-9a77-336eb99bd573
id	5a1f90ed-aa3c-4df3-82cb-116556e96bf1
port_id	
router_id	
tenant_id	247e478c599f45b5bd297e8ddbbc9b6a

2. Get the port ID of the VM with ID 7c5e6499-7ef7-4e36-8216-62c2941d21ff

```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 port-list -- --
device_id 7c5e6499-7ef7-4e36-8216-62c2941d21ff
```

id	name	mac_address
fixed_ips		
6071d430-c66e-4125-b972-9a937c427520		fa:16:3e:a0:73:0d
{"subnet_id": "51e2c223-0492-4385-b6e9-83d4e6d10657", "ip_address": "10.0.0.3"}		

3. Associate the floating IP with the VM port


```
quantum --os-tenant-name TenantA --os-username UserA --os-password
password --os-auth-url=http://localhost:5000/v2.0 floatingip-
associate 5alf90ed-aa3c-4df3-82cb-116556e96bf1 6071d430-c66e-4125-
b972-9a937c427520
Associated floatingip 5alf90ed-aa3c-4df3-82cb-116556e96bf1
quantum floatingip-list
```

id	floating_ip_address	port_id	fixed_ip_address
5alf90ed-aa3c-4df3-82cb-116556e96bf1	6071d430-c66e-4125-b972-9a937c427520		10.0.0.3
			30.0.0.2

f. Ping the public network from the server of TenantA

In my environment, 192.168.1.0/24 is my public network connected with my physical router, which also connects to the external network 30.0.0.0/24. With the floating IP and virtual router, we can ping the public network within the server of tenant A:

```
ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=1.74 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=1.50 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.23 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.234/1.495/1.745/0.211 ms
```

g. Ping floating IP of the TenantA's server

```
ping 30.0.0.2
PING 30.0.0.2 (30.0.0.2) 56(84) bytes of data.
64 bytes from 30.0.0.2: icmp_req=1 ttl=63 time=45.0 ms
64 bytes from 30.0.0.2: icmp_req=2 ttl=63 time=0.898 ms
64 bytes from 30.0.0.2: icmp_req=3 ttl=63 time=0.940 ms
^C
--- 30.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.898/15.621/45.027/20.793 ms
```

h. Create other servers for TenantA

We can create more servers for TenantA and add floating IPs for them.

6. Serve TenantC

For TenantC, we will create two private networks with subnet 10.0.0.0/24 and subnet 10.0.1.0/24, some servers, one router to connect to these two subnets and some floating IPs.

a. Create networks and subnets for TenantC

```

quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 net-create TenantC-
Net1
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantC-
Net1 10.0.0.0/24 --name TenantC-Subnet1
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 net-create TenantC-
Net2
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantC-
Net2 10.0.1.0/24 --name TenantC-Subnet2

```

After that we can use admin user to query the network's provider network information:

```

quantum net-show TenantC-Net1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| id | 91309738-c317-40a3-81bb-bed7a3917a85 |
| name | TenantC-Net1 |
| provider:network_type | gre |
| provider:physical_network | |
| provider:segmentation_id | 2 |
| router:external | False |
| shared | False |
| status | ACTIVE |
| subnets | cf03fd1e-164b-4527-bc87-2b2631634b83 |
| tenant_id | 2b4fec24e62e4ff28a8445ad83150f9d |
+-----+-----+
quantum net-show TenantC-Net2
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| id | 5b373ad2-7866-44f4-8087-f87148abd623 |
| name | TenantC-Net2 |
| provider:network_type | gre |
| provider:physical_network | |
| provider:segmentation_id | 3 |
| router:external | False |
| shared | False |
| status | ACTIVE |
| subnets | 38f0b2f0-9f98-4bf6-9520-f4abede03300 |
| tenant_id | 2b4fec24e62e4ff28a8445ad83150f9d |
+-----+-----+

```

We can see that we have GRE tunnel IDs (I.E. provider:segmentation_id) 2 and 3. And also note down the network IDs and subnet IDs because we will use them to create VMs and router.

- b. Create a server TenantC-VM1 for TenantC on TenantC-Net1

```
nova --os-tenant-name TenantC --os-username UserC --os-password password
--os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1 --
nic net-id=91309738-c317-40a3-81bb-bed7a3917a85 TenantC_VM1
```

c. Create a server TenantC-VM3 for TenantC on TenantC-Net2

```
nova --os-tenant-name TenantC --os-username UserC --os-password password
--os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1 --
nic net-id=5b373ad2-7866-44f4-8087-f87148abd623 TenantC_VM3
```

d. List servers of TenantC

```
nova --os-tenant-name TenantC --os-username UserC --os-password password
--os-auth-url=http://localhost:5000/v2.0 list
```

ID	Name	Status	Networks
b739fa09-902f-4b37-bcb4-06e8a2506823	TenantC_VM1	ACTIVE	TenantC-Net1=10.0.0.3
17e255b2-b14f-48b3-ab32-5df36566d2e8	TenantC_VM3	ACTIVE	TenantC-Net2=10.0.1.3

Note down the server IDs since we will use them later.

e. Make sure servers get their IPs

We can use VNC to log on the VMs to check if they get IPs. If not, we have to make sure the OpenStack Networking components are running right and the GRE tunnels work.

f. Create and configure a router for TenantC:

```
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 router-create TenantC-
R1
```

```
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 router-interface-add
TenantC-R1 cf03fd1e-164b-4527-bc87-2b2631634b83
quantum --os-tenant-name TenantC --os-username UserC --os-password
password --os-auth-url=http://localhost:5000/v2.0 router-interface-add
TenantC-R1 38f0b2f0-9f98-4bf6-9520-f4abede03300
```

```
quantum router-gateway-set TenantC-R1 Ext-Net
```

We are using admin user to run last command since our external network is owned by admin tenant.

g. Checkpoint: ping from within TenantC's servers

Since we have a router connecting to two subnets, the VMs on these subnets are able to ping each other. And since we have set the router's gateway interface, TenantC's servers are able to ping external network IPs, such as 192.168.1.1, 30.0.0.1 etc.

h. Associate floating IPs for TenantC's servers

We can use the similar commands as we used in TenantA's section to finish this task.

Scalable and Highly Available DHCP Agents

This section describes how to use the agent management (alias agent) and scheduler (alias agent_scheduler) extensions for DHCP agents scalability and HA

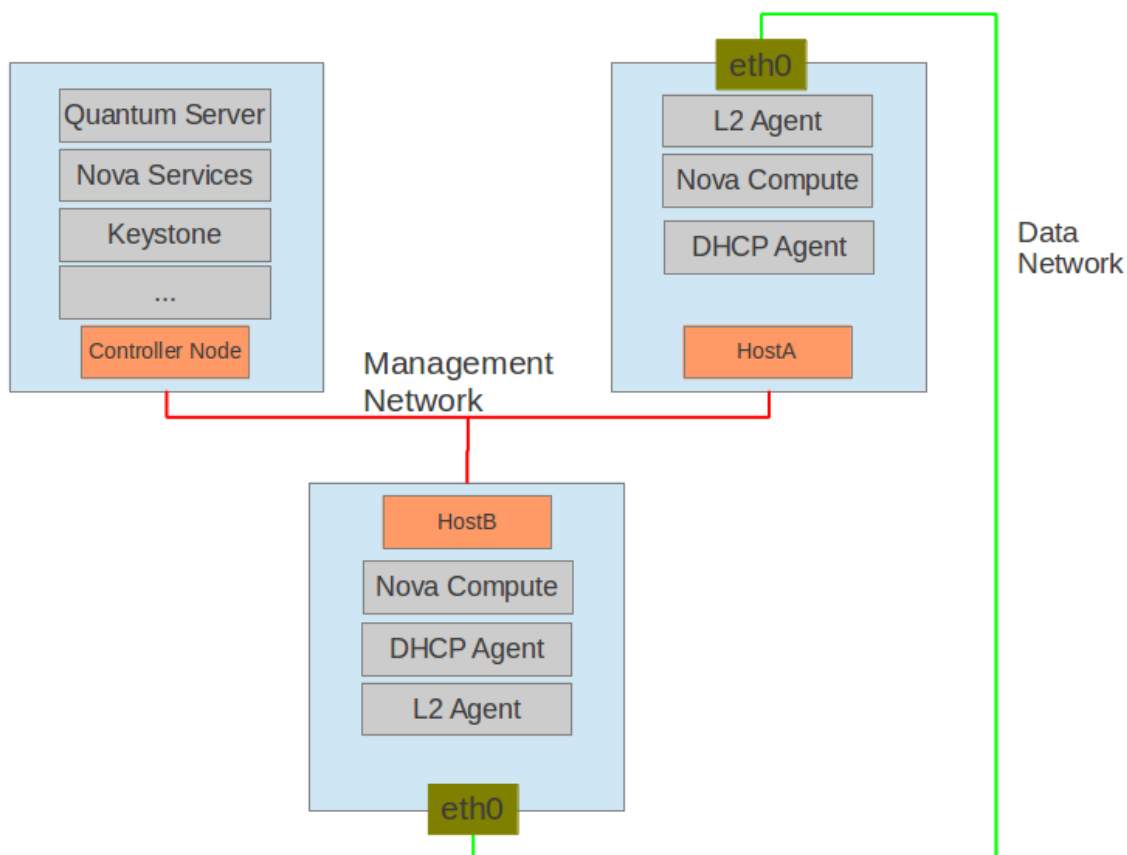


Note

Use the ext-list quantum client command to check if these extensions are enabled:

```
$ quantum ext-list -c name -c alias
```

alias	name
agent_scheduler	Agent Schedulers
binding	Port Binding
quotas	Quota management support
agent	agent
provider	Provider Network
router	Quantum L3 Router
lbaas	LoadBalancing service
extraroute	Quantum Extra Route



There will be three hosts in the setup.

Table A.2. Hosts for Demo

Host	Description
OpenStack Controller host - controlnode	Runs the Quantum service, Keystone and all of the Nova services that are required to deploy VMs. The node must have at least one network interface, this should be connected to the "Management Network". Note nova-network should not be running since it is replaced by Quantum.
HostA	Runs Nova compute, the Quantum L2 agent and DCHP agent
HostB	Same as HostA

Configuration

- **controlnode - Quantum Server**

1. Quantum configuration file `/etc/quantum/quantum.conf`:

```
[DEFAULT]
core_plugin = quantum.plugins.linuxbridge.lb_quantum_plugin.
LinuxBridgePluginV2
rabbit_host = controlnode
allow_overlapping_ips = True
host = controlnode
agent_down_time = 5
```

2. Update the plugin configuration file `/etc/quantum/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[VLANS]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[DATABASE]
sql_connection = mysql://root:root@127.0.0.1:3306/quantum_linux_bridge
reconnect_interval = 2
[LINUX_BRIDGE]
physical_interface_mappings = physnet1:eth0
```

- **HostA and HostB - L2 Agent**

1. Quantum configuration file `/etc/quantum/quantum.conf`:

```
[DEFAULT]
rabbit_host = controlnode
rabbit_password = openstack
# host = HostB on hostb
host = HostA
```

2. Update the plugin configuration file `/etc/quantum/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[VLANS]
tenant_network_type = vlan
```

```
network_vlan_ranges = physnet1:1000:2999
[DATABASE]
sql_connection = mysql://root:root@127.0.0.1:3306/quantum_linux_bridge
reconnect_interval = 2
[LINUX_BRIDGE]
physical_interface_mappings = physnet1:eth0
```

3. Update the nova configuration file `/etc/nova/nova.conf`:

```
[DEFAULT]
network_api_class=nova.network.quantumv2.api.API

quantum_admin_username=quantum
quantum_admin_password=servicepassword
quantum_admin_auth_url=http://controlnode:35357/v2.0/
quantum_auth_strategy=keystone
quantum_admin_tenant_name=servicetenant
quantum_url=http://100.1.1.10:9696/
firewall_driver=nova.virt.firewall.NoopFirewallDriver
```

- **HostA and HostB - DHCP Agent**

1. Update the DHCP configuration file `/etc/quantum/dhcp_agent.ini`:

```
[DEFAULT]
interface_driver = quantum.agent.linux.interface.BridgeInterfaceDriver
```

Commands in agent management and scheduler extensions

The following commands require the tenant running the command to have an admin role.



Note

Please ensure that the following environment variables are set. These are used by the various clients to access Keystone.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controlnode:5000/v2.0/
```

- **Settings**

We need some VMs and a quantum network to experiment. Here they are:

```
$ nova list
+-----+-----+-----+-----+
+ ID | Name | Status | Networks |
+-----+-----+-----+-----+
+ c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=10.0.1.3 |
+ 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=10.0.1.4 |
+ c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=10.0.1.5 |
```

```

+-----+-----+-----+-----+
+
$ quantum net-list
+-----+-----+-----+
| id | name | subnets |
+-----+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45- |
d5cf646db9d1 |
+-----+-----+-----+

```

- **Manage agents in quantum deployment**

Every agent which supports these extensions will register itself with the quantum server when it starts up.

1. List all agents:

```

$ quantum agent-list
+-----+-----+-----+-----+
| id | agent_type | host |
+-----+-----+-----+-----+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :- ) |
| True |
+-----+-----+-----+-----+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | DHCP agent | HostA | :- ) |
| True |
+-----+-----+-----+-----+
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :- ) |
| True |
+-----+-----+-----+-----+
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent | HostB | :- ) |
| True |
+-----+-----+-----+-----+

```

Just as shown, we have four agents now, and they have reported their state. The 'alive' will be ':-)' if the agent reported its state within the period defined by the option 'agent_down_time' in quantum server's quantum.conf. Otherwise the 'alive' is 'xxx'.

2. List the DHCP agents hosting a given network

In some deployments, one DHCP agent is not enough to hold all the network data. In addition, we should have backup for it even when the deployment is small one. The same network can be assigned to more than one DHCP agent and one DHCP agent can host more than one network. Let's first go with command that lists DHCP agents hosting a given network.

```

$ quantum dhcp-agent-list-hosting-net net1
+-----+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+-----+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True | :- ) |
+-----+-----+-----+-----+

```


3. List the networks hosted by a given DHCP agent.

This command is to show which networks a given dhcp agent is managing.

```
$ quantum net-list-on-dhcp-agent a0c1c21c-d4f4-4577-9ec7-908f2d48622d
+-----+
+-----+
| id | name | subnets |
+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-d5cf646db9d1 10.0.1.0/24 |
+-----+
+-----+
```

4. Show the agent detail information.

The 'agent-list' command gives very general information about agents. To obtain the detail information of an agent, we can use 'agent-show'.

```
$ quantum agent-show a0c1c21c-d4f4-4577-9ec7-908f2d48622d
+-----+
+-----+
| Field | Value |
+-----+
| admin_state_up | True |
| agent_type | DHCP agent |
| alive | False |
| binary | quantum-dhcp-agent |
| configurations | {
| | "subnets": 1,
| | "use_namespaces": true,
| | "dhcp_driver": "quantum.agent.linux.dhcp.
Dnsmasq",
| | "networks": 1,
| | "dhcp_lease_time": 120,
| | "ports": 3
| | }
| created_at | 2013-03-16T01:16:18.000000 |
| description |
```

```

| heartbeat_timestamp | 2013-03-17T01:37:22.000000
|
| host                | HostA
|
| id                  | 58f4ce07-6789-4bb3-aa42-ed3779db2b03
|
| started_at         | 2013-03-16T06:48:39.000000
|
| topic              | dhcp_agent
|
+-----+
+-----+

```

In the above output, 'heartbeat_timestamp' is the time on quantum server. So we don't need all agents synced to quantum server's time for this extension to run well. 'configurations' is about the agent's static configuration or run time data. We can see that this agent is a DHCP agent, and it is hosting one network, one subnet and 3 ports.

Different type of agents has different detail. Below is information for a 'Linux bridge agent'

```

$ quantum agent-show ed96b856-ae0f-4d75-bb28-40a47ffd7695
+-----+
| Field                | Value
+-----+
| admin_state_up       | True
| binary               | quantum-linuxbridge-agent
| configurations        | {
|                       |     "physnet1": "eth0",
|                       |     "devices": "4"
|                       | }
| created_at           | 2013-03-16T01:49:52.000000
| description          |
| disabled             | False
| group                | agent
| heartbeat_timestamp   | 2013-03-16T01:59:45.000000
| host                 | HostB
| id                   | ed96b856-ae0f-4d75-bb28-40a47ffd7695
| topic                | N/A
| started_at           | 2013-03-16T06:48:39.000000
| type                 | Linux bridge agent
+-----+

```

Just as shown, we can see bridge-mapping and the number of VM's virtual network devices on this L2 agent.

- **Manage assignment of networks to DHCP agent**

We have shown 'net-list-on-dhcp-agent' and 'dhcp-agent-list-hosting-net' commands. Now let's look at how to add a network to a DHCP agent and remove one from it.

1. Default scheduling.

When a network is created and one port is created on it, we will try to schedule it to an active DHCP agent. If there are many active DHCP agents, we select one randomly.

(We can design more sophisticated scheduling algorithm just like we do in nova-schedule later.)

```
$ quantum net-create net2
$ quantum subnet-create net2 9.0.1.0/24 --name subnet2
$ quantum port-create net2
$ quantum dhcp-agent-list-hosting-net net2
```

id	host	admin_state_up	alive
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	HostA	True	: -)

We can see it is allocated to DHCP agent on HostA. If we want to validate the behavior via dnsmasq, don't forget to create a subnet for the network since DHCP agent starts the dnsmasq only if there is a DHCP enabled subnet on it.

2. Assign a network to a given DHCP agent.

We have two DHCP agents, and we want another DHCP agent to host the network too.

```
$ quantum dhcp-agent-network-add f28aa126-6edb-4ea5-a81e-8850876bc0a8 net2
Added network net2 to dhcp agent
$ quantum dhcp-agent-list-hosting-net net2
```

id	host	admin_state_up	alive
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	HostA	True	: -)
f28aa126-6edb-4ea5-a81e-8850876bc0a8	HostB	True	: -)

We can see both DHCP agents are hosting 'net2' network.

3. Remove a network from a given DHCP agent.

This command is the sibling command for the previous one. Let's remove 'net2' from HostA's DHCP agent.

```
$ quantum dhcp-agent-network-remove a0c1c21c-d4f4-4577-9ec7-908f2d48622d net2
Removed network net2 to dhcp agent
$ quantum dhcp-agent-list-hosting-net net2
```

id	host	admin_state_up	alive
f28aa126-6edb-4ea5-a81e-8850876bc0a8	HostB	True	: -)

We can see now only HostB's DHCP agent is hosting 'net2' network.

- HA of DHCP agents

First we will boot a VM on net2, then we let both DHCP agents host 'net2'. After that, we fail the agents in turn and to see if the VM can still get the wanted IP during that time.

1. Boot a VM on net2.

```
$ quantum net-list
+-----+-----+
+-----+-----+
| id | name | subnets |
+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-d5cf646db9d1 10.0.1.0/24 |
| 9b96b14f-71b8-4918-90aa-c5d705606b1a | net2 | 6979b71a-0ae8-448c-aa87-65f68eedcaaa 9.0.1.0/24 |
+-----+-----+

$ nova boot --image tty --flavor 1 myserver4 --nic net-id=9b96b14f-71b8-4918-90aa-c5d705606b1a
$ nova list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ID | Name | Status | Networks |
+-----+-----+-----+-----+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=10.0.1.3 |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=10.0.1.4 |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=10.0.1.5 |
| f62f4731-5591-46b1-9d74-f0c901de567f | myserver4 | ACTIVE | net2=9.0.1.2 |
+-----+-----+-----+-----+
```

2. Make sure both DHCP agents hosting 'net2'.

We can use commands shown before to assign the network to agents.

```
$ quantum dhcp-agent-list-hosting-net net2
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+-----+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True | :- ) |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True | :- ) |
+-----+-----+-----+-----+
```

3. Testing the HA.

Step 1. Login to the 'myserver4' VM, and run 'udhcpc', 'dhclient' or other DHCP client.

Step 2. Stop the DHCP agent on HostA (Beside stopping the quantum-dhcp-agent binary, we must make sure dnsmasq processes are gone too.)

Step 3. Run a DHCP client in VM. We can see it can get the wanted IP.

Step 4. Stop the DHCP agent on HostB too.

Step 5. Run 'udhcpd' in VM. We can see it cannot get the wanted IP.

Step 6. Start DHCP agent on HostB. We can see VM can get the wanted IP again.

- Disable and remove an agent

An admin user wants to disable an agent if there is a system upgrade planned, whatever hardware or software. Some agents which support scheduling support disable or enable too, such as L3 agent and DHCP agent. Once the agent is disabled, the scheduler will not schedule new resources to the agent. After the agent is disabled, we can remove the agent safely. We should remove the resources on the agent before we delete the agent itself.

To run the commands below, we need first stop the DHCP agent on HostA.

```
$ quantum agent-update --admin-state-up False a0c1c21c-d4f4-4577-9ec7-908f2d48622d
$ quantum agent-list
```

id	admin_state_up	agent_type	host	alive
1b69828d-6a9b-4826-87cd-1757f0e27f31	True	Linux bridge agent	HostA	:-)
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	False	DHCP agent	HostA	:-)
ed96b856-ae0f-4d75-bb28-40a47ffd7695	True	Linux bridge agent	HostB	:-)
f28aa126-6edb-4ea5-a81e-8850876bc0a8	True	DHCP agent	HostB	:-)

```
$ quantum agent-delete a0c1c21c-d4f4-4577-9ec7-908f2d48622d
Deleted agent: a0c1c21c-d4f4-4577-9ec7-908f2d48622d
$ quantum agent-list
```

id	admin_state_up	agent_type	host	alive
1b69828d-6a9b-4826-87cd-1757f0e27f31	True	Linux bridge agent	HostA	:-)
ed96b856-ae0f-4d75-bb28-40a47ffd7695	True	Linux bridge agent	HostB	:-)
f28aa126-6edb-4ea5-a81e-8850876bc0a8	True	DHCP agent	HostB	:-)

```
+-----+-----+-----+-----+
+-----+
```

After deletion, if we restart the DHCP agent, it will be on agent list again.

Appendix B. Core Configuration File Options

Table of Contents

quantum.conf	120
ovs_quantum_plugin.ini	124
linuxbridge_conf.ini	126
dhcp_agent.ini	127
l3_agent.ini	128
Common Device Manager Options	128

quantum.conf

Find the configuration settings for the OpenStack Networking services in `/etc/quantum/quantum.conf`.

Debugging Options

Table B.1. Debugging Options

Configuration option=Default value	(Type) Description
debug=False	(BoolOpt) Prints debugging output.
verbose=False	(BoolOpt) Prints more verbose output.

Logging Options

Table B.2. Logging Options

Configuration option=Default value	(Type) Description
log_config=	(StrOpt) The logging configuration file. Overrides any other logging options specified. See the Python logging module documentation for details on logging configuration files.
log_format=%(asctime)s %(levelname)s [%(name)s] %(message)s	(StrOpt) A logging.Formatter log message format string that can use any of the available logging.LogRecord attributes.
log_date_format=%Y-%m-%d %H:%M:%S	(StrOpt) Format string for %(asctime)s in log records.
log_file=	(StrOpt) (Optional) Name of log file to output to. If not set, logging goes to stdout.
log_dir=	(StrOpt) (Optional) The directory to keep log files in (will be prepended to <code>-logfile</code>).
use_syslog=False	(BoolOpt) Uses syslog for logging.
syslog_log_facility=LOG_USER	(StrOpt) The syslog facility to receive log lines.

Authentication Options

The `auth_token` middleware for keystone now enables you to configure `auth_token` in the `quantum.conf` file. You no longer have to edit the `api-paste.ini` file. This change does not break backward compatibility. The `auth_token` middleware first tries the configurations in `/etc/quantum/api-paste.ini` and then tries the `quantum.conf` configuration. If you currently use `api-paste.ini`, you do not need to change it.

Table B.3. Authentication Options

Configuration option=Default value	(Type) Description
<code>auth_host = 127.0.0.1</code>	(StrOpt) Authentication listening host.
<code>auth_port = 35357</code>	(IntOpt) Authentication listening port.
<code>auth_protocol = http</code>	(StrOpt) Authentication protocol.
<code>admin_tenant_name =</code>	(StrOpt) The administrative tenant name.
<code>admin_user =</code>	(StrOpt) The administrative user name.
<code>admin_password =</code>	(StrOpt) The password for the administrative user.
<code>signing_dir = /var/lib/quantum/keystone-signing</code>	(StrOpt) The signing directory.

Service Options



Table B.4. Service Options

Configuration option=Default value	(Type) Description
<code>bind_host=0.0.0.0</code>	(StrOpt) Server listening IP.
<code>bind_port=9696</code>	(IntOpt) Server listening port.
<code>api_paste_config=api-paste.ini</code>	(StrOpt) The paste configuration file, which configures the WSGI application.
<code>api_extensions_path=</code>	(StrOpt) Enables custom addition to be made to the above configuration.
<code>policy_file=policy.json</code>	(StrOpt) JSON file representing policies to access and view data. The usage and format is discussed in more detail in the Authentication and Authorization section.
<code>auth_strategy=keystone</code>	(StrOpt) The strategy used for authentication. The supported values are 'keystone' and 'noauth'.
<code>core_plugin=quantum.plugins.sample.SamplePlugin.FakePlugin</code>	(StrOpt) The plugin to be loaded by the service.
<code>pagination_max_limit=1</code>	(StrOpt) The maximum number of items returned in a single response. A value of 'infinite', or a negative integer means no limit.

Plugin Options

Table B.5. Base Plugin Options

Configuration option=Default value	(Type) Description
<code>base_mac=fa:16:3e:00:00:00</code>	(StrOpt) MAC addresses for a port are generated. The first 3 octets will remain unchanged. If the 4h octet is not 00, it will also be used. The others will be randomly generated.
<code>mac_generation_retries=16</code>	(IntOpt) The number of times the plugin attempts to generate a unique MAC address.

Configuration option=Default value	(Type) Description
allow_bulk=True	(BoolOpt) Enables or disables bulk create/update/delete operations.
allow_pagination=False	<p>(BoolOpt) Enables or disables pagination. If plugin doesn't support native pagination, it will enable emulated pagination. Please note native pagination depends on native sorting. If native pagination is enabled, native emulated sorting will be enabled automatically.</p> <div>  <p>Note</p> <p>If the plugin supports native pagination, the plugin will return the maximum limit of items as requested. If the plugin didn't support that, quantum API can emulate the pagination behavior. The performance of native pagination is better than emulated pagination.</p> </div>
allow_sorting=False	<p>(BoolOpt) Enables or disables sorting. If plugin doesn't support native sorting, it will enable emulated sorting.</p> <div>  <p>Note</p> <p>If the plugin supports native sorting, the plugin will return ordered items as requested. If the plugin didn't support that, quantum API can emulate the sorting behavior. The performance of native sorting is better than emulated sorting.</p> </div>
max_dns_nameservers=5	(IntOpt) The maximum amount of DNS nameservers that can be configured per subnet.
max_subnet_host_routes=20	(IntOpt) The maximum amount of host routes that can be configured per subnet.
state_path=.	(StrOpt) Top level directory for configuration files.
dhcp_lease_duration=120	(IntOpt) The default expiration time, in seconds, for a DHCP address.

Common RPC Message Options

Table B.6. Common RPC Message Options

Configuration option=Default value	(Type) Description
control_exchange=quantum	(StrOpt) AMQP exchange to connect to if using RabbitMQ or QPID.
rpc_back_end=quantum.openstack.common.rpc.impl_kombu	(StrOpt) The messaging module to use, defaults to kombu. For qpid, make use of quantum.openstack.common.rpc.impl_qpid.
rpc_thread_pool_size=64	(IntOpt) Size of RPC thread pool.
rpc_conn_pool_size=30	(IntOpt) Size of RPC connection pool.
rpc_response_timeout=60	(IntOpt) Seconds to wait for a response from call or multi call.
allowed_rpc_exception_modules='quantum.openstack.common.rpc.exception', 'nova.exception'	(ListOpt) Modules of exceptions that are permitted to be recreated upon receiving exception data from an rpc call.
fake_rabbit=False	(BoolOpt) If passed, use a fake RabbitMQ provider.

Rabbit RPC Options

Table B.7. Rabbit RPC Options

Configuration option=Default value	(Type) Description
kombu_ssl_version=	(StrOpt) SSL version to use (valid only if SSL enabled).
kombu_ssl_keyfile=	(StrOpt) SSL key file (valid only if SSL enabled).
kombu_ssl_certfile=	(StrOpt) SSL cert file (valid only if SSL enabled).
kombu_ssl_ca_certs=	(StrOpt) SSL certification authority file (valid only if SSL enabled).
rabbit_host=localhost	(StrOpt) IP address of the RabbitMQ installation.
rabbit_password=guest	Password of the RabbitMQ server.
rabbit_port=5672	(IntOpt) Port where RabbitMQ server is running/listening.
rabbit_userid=guest	(StrOpt) User ID used for RabbitMQ connections.
rabbit_virtual_host=/	(StrOpt) Location of a virtual RabbitMQ installation.
rabbit_max_retries=0	(IntOpt) Maximum retries with trying to connect to RabbitMQ. The default of 0 implies an infinite retry count.
rabbit_retry_interval=1	(IntOpt) RabbitMQ connection retry interval.

QPID RPC Options

Table B.8. QPID RPC Options

Configuration option=Default value	(Type) Description
qpid_hostname=localhost	(StrOpt) Qpid broker hostname.
qpid_port=5672	(IntOpt) Qpid broker port.
qpid_username=	(StrOpt) Username for qpid connection.
qpid_password=	(StrOpt) Password for qpid connection.
qpid_sasl_mechanisms=	(StrOpt) Space separated list of SASL mechanisms to use for auth.
qpid_reconnect=True	(BoolOpt) Automatically reconnect.
qpid_reconnect_timeout=0	(IntOpt) The number of seconds to wait before deciding that a reconnect attempt has failed.
qpid_reconnect_limit=0	(IntOpt) The limit for the number of times to reconnect before considering the connection to be failed.
qpid_reconnect_interval_min=0	(IntOpt) Minimum seconds between reconnection attempts.
qpid_reconnect_interval_max=0	(IntOpt) Maximum seconds between reconnection attempts.
qpid_reconnect_interval=0	(IntOpt) Equivalent to setting max and min to the same value.
qpid_heartbeat=60	(IntOpt) Seconds between connection keepalive heartbeats.
qpid_protocol=tcp	(StrOpt) Transport to use, either 'tcp' or 'ssl.'
qpid_tcp_nodelay=True	(BoolOpt) Disable Nagle algorithm.

Notification Options

Table B.9. Notification Options

Configuration option=Default value	(Type) Description
notification_driver=quantum.openstack.common.notifier.list_notifier	(ListOpt) Driver or drivers to handle sending notifications. The default is set as notifier as the DHCP agent makes use of the notifications.
default_notification_level=INFO	(StrOpt) Default notification level for outgoing notifications.
default_publisher_id=\$host	(StrOpt) Default publisher_id for outgoing notifications.
list_notifier_drivers='quantum.openstack.common.notifier.list_notifier'	(ListOpt) List of drivers to send notifications.
notification_topics='notifications'	(ListOpt) AMQP topic used for openstack notifications.

Quota Options

Table B.10. Quota Options

Configuration option=Default value	(Type) Description
quota_driver=quantum.quota.ConfDriver	(StrOpt) Default driver to use for quota checks. If the default driver is used then the configuration values below are in effect. To limit quotas per tenant then use: <code>quantum.db.quota_db.DbQuotaDriver</code>
quota_items=network,subnet,port	(ListOpt) Resource names that are supported by the Quotas feature.
default_quota=-1	(IntOpt) Default number of resources allowed per tenant, minus for unlimited.
quota_network=10	(IntOpt) Number of networks allowed per tenant, and minus means unlimited.
quota_subnet=10	(IntOpt) Number of subnets allowed per tenant, and minus means unlimited.
quota_port=50	(IntOpt) Number of ports allowed per tenant, and minus means unlimited.


ovs_quantum_plugin.ini

For information about the Open vSwitch plugin configurations, see <http://wiki.openstack.org/ConfigureOpenvswitch>.

Database Access by Plugin

Table B.11. Database Access by Plugin

Configuration option=Default value	(Type) Description
sql_connection=sqlite://	(StrOpt) The details of the database connection. For example <code>mysql://root:nova@127.0.0.1:3306/ovs_quantum</code> . Replace 127.0.0.1 above with the IP address of the database used by the main OpenStack Networking server. (Leave it as is if the database runs on this host.).

Configuration option=Default value	(Type) Description
	 Note Change this line to ensure that the database values are persistent. The sqlite is used for testing.
sql_max_retries=10	(IntOpt) The number of database re-connection retry times. Used if connectivity is lost with the database. -1 implies an infinite retry count.
reconnect_interval=2	(IntOpt) The database reconnection interval in seconds. Used if connectivity is lost.

OVS Options

Specify these parameters in the `OVS` section.

These OVS options are common to the plugin and agent.

Table B.12. OVS Options Common to Plugin and Agent

Configuration option=Default value	(Type) Description
network_vlan_ranges=default:2000:3999	(ListOpt) Comma-separated list of <physical_network>:<vlan_min>:<vlan_max> tuples enumerating ranges of VLAN IDs on named physical networks that are available for allocation.
tunnel_id_ranges=	(ListOpt) Comma-separated list of <tun_min>:<tun_max> tuples enumerating ranges of GRE tunnel IDs that are available for allocation.
integration_bridge=br-int	(StrOpt) The name of the OVS integration bridge. There is one per hypervisor. The integration bridge acts as a virtual "patch port". All VM VIFs are attached to this bridge and then "patched" according to their network connectivity. Do not change this parameter unless you have a good reason to.
tunnel_bridge=br-tun	(StrOpt) The name of the OVS tunnel bridge used by the agent for GRE tunnels. Only used if tunnel_id_ranges is not empty.
bridge_mappings=default:br-eth1	(ListOpt) Comma-separated list of <physical_network>:<bridge> tuples mapping physical network names to agent's node-specific OVS bridge names. Each bridge must exist, and should have physical network # interface configured as a port.
local_ip=10.0.0.3	(StrOpt) The local IP address of this hypervisor. Used only when tunnel_id_ranges are used.
enable_tunneling=False	(BoolOpt) A flag indicating if tunneling is supported. Not all systems that support Open vSwitch support its GRE tunneling feature, that is, it is not supported in the Linux kernel source tree. This applies to both the server and agent.

Agent Options

Specify these options in the `AGENT` section.


Table B.13. Agent Options

Configuration option=Default value	(Type) Description
rpc=True	(BoolOpt) If <code>True</code> , the agent communicates with the plugin through the OpenStack RPC, which is configured in <code>quantum.conf</code> . If <code>False</code> , the agent polls the database for changes. If <code>False</code> , you must update the relevant database settings on the agent so that it can access the database.
polling_interval=2	(IntOpt) Agent's polling interval in seconds.
root_helper=sudo	(StrOpt) Limits the commands that can be run. See the <code>rootwrap</code> section for more details.

linuxbridge_conf.ini

Database Access by Plugin

Table B.14. Database Access by Plugin

Configuration option=Default value	(Type) Description
sql_connection=sqlite://	(StrOpt) The details of the database connection. For example <code>mysql://root:nova@127.0.0.1:3306/ovs_quantum</code> . Replace <code>127.0.0.1</code> above with the IP address of the database used by the main OpenStack Networking server. (Leave it as is if the database runs on this host.).  Note This line must be changed to ensure that the database values are persistent. The <code>sqlite</code> is used for testing.
sql_max_retries=10	(IntOpt) Database re-connection retry times. Used if connectivity is lost with the database. -1 implies an infinite retry count.
reconnect_interval=2	(IntOpt) Database reconnection interval in seconds. Used if connectivity is lost.

VLAN Configurations

Table B.15. VLAN Configurations

Configuration option=Default value	(Type) Description
network_vlan_ranges=default:1000:2999	(ListOpt) Comma-separated list of <code><physical_network>:<vlan_min>:<vlan_max></code> tuples enumerating ranges of VLAN IDs on named physical networks that are available for allocation.

Networking Options on the Agent

Table B.16. Networking Options on the Agent

Configuration option=Default value	(Type) Description
physical_interface_mappings=default:eth1	(ListOpt) Comma-separated list of <code><physical_network>:<physical_interface></code> tuples mapping

Configuration option=Default value	(Type) Description
	physical network names to agent's node-specific physical network interfaces. Server uses physical network names for validation but ignores interfaces.

Agent Options

Table B.17. Agent Options


Configuration option=Default value	(Type) Description
rpc=True	(BoolOpt) If <code>True</code> , the agent communicates with the plugin through the OpenStack RPC, which is configured in <code>quantum.conf</code> . If <code>False</code> , the agent polls the database for changes. If <code>False</code> , you must update the relevant database settings on the agent so that it can access the database.
polling_interval=2	(IntOpt) Agent's polling interval in seconds.
root_helper=sudo	(StrOpt) Limits the commands that can be run. See the rootwrap section for more details.

dhcp_agent.ini

For device manager options, see [the section called “Common Device Manager Options” \[128\]](#).

DHCP-Specific Options

Table B.18. DHCP-Specific Options

Configuration option=Default value	(Type) Description
root_helper=sudo	(StrOpt) Limits the commands that can be run. See the rootwrap section for more details.
dhcp_driver=quantum.agent.linux.dhcp.Dnsmasq	(StrOpt) The driver used to manage the DHCP server.
dhcp_lease_relay_socket=\$state_path/dhcp/lease_relay	(StrOpt) Location to DHCP lease relay UNIX domain socket.
use_namespaces=True	<div> <div>(BoolOpt) Allows overlapping IP.</div> <div>  <div> Note <p>If you run multiple agents with different IP addresses on the same host, set this parameter to <code>True</code>. Otherwise, routing problems occur.</p> </div> </div> </div>

dnsmasq Options

Table B.19. dnsmasq Options

Configuration option=Default value	(Type) Description
dhcp_confs=\$state_path/dhcp	(StrOpt) Location to store DHCP server config files.
dhcp_lease_time=120	(IntOpt) Lifetime of a DHCP lease in seconds.

Configuration option=Default value	(Type) Description
dhcp_domain=openstacklocal	(StrOpt) Domain to use for building the host names.
dnsmasq_config_file=	(StrOpt) Overrides the default dnsmasq settings with this file.
dnsmasq_dns_server=	(StrOpt) Specifies whether to use another DNS server before any in <code>/etc/resolv.conf</code> .

I3_agent.ini

For device manager options, see [the section called “Common Device Manager Options” \[128\]](#).

Specific Options

Table B.20. L3 Specific Options

Configuration option=Default value	(Type) Description
root_helper=sudo	(StrOpt) Limits the commands that can be run. See the rootwrap section for more details.
external_network_bridge=br-ex	(StrOpt) Name of bridge used for external network traffic.
use_namespaces=True	(BoolOpt) Allows overlapping IP. <div data-bbox="889 947 959 1024" data-label="Image"></div> <div data-bbox="1015 938 1105 974" data-label="Section-Header">Note</div> <div data-bbox="1015 1001 1399 1104" data-label="Text"> <p>If you run multiple agents with different IP addresses on the same host, set this parameter to <code>True</code>. Otherwise, routing problems occur.</p> </div>
polling_interval=3	(IntOpt) The time, in seconds, between state poll requests.
metadata_ip=	(StrOpt) IP address used by OpenStack Compute metadata server.
metadata_port=8775	(IntOpt) TCP Port used by OpenStack Compute metadata server.
router_id=	(StrOpt) If namespaces is disabled, the l3 agent can only configure a router whose ID matches this parameter.
handle_internal_only_routers=True	(BoolOpt) Agent should implement routers with no gateway.
gateway_external_network_id=	(StrOpt) UUID of external network for routers implemented by the agents.

Common Device Manager Options

Use the following device manager options in the `dhcp_agent.ini` file for the DHCP agent or the `l3_agent.ini` file for the L3 agent.

Table B.21. Device Manager Options

Configuration option=Default value	(Type) Description
interface_driver=	(StrOpt) The driver used to manage the virtual interface.
ovs_use_veth=False	(BoolOpt) Specifies whether to use veth for an interface. Set to <code>True</code> for OVS-based plugins that use Open vSwitch as OpenFlow switch and check port status.

Configuration option=Default value	(Type) Description
admin_user=	(StrOpt) The administrative user name for OpenStack Networking, which is defined in OpenStack Identity (keystone). Only relevant if using MetaPlugin.
admin_password=	(StrOpt) The password for the administrative user. Only relevant if using MetaPlugin.
admin_tenant_name=	(StrOpt) The administrative user's tenant name. Only relevant if using MetaPlugin.
auth_url=	(StrOpt) The URL used to validate tokens. For example, <code>`auth_protocol`://`auth_host`:`auth_port`/v2.0</code> . Only relevant if using MetaPlugin.
auth_strategy=keystone	(StrOpt) The strategy to use for authentication. Supports noauth or keystone. Only relevant if using MetaPlugin.
auth_region=	(StrOpt) The authentication region. Only relevant if using MetaPlugin.
ovs_integration_bridge=br-int	(StrOpt) Name of Open vSwitch bridge to use. Only relevant if using Open vSwitch.
network_device_mtu=	(StrOpt) MTU setting for device. Only relevant if using Open vSwitch.
meta_flavor_driver_mappings=	(StrOpt). Mappings between flavors and drivers. Only relevant if using MetaPlugin.
resync_interval=30	(IntOpt) If an exception occurs on the quantum-server service, the DHCP agent ensures that it syncs with the <code>quantum.conf</code> configuration. The validation about syncing occurs every <code>resync_interval</code> seconds.

Appendix C. Plugin pagination and sorting support

Table C.1. The plugins are supporting native pagination and sorting

Plugin	Support Native Pagination	Support Native Sorting
Open vSwitch	True	True
LinuxBridge	True	True