



Networking Guide

Release Version: 15.0.0

OpenStack contributors

Jun 12, 2017

CONTENTS

Abstract	1
Contents	2
Conventions	2
Introduction	2
Configuration	20
Deployment examples	171
Operations	283
Migration	289
Miscellaneous	295
Appendix	302
Community support	302
Glossary	306
Glossary	306
Index	341

ABSTRACT

This guide targets OpenStack administrators seeking to deploy and manage OpenStack Networking (neutron).

This guide documents the OpenStack Ocata release.

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note: A comment with additional information that explains a part of the text.

Important: Something you must be aware of before proceeding.

Tip: An extra but helpful piece of practical advice.

Caution: Helpful information that prevents the user from making mistakes.

Warning: Critical information about the risk of data loss or security issues.

Command prompts

```
$ command
```

Any user, including the `root` user, can run commands that are prefixed with the `$` prompt.

```
# command
```

The `root` user must run commands that are prefixed with the `#` prompt. You can also prefix these commands with the `sudo` command, if available, to run them.

Introduction

The OpenStack *Networking service* provides an API that allows users to set up and define network connectivity and addressing in the cloud. The project code-name for Networking services is neutron. OpenStack Networking handles the creation and management of a virtual networking infrastructure, including networks, switches, subnets, and routers for devices managed by the OpenStack Compute service (nova). Advanced services such as firewalls or *virtual private networks (VPNs)* can also be used.

OpenStack Networking consists of the neutron-server, a database for persistent storage, and any number of plug-in agents, which provide other services such as interfacing with native Linux networking mechanisms, external devices, or SDN controllers.

OpenStack Networking is entirely standalone and can be deployed to a dedicated host. If your deployment uses a controller host to run centralized Compute components, you can deploy the Networking server to that specific host instead.

OpenStack Networking integrates with various OpenStack components:

- OpenStack *Identity service (keystone)* is used for authentication and authorization of API requests.
- OpenStack *Compute service (nova)* is used to plug each virtual NIC on the VM into a particular network.
- OpenStack *Dashboard (horizon)* is used by administrators and project users to create and manage network services through a web-based graphical interface.

Note: The network address ranges used in this guide are chosen in accordance with RFC 5737 and RFC 3849, and as such are restricted to the following:

IPv4:

- 192.0.2.0/24
- 198.51.100.0/24
- 203.0.113.0/24

IPv6:

- 2001:DB8::/32

The network address ranges in the examples of this guide should not be used for any purpose other than documentation.

Note: To reduce clutter, this guide removes command output without relevance to the particular action.

Basic networking

Ethernet

Ethernet is a networking protocol, specified by the IEEE 802.3 standard. Most wired network interface cards (NICs) communicate using Ethernet.

In the [OSI model](#) of networking protocols, Ethernet occupies the second layer, which is known as the data link layer. When discussing Ethernet, you will often hear terms such as *local network*, *layer 2*, *L2*, *link layer* and *data link layer*.

In an Ethernet network, the hosts connected to the network communicate by exchanging *frames*. Every host on an Ethernet network is uniquely identified by an address called the media access control (MAC) address. In particular, every virtual machine instance in an OpenStack environment has a unique MAC address, which is different from the MAC address of the compute host. A MAC address has 48 bits and is typically represented as a hexadecimal string, such as 08:00:27:b9:88:74. The MAC address is hard-coded into the NIC by the

manufacturer, although modern NICs allow you to change the MAC address programmatically. In Linux, you can retrieve the MAC address of a NIC using the `ip` command:

```
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT
    ↳ group default qlen 1000
        link/ether 08:00:27:b9:88:74 brd ff:ff:ff:ff:ff:ff
```

Conceptually, you can think of an Ethernet network as a single bus that each of the network hosts connects to. In early implementations, an Ethernet network consisted of a single coaxial cable that hosts would tap into to connect to the network. However, network hosts in modern Ethernet networks connect directly to a network device called a *switch*. Still, this conceptual model is useful, and in network diagrams (including those generated by the OpenStack dashboard) an Ethernet network is often depicted as if it was a single bus. You'll sometimes hear an Ethernet network referred to as a *layer 2 segment*.

In an Ethernet network, every host on the network can send a frame directly to every other host. An Ethernet network also supports broadcasts so that one host can send a frame to every host on the network by sending to the special MAC address `ff:ff:ff:ff:ff:ff`. [ARP](#) and [DHCP](#) are two notable protocols that use Ethernet broadcasts. Because Ethernet networks support broadcasts, you will sometimes hear an Ethernet network referred to as a *broadcast domain*.

When a NIC receives an Ethernet frame, by default the NIC checks to see if the destination MAC address matches the address of the NIC (or the broadcast address), and the Ethernet frame is discarded if the MAC address does not match. For a compute host, this behavior is undesirable because the frame may be intended for one of the instances. NICs can be configured for *promiscuous mode*, where they pass all Ethernet frames to the operating system, even if the MAC address does not match. Compute hosts should always have the appropriate NICs configured for promiscuous mode.

As mentioned earlier, modern Ethernet networks use switches to interconnect the network hosts. A switch is a box of networking hardware with a large number of ports that forward Ethernet frames from one connected host to another. When hosts first send frames over the switch, the switch doesn't know which MAC address is associated with which port. If an Ethernet frame is destined for an unknown MAC address, the switch broadcasts the frame to all ports. The switch learns which MAC addresses are at which ports by observing the traffic. Once it knows which MAC address is associated with a port, it can send Ethernet frames to the correct port instead of broadcasting. The switch maintains the mappings of MAC addresses to switch ports in a table called a *forwarding table* or *forwarding information base* (FIB). Switches can be daisy-chained together, and the resulting connection of switches and hosts behaves like a single network.

VLANs

VLAN is a networking technology that enables a single switch to act as if it was multiple independent switches. Specifically, two hosts that are connected to the same switch but on different VLANs do not see each other's traffic. OpenStack is able to take advantage of VLANs to isolate the traffic of different projects, even if the projects happen to have instances running on the same compute host. Each VLAN has an associated numerical ID, between 1 and 4095. We say "VLAN 15" to refer to the VLAN with a numerical ID of 15.

To understand how VLANs work, let's consider VLAN applications in a traditional IT environment, where physical hosts are attached to a physical switch, and no virtualization is involved. Imagine a scenario where you want three isolated networks but you only have a single physical switch. The network administrator would choose three VLAN IDs, for example, 10, 11, and 12, and would configure the switch to associate switchports with VLAN IDs. For example, switchport 2 might be associated with VLAN 10, switchport 3 might be associated with VLAN 11, and so forth. When a switchport is configured for a specific VLAN, it is called an *access port*. The switch is responsible for ensuring that the network traffic is isolated across the VLANs.

Now consider the scenario that all of the switchports in the first switch become occupied, and so the organization buys a second switch and connects it to the first switch to expand the available number of switchports. The second switch is also configured to support VLAN IDs 10, 11, and 12. Now imagine host A connected to switch 1 on a port configured for VLAN ID 10 sends an Ethernet frame intended for host B connected to switch 2 on a port configured for VLAN ID 10. When switch 1 forwards the Ethernet frame to switch 2, it must communicate that the frame is associated with VLAN ID 10.

If two switches are to be connected together, and the switches are configured for VLANs, then the switchports used for cross-connecting the switches must be configured to allow Ethernet frames from any VLAN to be forwarded to the other switch. In addition, the sending switch must tag each Ethernet frame with the VLAN ID so that the receiving switch can ensure that only hosts on the matching VLAN are eligible to receive the frame.

A switchport that is configured to pass frames from all VLANs and tag them with the VLAN IDs is called a *trunk port*. IEEE 802.1Q is the network standard that describes how VLAN tags are encoded in Ethernet frames when trunking is being used.

Note that if you are using VLANs on your physical switches to implement project isolation in your OpenStack cloud, you must ensure that all of your switchports are configured as trunk ports.

It is important that you select a VLAN range not being used by your current network infrastructure. For example, if you estimate that your cloud must support a maximum of 100 projects, pick a VLAN range outside of that value, such as VLAN 200–299. OpenStack, and all physical network infrastructure that handles project networks, must then support this VLAN range.

Trunking is used to connect between different switches. Each trunk uses a tag to identify which VLAN is in use. This ensures that switches on the same VLAN can communicate.

Subnets and ARP

While NICs use MAC addresses to address network hosts, TCP/IP applications use IP addresses. The Address Resolution Protocol (ARP) bridges the gap between Ethernet and IP by translating IP addresses into MAC addresses.

IP addresses are broken up into two parts: a *network number* and a *host identifier*. Two hosts are on the same *subnet* if they have the same network number. Recall that two hosts can only communicate directly over Ethernet if they are on the same local network. ARP assumes that all machines that are in the same subnet are on the same local network. Network administrators must take care when assigning IP addresses and netmasks to hosts so that any two hosts that are in the same subnet are on the same local network, otherwise ARP does not work properly.

To calculate the network number of an IP address, you must know the *netmask* associated with the address. A netmask indicates how many of the bits in the 32-bit IP address make up the network number.

There are two syntaxes for expressing a netmask:

- dotted quad
- classless inter-domain routing (CIDR)

Consider an IP address of 192.168.1.5, where the first 24 bits of the address are the network number. In dotted quad notation, the netmask would be written as 255.255.255.0. CIDR notation includes both the IP address and netmask, and this example would be written as 192.168.1.5/24.

Note: Creating CIDR subnets including a multicast address or a loopback address cannot be used in an OpenStack environment. For example, creating a subnet using 224.0.0.0/16 or 127.0.1.0/24 is not supported.

Sometimes we want to refer to a subnet, but not any particular IP address on the subnet. A common convention is to set the host identifier to all zeros to make reference to a subnet. For example, if a host's IP address is 10.10.53.24/16, then we would say the subnet is 10.10.0.0/16.

To understand how ARP translates IP addresses to MAC addresses, consider the following example. Assume host *A* has an IP address of 192.168.1.5/24 and a MAC address of fc:99:47:49:d4:a0, and wants to send a packet to host *B* with an IP address of 192.168.1.7. Note that the network number is the same for both hosts, so host *A* is able to send frames directly to host *B*.

The first time host *A* attempts to communicate with host *B*, the destination MAC address is not known. Host *A* makes an ARP request to the local network. The request is a broadcast with a message like this:

To: everybody (ff:ff:ff:ff:ff:ff). I am looking for the computer who has IP address 192.168.1.7. Signed: MAC address fc:99:47:49:d4:a0.

Host *B* responds with a response like this:

To: fc:99:47:49:d4:a0. I have IP address 192.168.1.7. Signed: MAC address 54:78:1a:86:00:a5.

Host *A* then sends Ethernet frames to host *B*.

You can initiate an ARP request manually using the **arping** command. For example, to send an ARP request to IP address 10.30.0.132:

```
$ arping -I eth0 10.30.0.132
ARPING 10.30.0.132 from 10.30.0.131 eth0
Unicast reply from 10.30.0.132 [54:78:1A:86:1C:0B] 0.670ms
Unicast reply from 10.30.0.132 [54:78:1A:86:1C:0B] 0.722ms
Unicast reply from 10.30.0.132 [54:78:1A:86:1C:0B] 0.723ms
Sent 3 probes (1 broadcast(s))
Received 3 response(s)
```

To reduce the number of ARP requests, operating systems maintain an ARP cache that contains the mappings of IP addresses to MAC address. On a Linux machine, you can view the contents of the ARP cache by using the **arp** command:

\$ arp -n	Address	Hwtype	Hwaddress	Flags	Mask	Iface
	10.0.2.3	ether	52:54:00:12:35:03	C		eth0
	10.0.2.2	ether	52:54:00:12:35:02	C		eth0

DHCP

Hosts connected to a network use the *Dynamic Host Configuration Protocol (DHCP)* to dynamically obtain IP addresses. A DHCP server hands out the IP addresses to network hosts, which are the DHCP clients.

DHCP clients locate the DHCP server by sending a *UDP* packet from port 68 to address 255.255.255.255 on port 67. Address 255.255.255.255 is the local network broadcast address: all hosts on the local network see the UDP packets sent to this address. However, such packets are not forwarded to other networks. Consequently, the DHCP server must be on the same local network as the client, or the server will not receive the broadcast. The DHCP server responds by sending a UDP packet from port 67 to port 68 on the client. The exchange looks like this:

1. The client sends a discover (“I’m a client at MAC address 08:00:27:b9:88:74, I need an IP address”)
2. The server sends an offer (“OK 08:00:27:b9:88:74, I’m offering IP address 10.10.0.112”)

3. The client sends a request (“Server 10.10.0.131, I would like to have IP 10.10.0.112”)
4. The server sends an acknowledgement (“OK 08:00:27:b9:88:74, IP 10.10.0.112 is yours”)

OpenStack uses a third-party program called `dnsmasq` to implement the DHCP server. `Dnsmasq` writes to the syslog, where you can observe the DHCP request and replies:

```
Apr 23 15:53:46 c100-1 dhcpd: DHCPDISCOVER from 08:00:27:b9:88:74 via eth2
Apr 23 15:53:46 c100-1 dhcpd: DHCPOFFER on 10.10.0.112 to 08:00:27:b9:88:74 via eth2
Apr 23 15:53:48 c100-1 dhcpd: DHCPREQUEST for 10.10.0.112 (10.10.0.131) from
  ↳ 08:00:27:b9:88:74 via eth2
Apr 23 15:53:48 c100-1 dhcpd: DHCPACK on 10.10.0.112 to 08:00:27:b9:88:74 via eth2
```

When troubleshooting an instance that is not reachable over the network, it can be helpful to examine this log to verify that all four steps of the DHCP protocol were carried out for the instance in question.

IP

The Internet Protocol (IP) specifies how to route packets between hosts that are connected to different local networks. IP relies on special network hosts called *routers* or *gateways*. A router is a host that is connected to at least two local networks and can forward IP packets from one local network to another. A router has multiple IP addresses: one for each of the networks it is connected to.

In the OSI model of networking protocols IP occupies the third layer, known as the network layer. When discussing IP, you will often hear terms such as *layer 3*, *L3*, and *network layer*.

A host sending a packet to an IP address consults its *routing table* to determine which machine on the local network(s) the packet should be sent to. The routing table maintains a list of the subnets associated with each local network that the host is directly connected to, as well as a list of routers that are on these local networks.

On a Linux machine, any of the following commands displays the routing table:

```
$ ip route show
$ route -n
$ netstat -rn
```

Here is an example of output from `ip route show`:

```
$ ip route show
default via 10.0.2.2 dev eth0
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
192.168.27.0/24 dev eth1 proto kernel scope link src 192.168.27.100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
```

Line 1 of the output specifies the location of the default route, which is the effective routing rule if none of the other rules match. The router associated with the default route (10.0.2.2 in the example above) is sometimes referred to as the *default gateway*. A *DHCP* server typically transmits the IP address of the default gateway to the *DHCP* client along with the client’s IP address and a netmask.

Line 2 of the output specifies that IPs in the 10.0.2.0/24 subnet are on the local network associated with the network interface eth0.

Line 3 of the output specifies that IPs in the 192.168.27.0/24 subnet are on the local network associated with the network interface eth1.

Line 4 of the output specifies that IPs in the 192.168.122.0/24 subnet are on the local network associated with the network interface virbr0.

The output of the **route -n** and **netstat -rn** commands are formatted in a slightly different way. This example shows how the same routes would be formatted using these commands:

```
$ route -n
Kernel IP routing table
Destination      Gateway        Genmask        Flags   MSS Window irtt Iface
0.0.0.0          10.0.2.2      0.0.0.0       UG        0 0          0 eth0
10.0.2.0         0.0.0.0       255.255.255.0 U          0 0          0 eth0
192.168.27.0     0.0.0.0       255.255.255.0 U          0 0          0 eth1
192.168.122.0    0.0.0.0       255.255.255.0 U          0 0          0 virbr0
```

The **ip route get** command outputs the route for a destination IP address. From the below example, destination IP address **10.0.2.14** is on the local network of **eth0** and would be sent directly:

```
$ ip route get 10.0.2.14
10.0.2.14 dev eth0  src 10.0.2.15
```

The destination IP address **93.184.216.34** is not on any of the connected local networks and would be forwarded to the default gateway at **10.0.2.2**:

```
$ ip route get 93.184.216.34
93.184.216.34 via 10.0.2.2 dev eth0  src 10.0.2.15
```

It is common for a packet to hop across multiple routers to reach its final destination. On a Linux machine, the **traceroute** and more recent **mtr** programs prints out the IP address of each router that an IP packet traverses along its path to its destination.

TCP/UDP/ICMP

For networked software applications to communicate over an IP network, they must use a protocol layered atop IP. These protocols occupy the fourth layer of the OSI model known as the *transport layer* or *layer 4*. See the [Protocol Numbers](#) web page maintained by the Internet Assigned Numbers Authority (IANA) for a list of protocols that layer atop IP and their associated numbers.

The *Transmission Control Protocol* (TCP) is the most commonly used layer 4 protocol in networked applications. TCP is a *connection-oriented* protocol: it uses a client-server model where a client connects to a server, where *server* refers to the application that receives connections. The typical interaction in a TCP-based application proceeds as follows:

1. Client connects to server.
2. Client and server exchange data.
3. Client or server disconnects.

Because a network host may have multiple TCP-based applications running, TCP uses an addressing scheme called *ports* to uniquely identify TCP-based applications. A TCP port is associated with a number in the range 1-65535, and only one application on a host can be associated with a TCP port at a time, a restriction that is enforced by the operating system.

A TCP server is said to *listen* on a port. For example, an SSH server typically listens on port 22. For a client to connect to a server using TCP, the client must know both the IP address of a server's host and the server's TCP port.

The operating system of the TCP client application automatically assigns a port number to the client. The client owns this port number until the TCP connection is terminated, after which the operating system reclaims the

port number. These types of ports are referred to as *ephemeral ports*.

IANA maintains a [registry of port numbers](#) for many TCP-based services, as well as services that use other layer 4 protocols that employ ports. Registering a TCP port number is not required, but registering a port number is helpful to avoid collisions with other services. See [firewalls and default ports](#) in OpenStack Administrator Guide for the default TCP ports used by various services involved in an OpenStack deployment.

The most common application programming interface (API) for writing TCP-based applications is called *Berkeley sockets*, also known as *BSD sockets* or, simply, *sockets*. The sockets API exposes a *stream oriented* interface for writing TCP applications. From the perspective of a programmer, sending data over a TCP connection is similar to writing a stream of bytes to a file. It is the responsibility of the operating system's TCP/IP implementation to break up the stream of data into IP packets. The operating system is also responsible for automatically retransmitting dropped packets, and for handling flow control to ensure that transmitted data does not overrun the sender's data buffers, receiver's data buffers, and network capacity. Finally, the operating system is responsible for re-assembling the packets in the correct order into a stream of data on the receiver's side. Because TCP detects and retransmits lost packets, it is said to be a *reliable* protocol.

The *User Datagram Protocol* (UDP) is another layer 4 protocol that is the basis of several well-known networking protocols. UDP is a *connectionless* protocol: two applications that communicate over UDP do not need to establish a connection before exchanging data. UDP is also an *unreliable* protocol. The operating system does not attempt to retransmit or even detect lost UDP packets. The operating system also does not provide any guarantee that the receiving application sees the UDP packets in the same order that they were sent in.

UDP, like TCP, uses the notion of ports to distinguish between different applications running on the same system. Note, however, that operating systems treat UDP ports separately from TCP ports. For example, it is possible for one application to be associated with TCP port 16543 and a separate application to be associated with UDP port 16543.

Like TCP, the sockets API is the most common API for writing UDP-based applications. The sockets API provides a *message-oriented* interface for writing UDP applications: a programmer sends data over UDP by transmitting a fixed-sized message. If an application requires retransmissions of lost packets or a well-defined ordering of received packets, the programmer is responsible for implementing this functionality in the application code.

[DHCP](#), the Domain Name System (DNS), the Network Time Protocol (NTP), and [Virtual extensible local area network \(VXLAN\)](#) are examples of UDP-based protocols used in OpenStack deployments.

UDP has support for one-to-many communication: sending a single packet to multiple hosts. An application can broadcast a UDP packet to all of the network hosts on a local network by setting the receiver IP address as the special IP broadcast address 255.255.255.255. An application can also send a UDP packet to a set of receivers using *IP multicast*. The intended receiver applications join a multicast group by binding a UDP socket to a special IP address that is one of the valid multicast group addresses. The receiving hosts do not have to be on the same local network as the sender, but the intervening routers must be configured to support IP multicast routing. VXLAN is an example of a UDP-based protocol that uses IP multicast.

The *Internet Control Message Protocol* (ICMP) is a protocol used for sending control messages over an IP network. For example, a router that receives an IP packet may send an ICMP packet back to the source if there is no route in the router's routing table that corresponds to the destination address (ICMP code 1, destination host unreachable) or if the IP packet is too large for the router to handle (ICMP code 4, fragmentation required and "don't fragment" flag is set).

The **ping** and **mtr** Linux command-line tools are two examples of network utilities that use ICMP.

Network components

Switches

Switches are Multi-Input Multi-Output (MIMO) devices that enable packets to travel from one node to another. Switches connect hosts that belong to the same layer-2 network. Switches enable forwarding of the packet received on one port (input) to another port (output) so that they reach the desired destination node. Switches operate at layer-2 in the networking model. They forward the traffic based on the destination Ethernet address in the packet header.

Routers

Routers are special devices that enable packets to travel from one layer-3 network to another. Routers enable communication between two nodes on different layer-3 networks that are not directly connected to each other. Routers operate at layer-3 in the networking model. They route the traffic based on the destination IP address in the packet header.

Firewalls

Firewalls are used to regulate traffic to and from a host or a network. A firewall can be either a specialized device connecting two networks or a software-based filtering mechanism implemented on an operating system. Firewalls are used to restrict traffic to a host based on the rules defined on the host. They can filter packets based on several criteria such as source IP address, destination IP address, port numbers, connection state, and so on. It is primarily used to protect the hosts from unauthorized access and malicious attacks. Linux-based operating systems implement firewalls through `iptables`.

Load balancers

Load balancers can be software-based or hardware-based devices that allow traffic to evenly be distributed across several servers. By distributing the traffic across multiple servers, it avoids overload of a single server thereby preventing a single point of failure in the product. This further improves the performance, network throughput, and response time of the servers. Load balancers are typically used in a 3-tier architecture. In this model, a load balancer receives a request from the front-end web server, which then forwards the request to one of the available back-end database servers for processing. The response from the database server is passed back to the web server for further processing.

Overlay (tunnel) protocols

Tunneling is a mechanism that makes transfer of payloads feasible over an incompatible delivery network. It allows the network user to gain access to denied or insecure networks. Data encryption may be employed to transport the payload, ensuring that the encapsulated user network data appears as public even though it is private and can easily pass the conflicting network.

Generic routing encapsulation (GRE)

Generic routing encapsulation (GRE) is a protocol that runs over IP and is employed when delivery and payload protocols are compatible but payload addresses are incompatible. For instance, a payload might think it is running on a datalink layer but it is actually running over a transport layer using datagram protocol over IP.

GRE creates a private point-to-point connection and works by encapsulating a payload. GRE is a foundation protocol for other tunnel protocols but the GRE tunnels provide only weak authentication.

Virtual extensible local area network (VXLAN)

The purpose of VXLAN is to provide scalable network isolation. VXLAN is a Layer 2 overlay scheme on a Layer 3 network. It allows an overlay layer-2 network to spread across multiple underlay layer-3 network domains. Each overlay is termed a VXLAN segment. Only VMs within the same VXLAN segment can communicate.

Network namespaces

A namespace is a way of scoping a particular set of identifiers. Using a namespace, you can use the same identifier multiple times in different namespaces. You can also restrict an identifier set visible to particular processes.

For example, Linux provides namespaces for networking and processes, among other things. If a process is running within a process namespace, it can only see and communicate with other processes in the same namespace. So, if a shell in a particular process namespace ran `ps aux`, it would only show the other processes in the same namespace.

Linux network namespaces

In a network namespace, the scoped ‘identifiers’ are network devices; so a given network device, such as `eth0`, exists in a particular namespace. Linux starts up with a default network namespace, so if your operating system does not do anything special, that is where all the network devices will be located. But it is also possible to create further non-default namespaces, and create new devices in those namespaces, or to move an existing device from one namespace to another.

Each network namespace also has its own routing table, and in fact this is the main reason for namespaces to exist. A routing table is keyed by destination IP address, so network namespaces are what you need if you want the same destination IP address to mean different things at different times - which is something that OpenStack Networking requires for its feature of providing overlapping IP addresses in different virtual networks.

Each network namespace also has its own set of iptables (for both IPv4 and IPv6). So, you can apply different security to flows with the same IP addressing in different namespaces, as well as different routing.

Any given Linux process runs in a particular network namespace. By default this is inherited from its parent process, but a process with the right capabilities can switch itself into a different namespace; in practice this is mostly done using the `ip netns exec NETNS COMMAND...` invocation, which starts `COMMAND` running in the namespace named `NETNS`. Suppose such a process sends out a message to IP address A.B.C.D, the effect of the namespace is that A.B.C.D will be looked up in that namespace’s routing table, and that will determine the network device that the message is transmitted through.

Virtual routing and forwarding (VRF)

Virtual routing and forwarding is an IP technology that allows multiple instances of a routing table to coexist on the same router at the same time. It is another name for the network namespace functionality described above.

Network address translation

Network Address Translation (NAT) is a process for modifying the source or destination addresses in the headers of an IP packet while the packet is in transit. In general, the sender and receiver applications are not aware that the IP packets are being manipulated.

NAT is often implemented by routers, and so we will refer to the host performing NAT as a *NAT router*. However, in OpenStack deployments it is typically Linux servers that implement the NAT functionality, not hardware routers. These servers use the `iptables` software package to implement the NAT functionality.

There are multiple variations of NAT, and here we describe three kinds commonly found in OpenStack deployments.

SNAT

In *Source Network Address Translation* (SNAT), the NAT router modifies the IP address of the sender in IP packets. SNAT is commonly used to enable hosts with *private addresses* to communicate with servers on the public Internet.

RFC 1918 reserves the following three subnets as private addresses:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

These IP addresses are not publicly routable, meaning that a host on the public Internet can not send an IP packet to any of these addresses. Private IP addresses are widely used in both residential and corporate environments.

Often, an application running on a host with a private IP address will need to connect to a server on the public Internet. An example is a user who wants to access a public website such as www.openstack.org. If the IP packets reach the web server at www.openstack.org with a private IP address as the source, then the web server cannot send packets back to the sender.

SNAT solves this problem by modifying the source IP address to an IP address that is routable on the public Internet. There are different variations of SNAT; in the form that OpenStack deployments use, a NAT router on the path between the sender and receiver replaces the packet's source IP address with the router's public IP address. The router also modifies the source TCP or UDP port to another value, and the router maintains a record of the sender's true IP address and port, as well as the modified IP address and port.

When the router receives a packet with the matching IP address and port, it translates these back to the private IP address and port, and forwards the packet along.

Because the NAT router modifies ports as well as IP addresses, this form of SNAT is sometimes referred to as *Port Address Translation* (PAT). It is also sometimes referred to as *NAT overload*.

OpenStack uses SNAT to enable applications running inside of instances to connect out to the public Internet.

DNAT

In *Destination Network Address Translation* (DNAT), the NAT router modifies the IP address of the destination in IP packet headers.

OpenStack uses DNAT to route packets from instances to the OpenStack metadata service. Applications running inside of instances access the OpenStack metadata service by making HTTP GET requests to a web server with

IP address 169.254.169.254. In an OpenStack deployment, there is no host with this IP address. Instead, OpenStack uses DNAT to change the destination IP of these packets so they reach the network interface that a metadata service is listening on.

One-to-one NAT

In *one-to-one NAT*, the NAT router maintains a one-to-one mapping between private IP addresses and public IP addresses. OpenStack uses one-to-one NAT to implement floating IP addresses.

OpenStack Networking

OpenStack Networking allows you to create and manage network objects, such as networks, subnets, and ports, which other OpenStack services can use. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

The Networking service, code-named neutron, provides an API that lets you define network connectivity and addressing in the cloud. The Networking service enables operators to leverage different networking technologies to power their cloud networking. The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and [NAT](#) to load balancing, perimeter firewalls, and virtual private networks.

It includes the following components:

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

OpenStack Networking plug-in and agents Plugs and unplugs ports, creates networks or subnets, and provides IP addressing. The chosen plug-in and agents differ depending on the vendor and technologies used in the particular cloud. It is important to mention that only one plug-in can be used at a time.

Messaging queue Accepts and routes RPC requests between agents to complete API operations. Message queue is used in the ML2 plug-in for RPC between the neutron server and neutron agents that run on each hypervisor, in the ML2 mechanism drivers for [Open vSwitch](#) and [Linux bridge](#).

Concepts

To configure rich network topologies, you can create and configure networks and subnets and instruct other OpenStack services like Compute to attach virtual devices to ports on these networks. OpenStack Compute is a prominent consumer of OpenStack Networking to provide connectivity for its instances. In particular, OpenStack Networking supports each project having multiple private networks and enables projects to choose their own IP addressing scheme, even if those IP addresses overlap with those that other projects use. There are two types of network, project and provider networks. It is possible to share any of these types of networks among projects as part of the network creation process.

Provider networks

Provider networks offer layer-2 connectivity to instances with optional support for DHCP and metadata services. These networks connect, or map, to existing layer-2 networks in the data center, typically using VLAN (802.1q)

tagging to identify and separate them.

Provider networks generally offer simplicity, performance, and reliability at the cost of flexibility. By default only administrators can create or update provider networks because they require configuration of physical network infrastructure. It is possible to change the user who is allowed to create or update provider networks with the following parameters of policy.json:

- create_network:provider:physical_network
- update_network:provider:physical_network

Warning: The creation and modification of provider networks enables use of physical network resources, such as VLAN-s. Enable these changes only for trusted tenants.

Also, provider networks only handle layer-2 connectivity for instances, thus lacking support for features such as routers and floating IP addresses.

In many cases, operators who are already familiar with virtual networking architectures that rely on physical network infrastructure for layer-2, layer-3, or other services can seamlessly deploy the OpenStack Networking service. In particular, provider networks appeal to operators looking to migrate from the Compute networking service (nova-network) to the OpenStack Networking service. Over time, operators can build on this minimal architecture to enable more cloud networking features.

In general, the OpenStack Networking software components that handle layer-3 operations impact performance and reliability the most. To improve performance and reliability, provider networks move layer-3 operations to the physical network infrastructure.

In one particular use case, the OpenStack deployment resides in a mixed environment with conventional virtualization and bare-metal hosts that use a sizable physical network infrastructure. Applications that run inside the OpenStack deployment might require direct layer-2 access, typically using VLANs, to applications outside of the deployment.

Routed provider networks

Routed provider networks offer layer-3 connectivity to instances. These networks map to existing layer-3 networks in the data center. More specifically, the network maps to multiple layer-2 segments, each of which is essentially a provider network. Each has a router gateway attached to it which routes traffic between them and externally. The Networking service does not provide the routing.

Routed provider networks offer performance at scale that is difficult to achieve with a plain provider network at the expense of guaranteed layer-2 connectivity.

See [Routed provider networks](#) for more information.

Self-service networks

Self-service networks primarily enable general (non-privileged) projects to manage networks without involving administrators. These networks are entirely virtual and require virtual routers to interact with provider and external networks such as the Internet. Self-service networks also usually provide DHCP and metadata services to instances.

In most cases, self-service networks use overlay protocols such as VXLAN or GRE because they can support many more networks than layer-2 segmentation using VLAN tagging (802.1q). Furthermore, VLANs typically require additional configuration of physical network infrastructure.

IPv4 self-service networks typically use private IP address ranges (RFC1918) and interact with provider networks via source NAT on virtual routers. Floating IP addresses enable access to instances from provider networks via destination NAT on virtual routers. IPv6 self-service networks always use public IP address ranges and interact with provider networks via virtual routers with static routes.

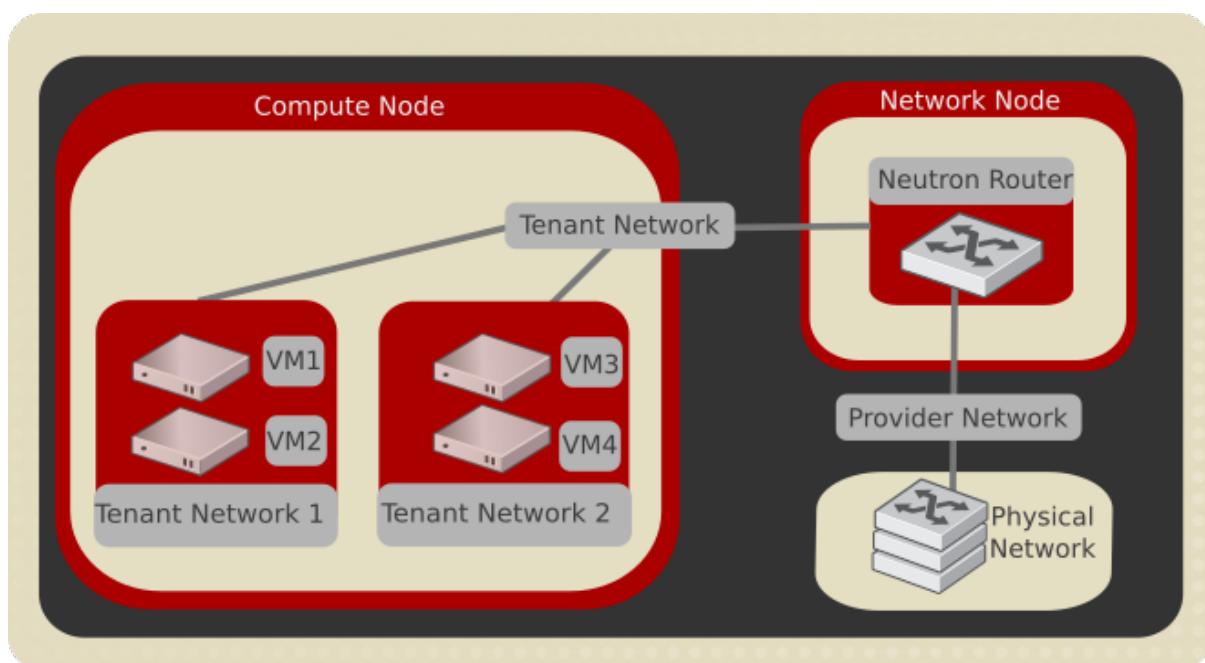
The Networking service implements routers using a layer-3 agent that typically resides at least one network node. Contrary to provider networks that connect instances to the physical network infrastructure at layer-2, self-service networks must traverse a layer-3 agent. Thus, oversubscription or failure of a layer-3 agent or network node can impact a significant quantity of self-service networks and instances using them. Consider implementing one or more high-availability features to increase redundancy and performance of self-service networks.

Users create project networks for connectivity within projects. By default, they are fully isolated and are not shared with other projects. OpenStack Networking supports the following types of network isolation and overlay technologies.

Flat All instances reside on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation takes place.

VLAN Networking allows users to create multiple provider or project networks using VLAN IDs (802.1Q tagged) that correspond to VLANs present in the physical network. This allows instances to communicate with each other across the environment. They can also communicate with dedicated servers, firewalls, load balancers, and other networking infrastructure on the same layer 2 VLAN.

GRE and VXLAN VXLAN and GRE are encapsulation protocols that create overlay networks to activate and control communication between compute instances. A Networking router is required to allow traffic to flow outside of the GRE or VXLAN project network. A router is also required to connect directly-connected project networks with external networks, including the Internet. The router provides the ability to connect to instances directly from an external network using floating IP addresses.



Subnets

A block of IP addresses and associated configuration state. This is also known as the native IPAM (IP Address Management) provided by the networking service for both project and provider networks. Subnets are used to allocate IP addresses when new ports are created on a network.

Subnet pools

End users normally can create subnets with any valid IP addresses without other restrictions. However, in some cases, it is nice for the admin or the project to pre-define a pool of addresses from which to create subnets with automatic allocation.

Using subnet pools constrains what addresses can be used by requiring that every subnet be within the defined pool. It also prevents address reuse or overlap by two subnets from the same pool.

See [Subnet pools](#) for more information.

Ports

A port is a connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. The port also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

Routers

Routers provide virtual layer-3 services such as routing and NAT between self-service and provider networks or among self-service networks belonging to a project. The Networking service uses a layer-3 agent to manage routers via namespaces.

Security groups

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

Each project contains a default security group that allows all egress traffic and denies all ingress traffic. You can change the rules in the default security group. If you launch an instance without specifying a security group, the default security group automatically applies to it. Similarly, if you create a port without specifying a security group, the default security group automatically applies to it.

Note: If you use the metadata service, removing the default egress rules denies access to TCP port 80 on 169.254.169.254, thus preventing instances from retrieving metadata.

Security group rules are stateful. Thus, allowing ingress TCP port 22 for secure shell automatically creates rules that allow return egress traffic and ICMP error messages involving those TCP connections.

By default, all security groups contain a series of basic (sanity) and anti-spoofing rules that perform the following actions:

- Allow egress traffic only if it uses the source MAC and IP addresses of the port for the instance, source MAC and IP combination in `allowed-address-pairs`, or valid MAC address (port or `allowed-address-pairs`) and associated EUI64 link-local IPv6 address.
- Allow egress DHCP discovery and request messages that use the source MAC address of the port for the instance and the unspecified IPv4 address (0.0.0.0).
- Allow ingress DHCP and DHCPv6 responses from the DHCP server on the subnet so instances can acquire IP addresses.
- Deny egress DHCP and DHCPv6 responses to prevent instances from acting as DHCP(v6) servers.
- Allow ingress/egress ICMPv6 MLD, neighbor solicitation, and neighbor discovery messages so instances can discover neighbors and join multicast groups.
- Deny egress ICMPv6 router advertisements to prevent instances from acting as IPv6 routers and forwarding IPv6 traffic for other instances.
- Allow egress ICMPv6 MLD reports (v1 and v2) and neighbor solicitation messages that use the source MAC address of a particular instance and the unspecified IPv6 address (::). Duplicate address detection (DAD) relies on these messages.
- Allow egress non-IP traffic from the MAC address of the port for the instance and any additional MAC addresses in `allowed-address-pairs` on the port for the instance.

Although non-IP traffic, security groups do not implicitly allow all ARP traffic. Separate ARP filtering rules prevent instances from using ARP to intercept traffic for another instance. You cannot disable or remove these rules.

You can disable security groups including basic and anti-spoofing rules by setting the port attribute `port_security_enabled` to `False`.

Extensions

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

DHCP

The optional DHCP service manages IP addresses for instances on provider and self-service networks. The Networking service implements the DHCP service using an agent that manages `qdhcp` namespaces and the `dnsmasq` service.

Metadata

The optional metadata service provides an API for instances to obtain metadata such as SSH keys.

Service and component hierarchy

Server

- Provides API, manages database, etc.

Plug-ins

- Manages agents

Agents

- Provides layer 2/3 connectivity to instances
- Handles physical-virtual network transition
- Handles metadata, etc.

Layer 2 (Ethernet and Switching)

- Linux Bridge
- OVS

Layer 3 (IP and Routing)

- L3
- DHCP

Miscellaneous

- Metadata

Services

Routing services

VPNaaS

The Virtual Private Network-as-a-Service (VPNaaS) is a neutron extension that introduces the VPN feature set.

LBaaS

The Load-Balancer-as-a-Service (LBaaS) API provisions and configures load balancers. The reference implementation is based on the HAProxy software load balancer.

FWaaS

The Firewall-as-a-Service (FWaaS) API is an experimental API that enables early adopters and vendors to test their networking implementations.

Firewall-as-a-Service (FWaaS)

The Firewall-as-a-Service (FWaaS) plug-in applies firewalls to OpenStack objects such as projects, routers, and router ports.

Note: We anticipate this to expand to VM ports in the Ocata cycle.

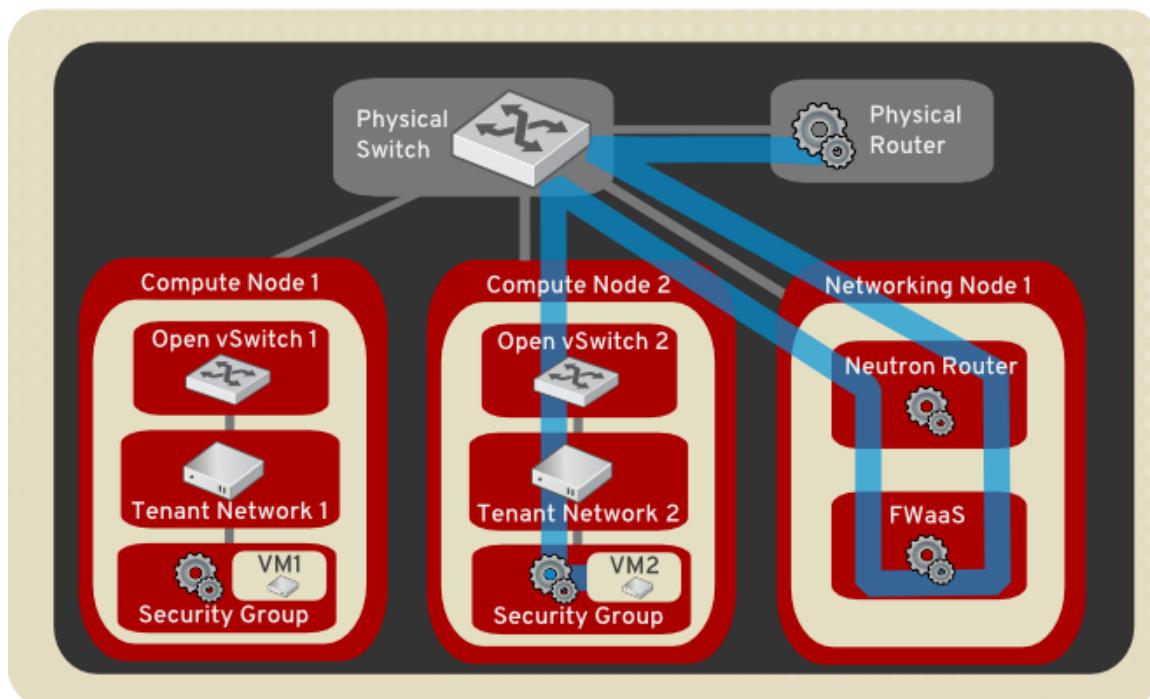
The central concepts with OpenStack firewalls are the notions of a firewall policy and a firewall rule. A policy is an ordered collection of rules. A rule specifies a collection of attributes (such as port ranges, protocol, and IP addresses) that constitute match criteria and an action to take (allow or deny) on matched traffic. A policy can be made public, so it can be shared across projects.

Firewalls are implemented in various ways, depending on the driver used. For example, an iptables driver implements firewalls using iptable rules. An OpenVSwitch driver implements firewall rules using flow entries in flow tables. A Cisco firewall driver manipulates NSX devices.

FWaaS v1

The original FWaaS implementation, v1, provides protection for routers. When a firewall is applied to a router, all internal ports are protected.

The following diagram depicts FWaaS v1 protection. It illustrates the flow of ingress and egress traffic for the VM2 instance:



FWaaS v2

The newer FWaaS implementation, v2, provides a much more granular service. The notion of a firewall has been replaced with firewall group to indicate that a firewall consists of two policies: an ingress policy and an egress policy. A firewall group is applied not at the router level (all ports on a router) but at the port level. Currently, router ports can be specified. For Ocata, VM ports can also be specified.

FWaaS v1 versus v2

The following table compares v1 and v2 features.

Feature	v1	v2
Supports L3 firewalling for routers	YES	NO*
Supports L3 firewalling for router ports	NO	YES
Supports L2 firewalling (VM ports)	NO	NO**
CLI support	YES	YES
Horizon support	YES	NO

* A firewall group can be applied to all ports on a given router in order to effect this.

** This feature is planned for Ocata.

For further information, see [v1 configuration guide](#) or [v2 configuration guide](#).

Configuration

ML2 plug-in

Architecture

The Modular Layer 2 (ML2) neutron plug-in is a framework allowing OpenStack Networking to simultaneously use the variety of layer 2 networking technologies found in complex real-world data centers. The ML2 framework distinguishes between the two kinds of drivers that can be configured:

- Type drivers

Define how an OpenStack network is technically realized. Example: VXLAN

Each available network type is managed by an ML2 type driver. Type drivers maintain any needed type-specific network state. They validate the type specific information for provider networks and are responsible for the allocation of a free segment in project networks.

- Mechanism drivers

Define the mechanism to access an OpenStack network of a certain type. Example: Open vSwitch mechanism driver.

The mechanism driver is responsible for taking the information established by the type driver and ensuring that it is properly applied given the specific networking mechanisms that have been enabled.

Mechanism drivers can utilize L2 agents (via RPC) and/or interact directly with external devices or controllers.

Multiple mechanism and type drivers can be used simultaneously to access different ports of the same virtual network.

ML2 driver support matrix

Table 1: Mechanism drivers and L2 agents

type driver / mech driver	Flat	VLAN	VXLAN	GRE
Open vSwitch	yes	yes	yes	yes
Linux bridge	yes	yes	yes	no
SRIOV	yes	yes	no	no
MacVTap	yes	yes	no	no
L2 population	no	no	yes	yes

Note: L2 population is a special mechanism driver that optimizes BUM (Broadcast, unknown destination address, multicast) traffic in the overlay networks VXLAN and GRE. It needs to be used in conjunction with either the Linux bridge or the Open vSwitch mechanism driver and cannot be used as standalone mechanism driver. For more information, see the *Mechanism drivers* section below.

Configuration

Network type drivers

To enable type drivers in the ML2 plug-in. Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file:

```
[ml2]
type_drivers = flat,vlan,vxlan,gre
```

For more details, see the [Networking configuration options](#) of Configuration Reference.

The following type drivers are available

- Flat
- VLAN
- GRE
- VXLAN

Provider network types

Provider networks provide connectivity like project networks. But only administrative (privileged) users can manage those networks because they interface with the physical network infrastructure. More information about provider networks see [OpenStack Networking](#) or the [OpenStack Administrator Guide](#).

- Flat

The administrator needs to configure a list of physical network names that can be used for provider networks. For more details, see the related section in the Configuration Reference.

- VLAN

The administrator needs to configure a list of physical network names that can be used for provider networks. For more details, see the related section in the Configuration Reference.

- GRE

No additional configuration required.

- VXLAN

The administrator can configure the VXLAN multicast group that should be used.

Note: VXLAN multicast group configuration is not applicable for the Open vSwitch agent.

As of today it is not used in the Linux bridge agent. The Linux bridge agent has its own agent specific configuration option. Please see the following bug for more details: <https://bugs.launchpad.net/neutron/+bug/1523614>

Project network types

Project networks provide connectivity to instances for a particular project. Regular (non-privileged) users can manage project networks within the allocation that an administrator or operator defines for them. More information about project and provider networks see [OpenStack Networking](#) or the [OpenStack Administrator Guide](#).

Project network configurations are made in the `/etc/neutron/plugins/ml2/ml2_conf.ini` configuration file on the neutron server:

- VLAN

The administrator needs to configure the range of VLAN IDs that can be used for project network allocation. For more details, see the related section in the [Configuration Reference](#).

- GRE

The administrator needs to configure the range of tunnel IDs that can be used for project network allocation. For more details, see the related section in the [Configuration Reference](#).

- VXLAN

The administrator needs to configure the range of VXLAN IDs that can be used for project network allocation. For more details, see the related section in the [Configuration Reference](#).

Note: Flat networks for project allocation are not supported. They only can exist as a provider network.

Mechanism drivers

To enable mechanism drivers in the ML2 plug-in, edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file on the neutron server:

```
[ml2]
mechanism_drivers = ovs,l2pop
```

For more details, see the [Configuration Reference](#).

- Linux bridge

No additional configurations required for the mechanism driver. Additional agent configuration is required. For details, see the related *L2 agent* section below.

- Open vSwitch

No additional configurations required for the mechanism driver. Additional agent configuration is required. For details, see the related *L2 agent* section below.

- SRIOV

The administrator needs to define a list PCI hardware that shall be used by OpenStack. For more details, see the related section in the [Configuration Reference](#).

- MacVTap

No additional configurations required for the mechanism driver. Additional agent configuration is required. Please see the related section.

- L2 population

The administrator can configure some optional configuration options. For more details, see the related section in the [Configuration Reference](#).

- Specialized
 - Open source

External open source mechanism drivers exist as well as the neutron integrated reference implementations. Configuration of those drivers is not part of this document. For example:

- * OpenDaylight

- * OpenContrail

- Proprietary (vendor)

External mechanism drivers from various vendors exist as well as the neutron integrated reference implementations.

Configuration of those drivers is not part of this document.

Agents

L2 agent

An L2 agent serves layer 2 (Ethernet) network connectivity to OpenStack resources. It typically runs on each Network Node and on each Compute Node.

- Open vSwitch agent

The Open vSwitch agent configures the Open vSwitch to realize L2 networks for OpenStack resources.

Configuration for the Open vSwitch agent is typically done in the `openvswitch_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

- Linux bridge agent

The Linux bridge agent configures Linux bridges to realize L2 networks for OpenStack resources.

Configuration for the Linux bridge agent is typically done in the `linuxbridge_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

- SRIOV Nic Switch agent

The sriov nic switch agent configures PCI virtual functions to realize L2 networks for OpenStack instances. Network attachments for other resources like routers, DHCP, and so on are not supported.

Configuration for the SRIOV nic switch agent is typically done in the `sriov_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

- MacVTap agent

The MacVTap agent uses kernel MacVTap devices for realizing L2 networks for OpenStack instances. Network attachments for other resources like routers, DHCP, and so on are not supported.

Configuration for the MacVTap agent is typically done in the `macvtap_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

L3 agent

The L3 agent offers advanced layer 3 services, like virtual Routers and Floating IPs. It requires an L2 agent running in parallel.

Configuration for the L3 agent is typically done in the `l3_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

DHCP agent

The DHCP agent is responsible for [DHCP](#) and RADVD (Router Advertisement Daemon) services. It requires a running L2 agent on the same node.

Configuration for the DHCP agent is typically done in the `dhcp_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

Metadata agent

The Metadata agent allows instances to access cloud-init meta data and user data via the network. It requires a running L2 agent on the same node.

Configuration for the Metadata agent is typically done in the `metadata_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

L3 metering agent

The L3 metering agent enables layer3 traffic metering. It requires a running L3 agent on the same node.

Configuration for the L3 metering agent is typically done in the `metering_agent.ini` configuration file. Make sure that on agent start you pass this configuration file as argument.

For a detailed list of configuration options, see the related section in the [Configuration Reference](#).

Security

L2 agents support some important security configurations.

- Security Groups

For more details, see the related section in the [Configuration Reference](#).

- Arp Spoofing Prevention

Configured in the *L2 agent* configuration.

Reference implementations

Overview

In this section, the combination of a mechanism driver and an L2 agent is called ‘reference implementation’. The following table lists these implementations:

Table 2: Mechanism drivers and L2 agents

Mechanism Driver	L2 agent
Open vSwitch	Open vSwitch agent
Linux bridge	Linux bridge agent
SRIOV	SRIOV nic switch agent
MacVTap	MacVTap agent
L2 population	Open vSwitch agent, Linux bridge agent

The following tables shows which reference implementations support which non-L2 neutron agents:

Table 3: Reference implementations and other agents

Reference Implementation	L3 agent	DHCP agent	Metadata agent	L3 Metering agent
Open vSwitch & Open vSwitch agent	yes	yes	yes	yes
Linux bridge & Linux bridge agent	yes	yes	yes	yes
SRIOV & SRIOV nic switch agent	no	no	no	no
MacVTap & MacVTap agent	no	no	no	no

Note: L2 population is not listed here, as it is not a standalone mechanism. If other agents are supported depends on the conjunctive mechanism driver that is used for binding a port.

More information about L2 population see the [OpenStack Manuals](#).

Buying guide

This guide characterizes the L2 reference implementations that currently exist.

- Open vSwitch mechanism and Open vSwitch agent

Can be used for instance network attachments as well as for attachments of other network resources like routers, DHCP, and so on.

- Linux bridge mechanism and Linux bridge agent

Can be used for instance network attachments as well as for attachments of other network resources like routers, DHCP, and so on.

- SRIOV mechanism driver and SRIOV NIC switch agent

Can only be used for instance network attachments (device_owner = compute).

Is deployed besides an other mechanism driver and L2 agent such as OVS or Linux bridge. It offers instances direct access to the network adapter through a PCI Virtual Function (VF). This gives an instance direct access to hardware capabilities and high performance networking.

The cloud consumer can decide via the neutron APIs VNIC_TYPE attribute, if an instance gets a normal OVS port or an SRIOV port.

Due to direct connection, some features are not available when using SRIOV. For example, DVR, security groups, migration.

For more information see the [SR-IOV](#).

- MacVTap mechanism driver and MacVTap agent

Can only be used for instance network attachments (device_owner = compute) and not for attachment of other resources like routers, DHCP, and so on.

It is positioned as alternative to Open vSwitch or Linux bridge support on the compute node for internal deployments.

MacVTap offers a direct connection with very little overhead between instances and down to the adapter. You can use MacVTap agent on the compute node when you require a network connection that is performance critical. It does not require specific hardware (like with SRIOV).

Due to the direct connection, some features are not available when using it on the compute node. For example, DVR, security groups and arp-spoofing protection.

Address scopes

Address scopes build from subnet pools. While subnet pools provide a mechanism for controlling the allocation of addresses to subnets, address scopes show where addresses can be routed between networks, preventing the use of overlapping addresses in any two subnets. Because all addresses allocated in the address scope do not overlap, neutron routers do not NAT between your projects' network and your external network. As long as the addresses within an address scope match, the Networking service performs simple routing between networks.

Accessing address scopes

Anyone with access to the Networking service can create their own address scopes. However, network administrators can create shared address scopes, allowing other projects to create networks within that address

scope.

Access to addresses in a scope are managed through subnet pools. Subnet pools can either be created in an address scope, or updated to belong to an address scope.

With subnet pools, all addresses in use within the address scope are unique from the point of view of the address scope owner. Therefore, add more than one subnet pool to an address scope if the pools have different owners, allowing for delegation of parts of the address scope. Delegation prevents address overlap across the whole scope. Otherwise, you receive an error if two pools have the same address ranges.

Each router interface is associated with an address scope by looking at subnets connected to the network. When a router connects to an external network with matching address scopes, network traffic routes between without Network address translation (NAT). The router marks all traffic connections originating from each interface with its corresponding address scope. If traffic leaves an interface in the wrong scope, the router blocks the traffic.

Backwards compatibility

Networks created before the Mitaka release do not contain explicitly named address scopes, unless the network contains subnets from a subnet pool that belongs to a created or updated address scope. The Networking service preserves backwards compatibility with pre-Mitaka networks through special address scope properties so that these networks can perform advanced routing:

1. Unlimited address overlap is allowed.
2. Neutron routers, by default, will NAT traffic from internal networks to external networks.
3. Pre-Mitaka address scopes are not visible through the API. You cannot list address scopes or show details. Scopes exist implicitly as a catch-all for addresses that are not explicitly scoped.

Create shared address scopes as an administrative user

This section shows how to set up shared address scopes to allow simple routing for project networks with the same subnet pools.

Note: Irrelevant fields have been trimmed from the output of these commands for brevity.

1. Create IPv6 and IPv4 address scopes:

```
$ openstack address scope create --share --ip-version 6 address-scope-ip6

+-----+-----+
| Field      | Value
+-----+-----+
| headers    |
| id         | 28424dfc-9abd-481b-afa3-1da97a8fead7
| ip_version | 6
| name       | address-scope-ip6
| project_id | 098429d072d34d3596c88b7dbf7e91b6
| shared     | True
+-----+-----+
```

```
$ openstack address scope create --share --ip-version 4 address-scope-ip4

+-----+-----+
| Field | Value |
+-----+-----+
| headers |          |
| id      | 3193bd62-11b5-44dc-acf8-53180f21e9f2 |
| ip_version | 4           |
| name     | address-scope-ip4 |
| project_id | 098429d072d34d3596c88b7dbf7e91b6 |
| shared    | True          |
+-----+-----+
```

2. Create subnet pools specifying the name (or UUID) of the address scope that the subnet pool belongs to. If you have existing subnet pools, use the **openstack subnet pool set** command to put them in a new address scope:

```
$ openstack subnet pool create --address-scope address-scope-ip6 \
--share --pool-prefix 2001:db8:a583::/48 --default-prefix-length 64 \
subnet-pool-ip6

+-----+-----+
| Field | Value |
+-----+-----+
| address_scope_id | 28424dfc-9abd-481b-afa3-1da97a8fead7 |
| created_at       | 2016-12-13T22:53:30Z |
| default_prefixlen | 64           |
| default_quota    | None          |
| description      |              |
| id               | a59ff52b-0367-41ff-9781-6318b927dd0e |
| ip_version       | 6             |
| is_default       | False         |
| max_prefixlen    | 128          |
| min_prefixlen    | 64           |
| name             | subnet-pool-ip6 |
| prefixes         | 2001:db8:a583::/48 |
| project_id       | 098429d072d34d3596c88b7dbf7e91b6 |
| revision_number | 1             |
| shared           | True          |
| updated_at       | 2016-12-13T22:53:30Z |
+-----+-----+
```

```
$ openstack subnet pool create --address-scope address-scope-ip4 \
--share --pool-prefix 203.0.113.0/24 --default-prefix-length 26 \
subnet-pool-ip4

+-----+-----+
| Field | Value |
+-----+-----+
| address_scope_id | 3193bd62-11b5-44dc-acf8-53180f21e9f2 |
| created_at       | 2016-12-13T22:55:09Z |
| default_prefixlen | 26           |
| default_quota    | None          |
| description      |              |
| id               | d02af70b-d622-426f-8e60-ed9df2a8301f |
| ip_version       | 4             |
| is_default       | False         |
| max_prefixlen    | 32           |
```

min_prefixlen	8	
name	subnet-pool-ip4	
prefixes	203.0.113.0/24	
project_id	098429d072d34d3596c88b7dbf7e91b6	
revision_number	1	
shared	True	
updated_at	2016-12-13T22:55:09Z	
+-----+-----+-----+		

3. Make sure that subnets on an external network are created from the subnet pools created above:

\$ openstack subnet show ipv6-public-subnet		
Field	Value	
allocation_pools	2001:db8:a583::2-2001:db8:a583:0:ffff:ff	
	ff:ffff:ffff	
cidr	2001:db8:a583::/64	
created_at	2016-12-10T21:36:04Z	
description		
dns_nameservers		
enable_dhcp	False	
gateway_ip	2001:db8:a583::1	
host_routes		
id	b333bf5a-758c-4b3f-97ec-5f12d9bfceb7	
ip_version	6	
ipv6_address_mode	None	
ipv6_ra_mode	None	
name	ipv6-public-subnet	
network_id	05a8d31e-330b-4d96-a3fa-884b04abfa4c	
project_id	098429d072d34d3596c88b7dbf7e91b6	
revision_number	2	
segment_id	None	
service_types		
subnetpool_id	a59ff52b-0367-41ff-9781-6318b927dd0e	
updated_at	2016-12-10T21:36:04Z	
+-----+-----+-----+		

\$ openstack subnet show public-subnet		
Field	Value	
allocation_pools	203.0.113.2-203.0.113.62	
cidr	203.0.113.0/26	
created_at	2016-12-10T21:35:52Z	
description		
dns_nameservers		
enable_dhcp	False	
gateway_ip	203.0.113.1	
host_routes		
id	7fd48240-3acc-4724-bc82-16c62857edec	
ip_version	4	
ipv6_address_mode	None	
ipv6_ra_mode	None	
name	public-subnet	
network_id	05a8d31e-330b-4d96-a3fa-884b04abfa4c	
project_id	098429d072d34d3596c88b7dbf7e91b6	

revision_number	2
segment_id	None
service_types	
subnetpool_id	d02af70b-d622-426f-8e60-ed9df2a8301f
updated_at	2016-12-10T21:35:52Z
+-----+-----+	

Routing with address scopes for non-privileged users

This section shows how non-privileged users can use address scopes to route straight to an external network without NAT.

1. Create a couple of networks to host subnets:

\$ openstack network create network1	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2016-12-13T23:21:01Z
description	
headers	
id	1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0
ipv4_address_scope	None
ipv6_address_scope	None
mtu	1450
name	network1
port_security_enabled	True
project_id	098429d072d34d3596c88b7dbf7e91b6
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	94
revision_number	3
router:external	Internal
shared	False
status	ACTIVE
subnets	
tags	[]
updated_at	2016-12-13T23:21:01Z
+-----+-----+	

\$ openstack network create network2	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2016-12-13T23:21:45Z
description	
headers	
id	6c583603-c097-4141-9c5c-288b0e49c59f
ipv4_address_scope	None

ipv6_address_scope	None
mtu	1450
name	network2
port_security_enabled	True
project_id	098429d072d34d3596c88b7dbf7e91b6
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	81
revision_number	3
router:external	Internal
shared	False
status	ACTIVE
subnets	
tags	[]
updated_at	2016-12-13T23:21:45Z

2. Create a subnet not associated with a subnet pool or an address scope:

\$ openstack subnet create --network network1 --subnet-range \
198.51.100.0/26 subnet-ip4-1
+-----+-----+
Field Value
+-----+-----+
allocation_pools 198.51.100.2-198.51.100.62
cidr 198.51.100.0/26
created_at 2016-12-13T23:24:16Z
description
dns_nameservers
enable_dhcp True
gateway_ip 198.51.100.1
headers
host_routes
id 66874039-d31b-4a27-85d7-14c89341bbb7
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name subnet-ip4-1
network_id 1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0
project_id 098429d072d34d3596c88b7dbf7e91b6
revision_number 2
service_types
subnetpool_id None
updated_at 2016-12-13T23:24:16Z
+-----+-----+

\$ openstack subnet create --network network1 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --ip-version 6 --subnet-range \
2001:db8:80d2:c4d3::/64 subnet-ip6-1
+-----+-----+
Field Value
+-----+-----+
allocation_pools 2001:db8:80d2:c4d3::2-2001:db8:80d2:c4d
3:ffff:ffff:ffff:ffff
cidr 2001:db8:80d2:c4d3::/64
created_at 2016-12-13T23:28:28Z
description

dns_nameservers	
enable_dhcp	True
gateway_ip	2001:db8:80d2:c4d3::1
headers	
host_routes	
id	a7551b23-2271-4a88-9c41-c84b048e0722
ip_version	6
ipv6_address_mode	slaac
ipv6_ra_mode	slaac
name	subnet-ip6-1
network_id	1bcf3fe9-a0cb-4d88-a067-a4d7f8e635f0
project_id	098429d072d34d3596c88b7dbf7e91b6
revision_number	2
service_types	
subnetpool_id	None
updated_at	2016-12-13T23:28:28Z

3. Create a subnet using a subnet pool associated with an address scope from an external network:

\$ openstack subnet create --subnet-pool subnet-pool-ip4 \
--network network2 subnet-ip4-2
+-----+-----+
Field Value
+-----+-----+
allocation_pools 203.0.113.2-203.0.113.62
cidr 203.0.113.0/26
created_at 2016-12-13T23:32:12Z
description
dns_nameservers
enable_dhcp True
gateway_ip 203.0.113.1
headers
host_routes
id 12be8e8f-5871-4091-9e9e-4e0651b9677e
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name subnet-ip4-2
network_id 6c583603-c097-4141-9c5c-288b0e49c59f
project_id 098429d072d34d3596c88b7dbf7e91b6
revision_number 2
service_types
subnetpool_id d02af70b-d622-426f-8e60-ed9df2a8301f
updated_at 2016-12-13T23:32:12Z
+-----+-----+

\$ openstack subnet create --ip-version 6 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --subnet-pool subnet-pool-ip6 \
--network network2 subnet-ip6-2
+-----+-----+
Field Value
+-----+-----+
allocation_pools 2001:db8:a583::2-2001:db8:a583:0:fff
f:ffff:ffff:ffff
cidr 2001:db8:a583::/64
created_at 2016-12-13T23:31:17Z

description	
dns_nameservers	
enable_dhcp	True
gateway_ip	2001:db8:a583::1
headers	
host_routes	
id	b599c2be-e3cd-449c-ba39-3cfcc744c4be
ip_version	6
ipv6_address_mode	slaac
ipv6_ra_mode	slaac
name	subnet-ip6-2
network_id	6c583603-c097-4141-9c5c-288b0e49c59f
project_id	098429d072d34d3596c88b7dbf7e91b6
revision_number	2
service_types	
subnetpool_id	a59ff52b-0367-41ff-9781-6318b927dd0e
updated_at	2016-12-13T23:31:17Z

By creating subnets from scoped subnet pools, the network is associated with the address scope.

\$ openstack network show network2	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	nova
created_at	2016-12-13T23:21:45Z
description	
id	6c583603-c097-4141-9c5c-
	288b0e49c59f
ipv4_address_scope	3193bd62-11b5-44dc-
	acf8-53180f21e9f2
ipv6_address_scope	28424dfc-9abd-481b-
	afa3-1da97a8fead7
mtu	1450
name	network2
port_security_enabled	True
project_id	098429d072d34d3596c88b7dbf7e
	91b6
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	81
revision_number	10
router:external	Internal
shared	False
status	ACTIVE
subnets	12be8e8f-5871-4091-9e9e-
	4e0651b9677e, b599c2be-e3cd-
	449c-ba39-3cfcc744c4be
tags	[]
updated_at	2016-12-13T23:32:12Z

4. Connect a router to each of the project subnets that have been created, for example, using a router called router1:

```
$ openstack router add subnet router1 subnet-ip4-1
$ openstack router add subnet router1 subnet-ip4-2
$ openstack router add subnet router1 subnet-ip6-1
$ openstack router add subnet router1 subnet-ip6-2
```

Checking connectivity

This example shows how to check the connectivity between networks with address scopes.

1. Launch two instances, `instance1` on `network1` and `instance2` on `network2`. Associate a floating IP address to both instances.
2. Adjust security groups to allow pings and SSH (both IPv4 and IPv6):

```
$ openstack server list
+-----+-----+
| ID      | Name     | Networks
|         |          | Image Name |
+-----+-----+
| 97e49c8e-... | instance1 | network1=2001:db8:80d2:c4d3:f816:3eff:fe52:b69f, 198.51.
| 100.3, 203.0.113.3| cirros    |
| ceba9638-... | instance2 | network2=203.0.113.3, 2001:db8:a583:0:f816:3eff:fe42:1eeb, 203.0.113.4 | centos    |
+-----+-----+
```

Regardless of address scopes, the floating IPs can be pinged from the external network:

```
$ ping -c 1 203.0.113.3
1 packets transmitted, 1 received, 0% packet loss, time 0ms
$ ping -c 1 203.0.113.4
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

You can now ping `instance2` directly because `instance2` shares the same address scope as the external network:

Note: BGP routing can be used to automatically set up a static route for your instances.

```
# ip route add 203.0.113.0/26 via 203.0.113.2
$ ping -c 1 203.0.113.3
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
# ip route add 2001:db8:a583::/64 via 2001:db8::1
$ ping6 -c 1 2001:db8:a583:0:f816:3eff:fe42:1eeb
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

You cannot ping `instance1` directly because the address scopes do not match:

```
# ip route add 198.51.100.0/26 via 203.0.113.2
$ ping -c 1 198.51.100.3
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
# ip route add 2001:db8:80d2:c4d3::/64 via 2001:db8::1
$ ping6 -c 1 2001:db8:80d2:c4d3:f816:3eff:fe52:b69f
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

If the address scopes match between networks then pings and other traffic route directly through. If the scopes do not match between networks, the router either drops the traffic or applies NAT to cross scope boundaries.

Automatic allocation of network topologies

The auto-allocation feature introduced in Mitaka simplifies the procedure of setting up an external connectivity for end-users, and is also known as **Get Me A Network**.

Previously, a user had to configure a range of networking resources to boot a server and get access to the Internet. For example, the following steps are required:

- Create a network
- Create a subnet
- Create a router
- Uplink the router on an external network
- Downlink the router on the previously created subnet

These steps need to be performed on each logical segment that a VM needs to be connected to, and may require networking knowledge the user might not have.

This feature is designed to automate the basic networking provisioning for projects. The steps to provision a basic network are run during instance boot, making the networking setup transparent.

To make this possible, provide a default external network and default subnetpools (one for IPv4, or one for IPv6, or one of each) so that the platform can choose what to do in lieu of input. Once these are in place, users can boot their VMs without specifying any networking details. The Compute service will then use this feature automatically to wire user VMs.

Enabling the deployment for auto-allocation

To use this feature, the neutron service must have the following extensions enabled:

- auto-allocated-topology
- subnet_allocation
- external-net
- router

Before the end-user can use the auto-allocation feature, the operator must create the resources that will be used for the auto-allocated network topology creation. To perform this task, proceed with the following steps:

1. Set up a default external network

Setting up an external network is described in [OpenStack Administrator Guide](#). Assuming the external network to be used for the auto-allocation feature is named public, make it the default external network with the following command:

```
$ neutron net-update public --is-default=True
```

Note: The flag `--default` (and `--no-default` flag) is only effective with external networks and has no effects on regular (or internal) networks.

2. Create default subnetpools

The auto-allocation feature requires at least one default subnetpool. One for IPv4, or one for IPv6, or one of each.

```
$ openstack subnet pool create --share --default \
--pool-prefix 192.0.2.0/24 --default-prefix-length 26 \
shared-default

+-----+-----+
| Field | Value |
+-----+-----+
| address_scope_id | None |
| created_at | 2017-01-12T15:10:34Z |
| default_prefixlen | 26 |
| default_quota | None |
| description | |
| headers | |
| id | b41b7b9c-de57-4c19-b1c5-731985bceb7f |
| ip_version | 4 |
| is_default | True |
| max_prefixlen | 32 |
| min_prefixlen | 8 |
| name | shared-default |
| prefixes | 192.0.2.0/24 |
| project_id | 86acdbd1d72745fd8e8320edd7543400 |
| revision_number | 1 |
| shared | True |
| updated_at | 2017-01-12T15:10:34Z |
+-----+-----+



$ openstack subnet pool create --share --default \
--pool-prefix 2001:db8:8000::/48 --default-prefix-length 64 \
default-v6

+-----+-----+
| Field | Value |
+-----+-----+
| address_scope_id | None |
| created_at | 2017-01-12T15:14:35Z |
| default_prefixlen | 64 |
| default_quota | None |
| description | |
| headers | |
| id | 6f387016-17f0-4564-96ad-e34775b6ea14 |
| ip_version | 6 |
| is_default | True |
| max_prefixlen | 128 |
| min_prefixlen | 64 |
| name | default-v6 |
```

prefixes	2001:db8:8000::/48	
project_id	86acdbd1d72745fd8e8320edd7543400	
revision_number	1	
shared	True	
updated_at	2017-01-12T15:14:35Z	

Get Me A Network

In a deployment where the operator has set up the resources as described above, validate that users can get their auto-allocated network topology as follows:

```
$ neutron auto-allocated-topology-show
+-----+-----+
| Field | Value |
+-----+-----+
| id | 8b835bfb-cae2-4acc-b53f-c16bb5f9a7d0 |
| tenant_id | 3a4e311bcb3545b9b7ad326f93194f8c |
+-----+-----+
```

Operators (and users with admin role) can get the auto-allocated topology for a project by specifying the project ID:

```
$ neutron auto-allocated-topology-show 3a4e311bcb3545b9b7ad326f93194f8c
+-----+-----+
| Field | Value |
+-----+-----+
| id | 8b835bfb-cae2-4acc-b53f-c16bb5f9a7d0 |
| tenant_id | 3a4e311bcb3545b9b7ad326f93194f8c |
+-----+-----+
```

The ID returned by this command is a network which can be used for booting a VM.

```
$ openstack server create --flavor m1.small --image \
cirros-0.3.5-x86_64-uec --nic \
net-id=8b835bfb-cae2-4acc-b53f-c16bb5f9a7d0 vm1
```

The auto-allocated topology for a user never changes. In practice, when a user boots a server omitting the --nic option, and not have any neutron network available, nova will invoke the API behind auto-allocated-topology-show, fetch the network UUID, and pass it on during the boot process.

Validating the requirements for auto-allocation

To validate that the required resources are correctly set up for auto-allocation, without actually provisioning any resource, use the --dry-run option:

```
$ neutron auto-allocated-topology-show --dry-run
Deployment error: No default router:external network.

$ neutron net-update public --is-default=True

$ neutron auto-allocated-topology-show --dry-run
Deployment error: No default subnetpools defined.
```

```
$ neutron subnetpool-update shared-default --is-default=True  
  
$ neutron auto-allocated-topology-show --dry-run  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| dry-run | pass |  
+-----+-----+
```

The validation option behaves identically for all users. However, it is considered primarily an admin or service utility since it is the operator who must set up the requirements.

Project resources created by auto-allocation

The auto-allocation feature creates one network topology in every project where it is used. The auto-allocated network topology for a project contains the following resources:

Resource	Name
network	auto_allocated_network
subnet (IPv4)	auto_allocated_subnet_v4
subnet (IPv6)	auto_allocated_subnet_v6
router	auto_allocated_router

Compatibility notes

Nova uses the auto-allocated-topology feature with API micro version 2.37 or later. This is because, unlike the neutron feature which was implemented in the Mitaka release, the integration for nova was completed during the Newton release cycle. Note that the CLI option `--nic` can be omitted regardless of the microversion used as long as there is no more than one network available to the project, in which case nova fails with a 400 error because it does not know which network to use. Furthermore, nova does not start using the feature, regardless of whether or not a user requests micro version 2.37 or later, unless all of the nova-compute services are running Newton-level code.

Availability zones

An availability zone groups network nodes that run services like DHCP, L3, FW, and others. It is defined as an agent's attribute on the network node. This allows users to associate an availability zone with their resources so that the resources get high availability.

Use case

An availability zone is used to make network resources highly available. The operators group the nodes that are attached to different power sources under separate availability zones and configure scheduling for resources with high availability so that they are scheduled on different availability zones.

Required extensions

The core plug-in must support the `availability_zone` extension. The core plug-in also must support the `network_availability_zone` extension to schedule a network according to availability zones. The `M12Plugin` supports it. The router service plug-in must support the `router_availability_zone` extension to schedule a router according to the availability zones. The `L3RouterPlugin` supports it.

```
$ openstack extension list --network -c Alias -c Name
+-----+-----+
| Name           | Alias          |
+-----+-----+
...
| Network Availability Zone | network_availability_zone |
...
| Availability Zone        | availability_zone   |
...
| Router Availability Zone | router_availability_zone |
...
+-----+-----+
```

Availability zone of agents

The `availability_zone` attribute can be defined in `dhcp-agent` and `l3-agent`. To define an availability zone for each agent, set the value into [AGENT] section of `/etc/neutron/dhcp_agent.ini` or `/etc/neutron/l3_agent.ini`:

```
[AGENT]
availability_zone = zone-1
```

To confirm the agent's availability zone:

```
$ openstack network agent show 116cc128-4398-49af-a4ed-3e95494cd5fc
+-----+-----+
| Field           | Value          |
+-----+-----+
| admin_state_up  | UP             |
| agent_type      | DHCP agent     |
| alive            | True           |
| availability_zone | zone-1         |
| binary           | neutron-dhcp-agent |
| configurations   | dhcp_driver='neutron.agent.linux.dhcp.Dnsmasq', |
|                   | dhcp_lease_duration='86400', |
|                   | log_agent_heartbeats='False', networks='2', |
|                   | notifies_port_ready='True', ports='6', subnets='4' |
| created_at       | 2016-12-14 00:25:54 |
| description      | None           |
| heartbeat_timestamp | 2016-12-14 06:20:24 |
| host              | ankur-desktop |
| id                | 116cc128-4398-49af-a4ed-3e95494cd5fc |
| started_at        | 2016-12-14 00:25:54 |
| topic             | dhcp_agent    |
+-----+-----+

$ openstack network agent show 9632309a-2aa4-4304-8603-c4de02c4a55f
+-----+-----+
```

Field	Value
admin_state_up	UP
agent_type	L3 agent
alive	True
availability_zone	zone-1
binary	neutron-l3-agent
configurations	agent_mode='legacy', ex_gw_ports='2', external_network_bridge='', floating_ips='0', gateway_external_network_id='', handle_internal_only_routers='True', interface_driver='openvswitch', interfaces='4', log_agent_heartbeats='False', routers='2'
created_at	2016-12-14 00:25:58
description	None
heartbeat_timestamp	2016-12-14 06:20:28
host	ankur-desktop
id	9632309a-2aa4-4304-8603-c4de02c4a55f
started_at	2016-12-14 00:25:58
topic	l3_agent

Availability zone related attributes

The following attributes are added into network and router:

Attribute name	Access	Re-required	Input type	Description
availability_zone_hints	RW(POST only)	No	list of string	availability zone candidates for the resource
availability_zones	RO	N/A	list of string	availability zones for the resource

Use `availability_zone_hints` to specify the zone in which the resource is hosted:

\$ openstack network create --availability-zone-hint zone-1 \\\n--availability-zone-hint zone-2 net1	+-----+-----+
Field	Value
+-----+-----+	+-----+
admin_state_up	UP
availability_zone_hints	zone-1
	zone-2
availability_zones	
created_at	2016-12-14T06:23:36Z
description	
headers	
id	ad88e059-e7fa-4cf7-8857-6731a2a3a554
ipv4_address_scope	None
ipv6_address_scope	None
mtu	1450
name	net1
port_security_enabled	True
project_id	cfd1889ac7d64ad891d4f20aef9f8d7c
provider:network_type	vxlan

provider:physical_network	None	
provider:segmentation_id	77	
revision_number	3	
router:external	Internal	
shared	False	
status	ACTIVE	
subnets		
tags	[]	
updated_at	2016-12-14T06:23:37Z	
+	-----+-----+	

\$ openstack router create --ha --availability-zone-hint zone-1 \		
--availability-zone-hint zone-2 router1		
+	-----+-----+	
Field	Value	
+	-----+-----+	
admin_state_up	UP	
availability_zone_hints	zone-1	
	zone-2	
availability_zones		
created_at	2016-12-14T06:25:40Z	
description		
distributed	False	
external_gateway_info	null	
flavor_id	None	
ha	False	
headers		
id	ced10262-6cfe-47c1-8847-cd64276a868c	
name	router1	
project_id	cfd1889ac7d64ad891d4f20aef9f8d7c	
revision_number	3	
routes		
status	ACTIVE	
updated_at	2016-12-14T06:25:40Z	
+	-----+-----+	

Availability zone is selected from `default_availability_zones` in `/etc/neutron/neutron.conf` if a resource is created without `availability_zone_hints`:

```
default_availability_zones = zone-1,zone-2
```

To confirm the availability zone defined by the system:

\$ openstack availability zone list		
+	-----+-----+	
Zone Name	Zone Status	
+	-----+-----+	
zone-1	available	
zone-2	available	
zone-1	available	
zone-2	available	
+	-----+-----+	

Look at the `availability_zones` attribute of each resource to confirm in which zone the resource is hosted:

```
$ openstack network show net1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP
| availability_zone_hints | zone-1
| | zone-2
| availability_zones | zone-1
| | zone-2
| created_at | 2016-12-14T06:23:36Z
| description |
| headers |
| id | ad88e059-e7fa-4cf7-8857-6731a2a3a554
| ipv4_address_scope | None
| ipv6_address_scope | None
| mtu | 1450
| name | net1
| port_security_enabled | True
| project_id | cfd1889ac7d64ad891d4f20aef9f8d7c
| provider:network_type | vxlan
| provider:physical_network | None
| provider:segmentation_id | 77
| revision_number | 3
| router:external | Internal
| shared | False
| status | ACTIVE
| subnets |
| tags | []
| updated_at | 2016-12-14T06:23:37Z
+-----+-----+
```

```
$ openstack router show router1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP
| availability_zone_hints | zone-1
| | zone-2
| availability_zones | zone-1
| | zone-2
| created_at | 2016-12-14T06:25:40Z
| description |
| distributed | False
| external_gateway_info | null
| flavor_id | None
| ha | False
| headers |
| id | ced10262-6cfe-47c1-8847-cd64276a868c
| name | router1
| project_id | cfd1889ac7d64ad891d4f20aef9f8d7c
| revision_number | 3
| routes |
| status | ACTIVE
| updated_at | 2016-12-14T06:25:40Z
+-----+-----+
```

Note: The `availability_zones` attribute does not have a value until the resource is scheduled. Once the Networking service schedules the resource to zones according to `availability_zone_hints`, `availability_zones` shows in which zone the resource is hosted practically. The `availability_zones` may not match `availability_zone_hints`. For example, even if you specify a zone with `availability_zone_hints`, all agents of the zone may be dead before the resource is scheduled. In general, they should match, unless there are failures or there is no capacity left in the zone requested.

Availability zone aware scheduler

Network scheduler

Set `AZAwareWeightScheduler` to `network_scheduler_driver` in `/etc/neutron/neutron.conf` so that the Networking service schedules a network according to the availability zone:

```
network_scheduler_driver = neutron.scheduler.dhcp_agent_scheduler.AZAwareWeightScheduler
dhcp_load_type = networks
```

The Networking service schedules a network to one of the agents within the selected zone as with `WeightScheduler`. In this case, scheduler refers to `dhcp_load_type` as well.

Router scheduler

Set `AZLeastRoutersScheduler` to `router_scheduler_driver` in file `/etc/neutron/neutron.conf` so that the Networking service schedules a router according to the availability zone:

```
router_scheduler_driver = neutron.scheduler.l3_agent_scheduler.AZLeastRoutersScheduler
```

The Networking service schedules a router to one of the agents within the selected zone as with `LeastRouterScheduler`.

Achieving high availability with availability zone

Although, the Networking service provides high availability for routers and high availability and fault tolerance for networks' DHCP services, availability zones provide an extra layer of protection by segmenting a Networking service deployment in isolated failure domains. By deploying HA nodes across different availability zones, it is guaranteed that network services remain available in face of zone-wide failures that affect the deployment.

This section explains how to get high availability with the availability zone for L3 and DHCP. You should naturally set above configuration options for the availability zone.

L3 high availability

Set the following configuration options in file `/etc/neutron/neutron.conf` so that you get L3 high availability.

```
l3_ha = True
max_l3_agents_per_router = 3
```

HA routers are created on availability zones you selected when creating the router.

DHCP high availability

Set the following configuration options in file `/etc/neutron/neutron.conf` so that you get DHCP high availability.

```
dhcp_agents_per_network = 2
```

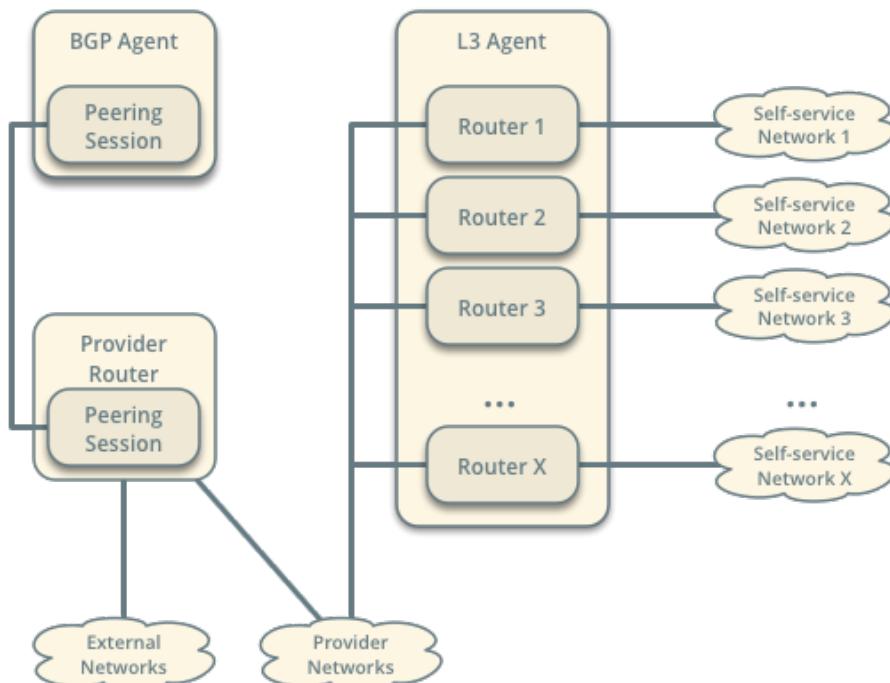
DHCP services are created on availability zones you selected when creating the network.

BGP dynamic routing

BGP dynamic routing enables advertisement of self-service (private) network prefixes to physical network devices that support BGP such as routers, thus removing the conventional dependency on static routes. The feature relies on [address scopes](#) and requires knowledge of their operation for proper deployment.

BGP dynamic routing consists of a service plug-in and an agent. The service plug-in implements the Networking service extension and the agent manages BGP peering sessions. A cloud administrator creates and configures a BGP speaker using the CLI or API and manually schedules it to one or more hosts running the agent. Agents can reside on hosts with or without other Networking service agents. Prefix advertisement depends on the binding of external networks to a BGP speaker and the address scope of external and internal IP address ranges or subnets.

BGP Dynamic Routing Overview



Note: Although self-service networks generally use private IP address ranges (RFC1918) for IPv4 subnets,

BGP dynamic routing can advertise any IPv4 address ranges.

Example configuration

The example configuration involves the following components:

- One BGP agent.
- One address scope containing IP address range 203.0.113.0/24 for provider networks, and IP address ranges 10.0.1.0/24 and 10.0.2.0/24 for self-service networks.
- One provider network using IP address range 203.0.113.0/24.
- Three self-service networks.
 - Self-service networks 1 and 2 use IP address ranges inside of the address scope.
 - Self-service network 3 uses a unique IP address range 10.0.3.0/24 to demonstrate that the BGP speaker does not advertise prefixes outside of address scopes.
- Three routers. Each router connects one self-service network to the provider network.
 - Router 1 contains IP addresses 203.0.113.11 and 10.0.1.1.
 - Router 2 contains IP addresses 203.0.113.12 and 10.0.2.1.
 - Router 3 contains IP addresses 203.0.113.13 and 10.0.3.1.

Note: The example configuration assumes sufficient knowledge about the Networking service, routing, and BGP. For basic deployment of the Networking service, consult one of the [Deployment examples](#). For more information on BGP, see [RFC 4271](#).

Controller node

- In the `neutron.conf` file, enable the conventional layer-3 and BGP dynamic routing service plug-ins:

```
[DEFAULT]
service_plugins = neutron_dynamic_routing.services.bgp.bgp_plugin.BgpPlugin,neutron.
    ↪services.l3_router.l3_router_plugin.L3RouterPlugin
```

Agent nodes

- In the `bgp_dagent.ini` file:
 - Configure the driver.

```
[BGP]
bgp_speaker_driver = neutron_dynamic_routing.services.bgp.agent.driver.ryu.driver.
    ↪RyuBgpDriver
```

Note: The agent currently only supports the Ryu BGP driver.

- Configure the router ID.

```
[BGP]
```

```
bgp_router_id = ROUTER_ID
```

Replace ROUTER_ID with a suitable unique 32-bit number, typically an IPv4 address on the host running the agent. For example, 192.0.2.2.

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of each BGP dynamic routing agent.

```
$ neutron agent-list --agent-type="BGP dynamic routing agent"
+-----+-----+-----+
| id | agent_type | host |
+-----+-----+-----+
| 37729181-2224-48d8-89ef-16eca8e2f77e | BGP dynamic routing agent | controller |
| :-( | True | neutron-bgp-dragent |
+-----+-----+-----+
```

Create the address scope and subnet pools

1. Create an address scope. The provider (external) and self-service networks must belong to the same address scope for the agent to advertise those self-service network prefixes.

```
$ openstack address scope create --share --ip-version 4 bgp
+-----+-----+
| Field | Value |
+-----+-----+
| headers | |
| id | f71c958f-dbe8-49a2-8fb9-19c5f52a37f1 |
| ip_version | 4 |
| name | bgp |
| project_id | 86acdbd1d72745fd8e8320edd7543400 |
| shared | True |
+-----+-----+
```

2. Create subnet pools. The provider and self-service networks use different pools.

- Create the provider network pool.

```
$ openstack subnet pool create --pool-prefix 203.0.113.0/24 \
--address-scope bgp provider
+-----+-----+
| Field | Value |
+-----+-----+
```

address_scope_id	f71c958f-dbe8-49a2-8fb9-19c5f52a37f1
created_at	2017-01-12T14:58:57Z
default_prefixlen	8
default_quota	None
description	
headers	
id	63532225-b9a0-445a-9935-20a15f9f68d1
ip_version	4
is_default	False
max_prefixlen	32
min_prefixlen	8
name	provider
prefixes	203.0.113.0/24
project_id	86acdbd1d72745fd8e8320edd7543400
revision_number	1
shared	False
updated_at	2017-01-12T14:58:57Z
+	-----+-----+

- Create the self-service network pool.

\$ openstack subnet pool create --pool-prefix 10.0.1.0/24 \	
--pool-prefix 10.0.2.0/24 --address-scope bgp \	
--share selfservice	
+	-----+-----+
Field	Value
+	-----+-----+
address_scope_id	f71c958f-dbe8-49a2-8fb9-19c5f52a37f1
created_at	2017-01-12T15:02:31Z
default_prefixlen	8
default_quota	None
description	
headers	
id	8d8270b1-b194-4b7e-914c-9c741dc9bd49b
ip_version	4
is_default	False
max_prefixlen	32
min_prefixlen	8
name	selfservice
prefixes	10.0.1.0/24, 10.0.2.0/24
project_id	86acdbd1d72745fd8e8320edd7543400
revision_number	1
shared	True
updated_at	2017-01-12T15:02:31Z
+	-----+-----+

Create the provider and self-service networks

1. Create the provider network.

```
$ openstack network create provider --external --provider-physical-network \
  provider --provider-network-type flat
Created a new network:
+
```

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2016-12-21T08:47:41Z
description	
headers	
id	190ca651-2ee3-4a4b-891f-dedda47974fe
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
mtu	1450
name	provider
port_security_enabled	True
project_id	c961a8f6d3654657885226378ade8220
provider:network_type	flat
provider:physical_network	provider
provider:segmentation_id	66
revision_number	3
router:external	External
shared	False
status	ACTIVE
subnets	
tags	[]
updated_at	2016-12-21T08:47:41Z

2. Create a subnet on the provider network using an IP address range from the provider subnet pool.

\$ neutron subnet-create --name provider --subnetpool provider \	
--prefixlen 24 --allocation-pool start=203.0.113.11,end=203.0.113.254 \	
--gateway 203.0.113.1 provider	
Created a new subnet:	
Field	Value
allocation_pools	{"start": "203.0.113.11", "end": "203.0.113.254"}
cidr	203.0.113.0/24
created_at	2016-03-17T23:17:16
description	
dns_nameservers	
enable_dhcp	True
gateway_ip	203.0.113.1
host_routes	
id	8ed65d41-2b2a-4f3a-9f92-45adb266e01a
ip_version	4
ipv6_address_mode	
ipv6_ra_mode	
name	provider
network_id	68ec148c-181f-4656-8334-8f4eb148689d
subnetpool_id	3771c0e7-7096-46d3-a3bd-699c58e70259
tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d
updated_at	2016-03-17T23:17:16

Note: The IP address allocation pool starting at .11 improves clarity of the diagrams. You can safely omit it.

3. Create the self-service networks.

```
$ openstack network create selfservice1
Created a new network:
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2016-12-21T08:49:38Z |
| description | |
| headers | |
| id | 9d842606-ef3d-4160-9ed9-e03fa63aed96 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| mtu | 1450 |
| name | selfservice1 |
| port_security_enabled | True |
| project_id | c961a8f6d3654657885226378ade8220 |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 106 |
| revision_number | 3 |
| router:external | Internal |
| shared | False |
| status | ACTIVE |
| subnets | |
| tags | [] |
| updated_at | 2016-12-21T08:49:38Z |
+-----+-----+

$ openstack network create selfservice2
Created a new network:
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2016-12-21T08:50:05Z |
| description | |
| headers | |
| id | f85639e1-d23f-438e-b2b1-f40570d86b1c |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| mtu | 1450 |
| name | selfservice2 |
| port_security_enabled | True |
| project_id | c961a8f6d3654657885226378ade8220 |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 21 |
```

```

| revision_number      | 3
| router:external     | Internal
| shared               | False
| status               | ACTIVE
| subnets              |
| tags                | []
| updated_at           | 2016-12-21T08:50:05Z
+-----+-----+-----+
$ openstack network create selfservice3
Created a new network:
+-----+-----+-----+
| Field          | Value          |
+-----+-----+-----+
| admin_state_up | UP
| availability_zone_hints | 
| availability_zones | 
| created_at     | 2016-12-21T08:50:35Z
| description    | 
| headers        | 
| id             | eeccdb82-5cf4-4999-8ab3-e7dc99e7d43b
| ipv4_address_scope | None
| ipv6_address_scope | None
| mtu            | 1450
| name           | selfservice3
| port_security_enabled | True
| project_id     | c961a8f6d3654657885226378ade8220
| provider:network_type | vxlan
| provider:physical_network | None
| provider:segmentation_id | 86
| revision_number | 3
| router:external | Internal
| shared          | False
| status          | ACTIVE
| subnets         |
| tags            | []
| updated_at     | 2016-12-21T08:50:35Z
+-----+-----+-----+

```

4. Create a subnet on the first two self-service networks using an IP address range from the self-service subnet pool.

```

$ neutron subnet-create --name selfservice1 --subnetpool selfservice \
--prefixlen 24 selfservice1
Created a new subnet:
+-----+-----+-----+
| Field          | Value          |
+-----+-----+-----+
| allocation_pools | [{"start": "10.0.1.2", "end": "10.0.1.254"}] |
| cidr           | 10.0.1.0/24
| created_at     | 2016-03-17T23:20:20
| description    | 
| dns_nameservers | 
| enable_dhcp    | True
| gateway_ip     | 10.0.1.1
| host_routes    |
| id             | 8edd3dc2-df40-4d71-816e-a4586d61c809
+-----+-----+-----+

```

```

| ip_version | 4
| ipv6_address_mode |
| ipv6_ra_mode |
| name | selfservice1
| network_id | be79de1e-5f56-11e6-9dfb-233e41cec48c
| subnetpool_id | c7e9737a-cfd3-45b5-a861-d1cee1135a92
| tenant_id | b3ac05ef10bf441fbf4aa17f16ae1e6d
| updated_at | 2016-03-17T23:20:20
+-----+-----+
$ neutron subnet-create --name selfservice2 --subnetpool selfservice \
--prefixlen 24 selfservice2
Created a new subnet:
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | {"start": "10.0.2.2", "end": "10.0.2.254"} |
| cidr | 10.0.2.0/24
| created_at | 2016-03-17T23:20:20
| description |
| dns_nameservers |
| enable_dhcp | True
| gateway_ip | 10.0.2.1
| host_routes |
| id | 8edd3dc2-df40-4d71-816e-a4586d61c809
| ip_version | 4
| ipv6_address_mode |
| ipv6_ra_mode |
| name | selfservice2
| network_id | c1fd9846-5f56-11e6-a8ac-0f998d9cc0a2
| subnetpool_id | c7e9737a-cfd3-45b5-a861-d1cee1135a92
| tenant_id | b3ac05ef10bf441fbf4aa17f16ae1e6d
| updated_at | 2016-03-17T23:20:20
+-----+-----+

```

5. Create a subnet on the last self-service network using an IP address range outside of the address scope.

```

$ neutron subnet-create --name subnet3 selfservice3 10.0.3.0/24
Created a new subnet:
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | {"start": "10.0.3.2", "end": "10.0.3.254"} |
| cidr | 10.0.3.0/24
| created_at | 2016-03-17T23:20:20
| description |
| dns_nameservers |
| enable_dhcp | True
| gateway_ip | 10.0.3.1
| host_routes |
| id | cd9f9156-5f59-11e6-aec-172ec7ee939a
| ip_version | 4
| ipv6_address_mode |
| ipv6_ra_mode |
| name | selfservice3
| network_id | c283dc1c-5f56-11e6-bfb6-efc30e1eb73b
| subnetpool_id |
+-----+-----+

```

tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d
updated_at	2016-03-17T23:20:20

Create and configure the routers

1. Create the routers.

```
$ openstack router create router1
+-----+-----+
| Field          | Value
+-----+-----+
| admin_state_up | UP
| availability_zone_hints |
| availability_zones |
| created_at     | 2017-01-10T13:15:19Z
| description    |
| distributed    | False
| external_gateway_info | null
| flavor_id      | None
| ha             | False
| headers         |
| id              | 3f6f4ef8-63be-11e6-bbb3-2fbcef363ab8
| name            | router1
| project_id     | b3ac05ef10bf441fbf4aa17f16ae1e6d
| revision_number | 1
| routes          |
| status          | ACTIVE
| updated_at      | 2017-01-10T13:15:19Z
+-----+-----+

$ openstack router create router2
+-----+-----+
| Field          | Value
+-----+-----+
| admin_state_up | UP
| availability_zone_hints |
| availability_zones |
| created_at     | 2017-01-10T13:15:19Z
| description    |
| distributed    | False
| external_gateway_info | null
| flavor_id      | None
| ha             | False
| headers         |
| id              | 3fd21a60-63be-11e6-9c95-5714c208c499
| name            | router2
| project_id     | b3ac05ef10bf441fbf4aa17f16ae1e6d
| revision_number | 1
| routes          |
| status          | ACTIVE
| updated_at      | 2017-01-10T13:15:19Z
+-----+-----+

$ openstack router create router3
```

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2017-01-10T13:15:19Z
description	
distributed	False
external_gateway_info	null
flavor_id	None
ha	False
headers	
id	40069a4c-63be-11e6-9ecc-e37c1eaa7e84
name	router3
project_id	b3ac05ef10bf441fbf4aa17f16ae1e6d
revision_number	1
routes	
status	ACTIVE
updated_at	2017-01-10T13:15:19Z

2. For each router, add one self-service subnet as an interface on the router.

```
$ neutron router-interface-add router1 selfservice1
Added interface 90e3880a-5f5c-11e6-914c-9f3e20c8c151 to router router1.

$ neutron router-interface-add router2 selfservice2
Added interface 91628362-5f5c-11e6-826a-7322fb03a821 to router router2.

$ neutron router-interface-add router3 selfservice3
Added interface 91d51044-5f5c-11e6-bf55-ffd180541cc2 to router router3.
```

3. Add the provider network as a gateway on each router.

```
$ neutron router-gateway-set router1 provider
Set gateway for router router1

$ neutron router-gateway-set router2 provider
Set gateway for router router2

$ neutron router-gateway-set router3 provider
Set gateway for router router3
```

Create and configure the BGP speaker

The BGP speaker advertises the next-hop IP address for eligible self-service networks and floating IP addresses for instances using those networks.

1. Create the BGP speaker.

```
$ neutron bgp-speaker-create --ip-version 4 \
    --local-as LOCAL_AS bgpspeaker
Created a new bgp_speaker:
```

Field	Value
advertise_floating_ip_host_routes	True
advertise_tenant_networks	True
id	5f227f14-4f46-4eca-9524-fc5a1eabc358
ip_version	4
local_as	1234
name	bgpspeaker
networks	
peers	
tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d

Replace LOCAL_AS with an appropriate local autonomous system number. The example configuration uses AS 1234.

2. A BGP speaker requires association with a provider network to determine eligible prefixes. The association builds a list of all virtual routers with gateways on provider and self-service networks in the same address scope so the BGP speaker can advertise self-service network prefixes with the corresponding router as the next-hop IP address. Associate the BGP speaker with the provider network.

```
$ neutron bgp-speaker-network-add bgpspeaker provider
Added network provider to BGP speaker bgpspeaker.
```

3. Verify association of the provider network with the BGP speaker.

Field	Value
advertise_floating_ip_host_routes	True
advertise_tenant_networks	True
id	5f227f14-4f46-4eca-9524-fc5a1eabc358
ip_version	4
local_as	1234
name	bgpspeaker
networks	68ec148c-181f-4656-8334-8f4eb148689d
peers	
tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d

4. Verify the prefixes and next-hop IP addresses that the BGP speaker advertises.

```
$ neutron bgp-speaker-advertiserroute-list bgpspeaker
+-----+
| destination | next_hop |
+-----+
| 10.0.1.0/24 | 203.0.113.11 |
| 10.0.2.0/24 | 203.0.113.12 |
+-----+
```

5. Create a BGP peer.

```
$ neutron bgp-peer-create --peer-ip 192.0.2.1 \
--remote-as REMOTE_AS bgppeer
Created a new bgp_peer:
+-----+
```

Field	Value
auth_type	none
id	35c89ca0-ac5a-4298-a815-0b073c2362e9
name	bgppeer
peer_ip	192.0.2.1
remote_as	4321
tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d

Replace REMOTE_AS with an appropriate remote autonomous system number. The example configuration uses AS 4321 which triggers EBGP peering.

Note: The host containing the BGP agent must have layer-3 connectivity to the provider router.

6. Add a BGP peer to the BGP speaker.

```
$ neutron bgp-speaker-peer-add bgpspeaker bgppeer
Added BGP peer bgppeer to BGP speaker bgpspeaker.
```

7. Verify addition of the BGP peer to the BGP speaker.

\$ neutron bgp-speaker-show bgpspeaker	
Field	Value
advertise_floating_ip_host_routes	True
advertise_tenant_networks	True
id	5f227f14-4f46-4eca-9524-fc5a1eabc358
ip_version	4
local_as	1234
name	bgpspeaker
networks	68ec148c-181f-4656-8334-8f4eb148689d
peers	35c89ca0-ac5a-4298-a815-0b073c2362e9
tenant_id	b3ac05ef10bf441fbf4aa17f16ae1e6d

Note: After creating a peering session, you cannot change the local or remote autonomous system numbers.

Schedule the BGP speaker to an agent

1. Unlike most agents, BGP speakers require manual scheduling to an agent. BGP speakers only form peering sessions and begin prefix advertisement after scheduling to an agent. Schedule the BGP speaker to agent 37729181-2224-48d8-89ef-16eca8e2f77e.

```
$ neutron bgp-dragent-speaker-add 37729181-2224-48d8-89ef-16eca8e2f77e bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.
```

2. Verify scheduling of the BGP speaker to the agent.

```
$ neutron bgp-dragent-list-hosting-speaker bgpspeaker
+-----+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+-----+
| 37729181-2224-48d8-89ef-16eca8e2f77e | controller | True | :-)
+-----+-----+-----+-----+
$ neutron bgp-speaker-list-on-dagent 37729181-2224-48d8-89ef-16eca8e2f77e
+-----+-----+-----+-----+
| id | name | local_as | ip_version |
+-----+-----+-----+-----+
| 5f227f14-4f46-4eca-9524-fc5a1eabc358 | bgpspeaker | 1234 | 4 |
+-----+-----+-----+-----+
```

Prefix advertisement

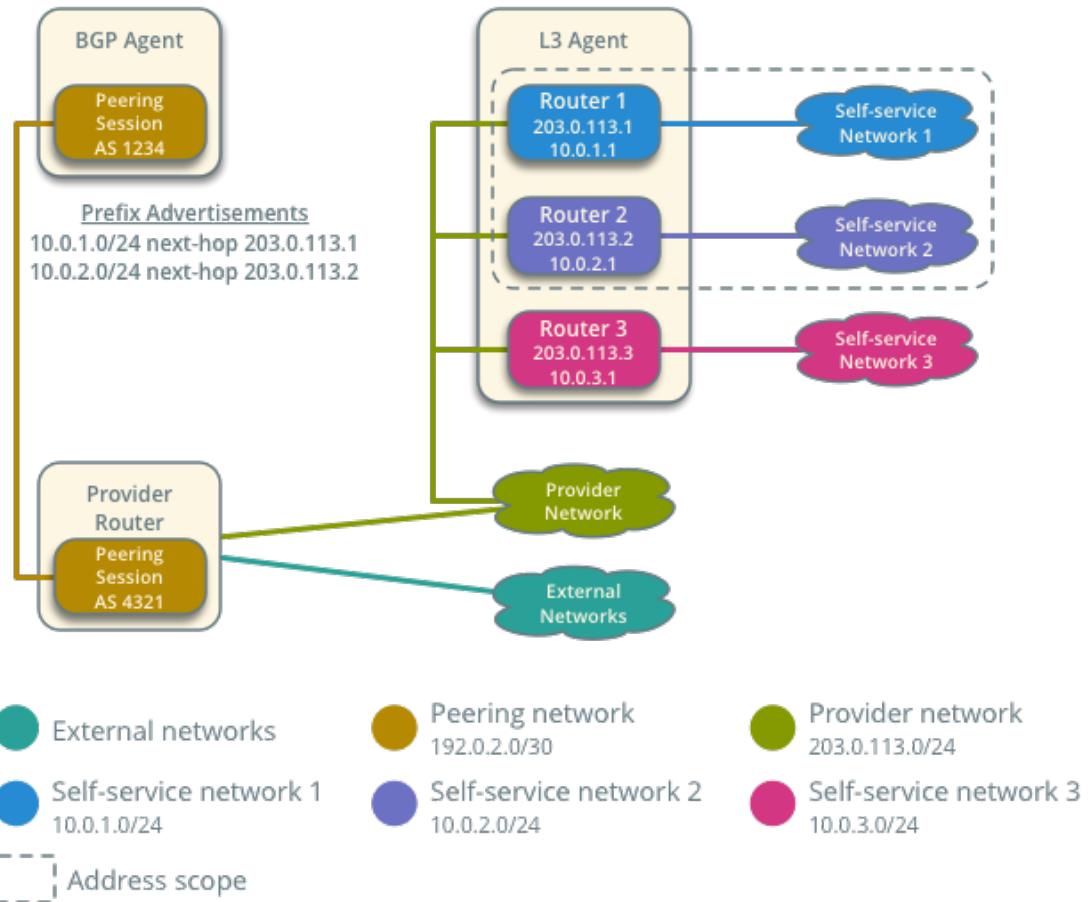
BGP dynamic routing advertises prefixes for self-service networks and host routes for floating IP addresses.

Advertisement of a self-service network requires satisfying the following conditions:

- The external and self-service network reside in the same address scope.
- The router contains an interface on the self-service subnet and a gateway on the external network.
- The BGP speaker associates with the external network that provides a gateway on the router.
- The BGP speaker has the `advertise_tenant_networks` attribute set to `True`.

BGP Dynamic Routing

Example with self-service networks

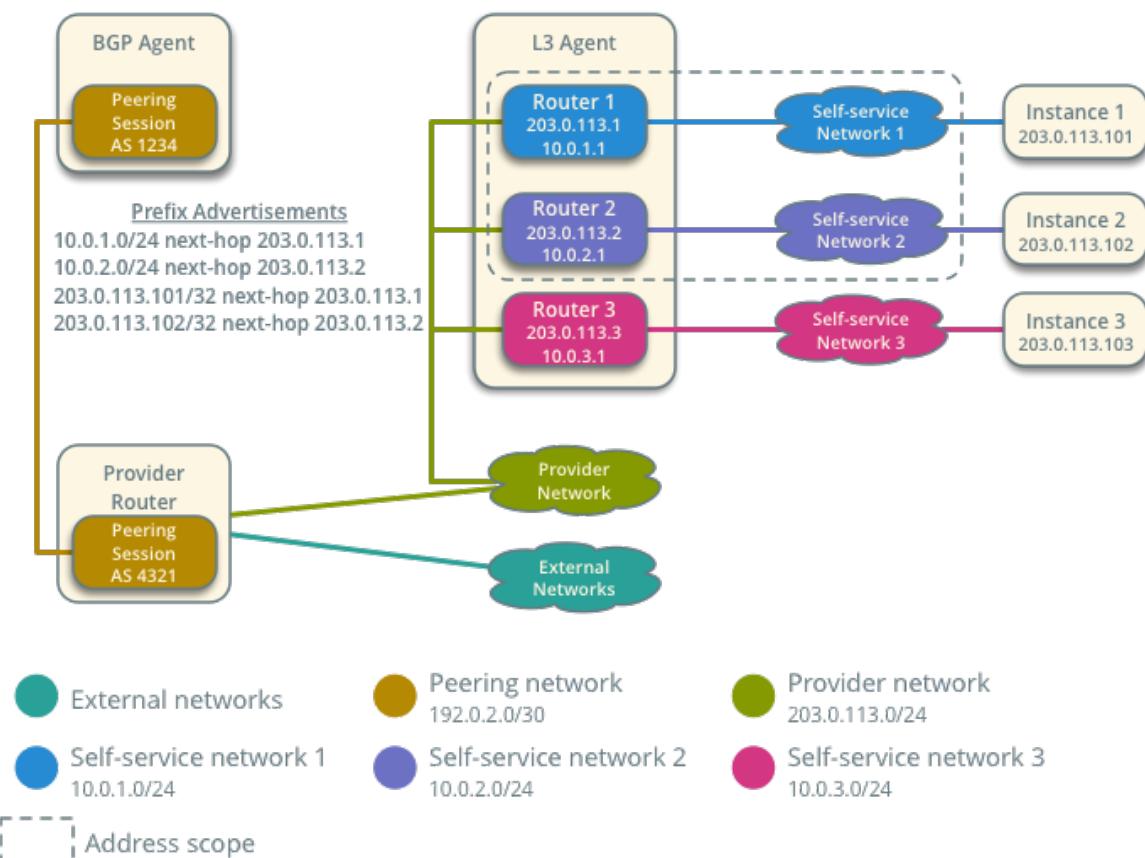


Advertisement of a floating IP address requires satisfying the following conditions:

- The router with the floating IP address binding contains a gateway on an external network with the BGP speaker association.
- The BGP speaker has the `advertise_floating_ip_host_routes` attribute set to True.

BGP Dynamic Routing

Example with floating IP addresses



Operation with Distributed Virtual Routers (DVR)

In deployments using DVR, the BGP speaker advertises floating IP addresses and self-service networks differently. For floating IP addresses, the BGP speaker advertises the floating IP agent gateway on the corresponding compute node as the next-hop IP address. For self-service networks using SNAT, the BGP speaker advertises the DVR SNAT node as the next-hop IP address.

For example, consider the following components:

1. A provider network using IP address range 203.0.113.0/24, and supporting floating IP addresses 203.0.113.101, 203.0.113.102, and 203.0.113.103.
2. A self-service network using IP address range 10.0.1.0/24.
3. The SNAT gateway resides on 203.0.113.11.
4. The floating IP agent gateways (one per compute node) reside on 203.0.113.12, 203.0.113.13, and 203.0.113.14.
5. Three instances, one per compute node, each with a floating IP address.

```
$ neutron bgp-speaker-advertiseroute-list bgpspeaker
+-----+-----+
| destination | next_hop |
```

10.0.1.0/24	203.0.113.11	
203.0.113.101/32	203.0.113.12	
203.0.113.102/32	203.0.113.13	
203.0.113.103/32	203.0.113.14	

Note: DVR lacks support for routing directly to a fixed IP address via the floating IP agent gateway port and thus prevents the BGP speaker from advertising fixed IP addresses.

You can also identify floating IP agent gateways in your environment to assist with verifying operation of the BGP speaker.

\$ neutron port-list --device_owner="network:floatingip_agent_gateway"			
id	name	mac_address	fixed_ips
87cf2970-4970-462e-939e-00e808295dfa	fa:16:3e:7c:68:e3	{"subnet_id": "8ed65d41-2b2a-4f3a-9f92-45adb266e01a", "ip_address": "203.0.113.12"}	
8d218440-0d2e-49d0-8a7b-3266a6146dc1	fa:16:3e:9d:78:cf	{"subnet_id": "8ed65d41-2b2a-4f3a-9f92-45adb266e01a", "ip_address": "203.0.113.13"}	
87cf2970-4970-462e-939e-00e802281dfa	fa:16:3e:6b:18:e0	{"subnet_id": "8ed65d41-2b2a-4f3a-9f92-45adb266e01a", "ip_address": "203.0.113.14"}	

IPv6

BGP dynamic routing supports peering via IPv6 and advertising IPv6 prefixes.

- To enable peering via IPv6, create a BGP peer and use an IPv6 address for `peer_ip`.
- To enable advertising IPv6 prefixes, create an address scope with `ip_version=6` and a BGP speaker with `ip_version=6`.

Note: DVR with IPv6 functions similarly to DVR with IPv4.

High availability

BGP dynamic routing supports scheduling a BGP speaker to multiple agents which effectively multiplies prefix advertisements to the same peer. If an agent fails, the peer continues to receive advertisements from one or more operational agents.

1. Show available dynamic routing agents.

\$ neutron agent-list --agent-type="BGP dynamic routing agent"			
+	-	-	-

id	agent_type	host	
↳availability_zone	alive admin_state_up binary		
37729181-2224-48d8-89ef-16eca8e2f77e	BGP dynamic routing agent bgp-ha1		
(:-) True	neutron-bgp-dragnet		
1a2d33bb-9321-30a2-76ab-22eff3d2f56a	BGP dynamic routing agent bgp-ha2		
(:-) True	neutron-bgp-dragnet		

2. Schedule BGP speaker to multiple agents.

```
$ neutron bgp-dragnet-speaker-add 37729181-2224-48d8-89ef-16eca8e2f77e bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.

$ neutron bgp-dragnet-speaker-add 1a2d33bb-9321-30a2-76ab-22eff3d2f56a bgpspeaker
Associated BGP speaker bgpspeaker to the Dynamic Routing agent.

$ neutron bgp-dragnet-list-hosting-speaker bgpspeaker
+-----+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+-----+
| 37729181-2224-48d8-89ef-16eca8e2f77e | bgp-ha1 | True | (:-) |
| 1a2d33bb-9321-30a2-76ab-22eff3d2f56a | bgp-ha2 | True | (:-) |
+-----+-----+-----+-----+

$ neutron bgp-speaker-list-on-dragnet 37729181-2224-48d8-89ef-16eca8e2f77e
+-----+-----+-----+-----+
| id | name | local_as | ip_version |
+-----+-----+-----+-----+
| 5f227f14-4f46-4eca-9524-fc5a1eabc358 | bgpspeaker | 1234 | 4 |
+-----+-----+-----+-----+

$ neutron bgp-speaker-list-on-dragnet 1a2d33bb-9321-30a2-76ab-22eff3d2f56a
+-----+-----+-----+-----+
| id | name | local_as | ip_version |
+-----+-----+-----+-----+
| 5f227f14-4f46-4eca-9524-fc5a1eabc358 | bgpspeaker | 1234 | 4 |
+-----+-----+-----+-----+
```

High-availability for DHCP

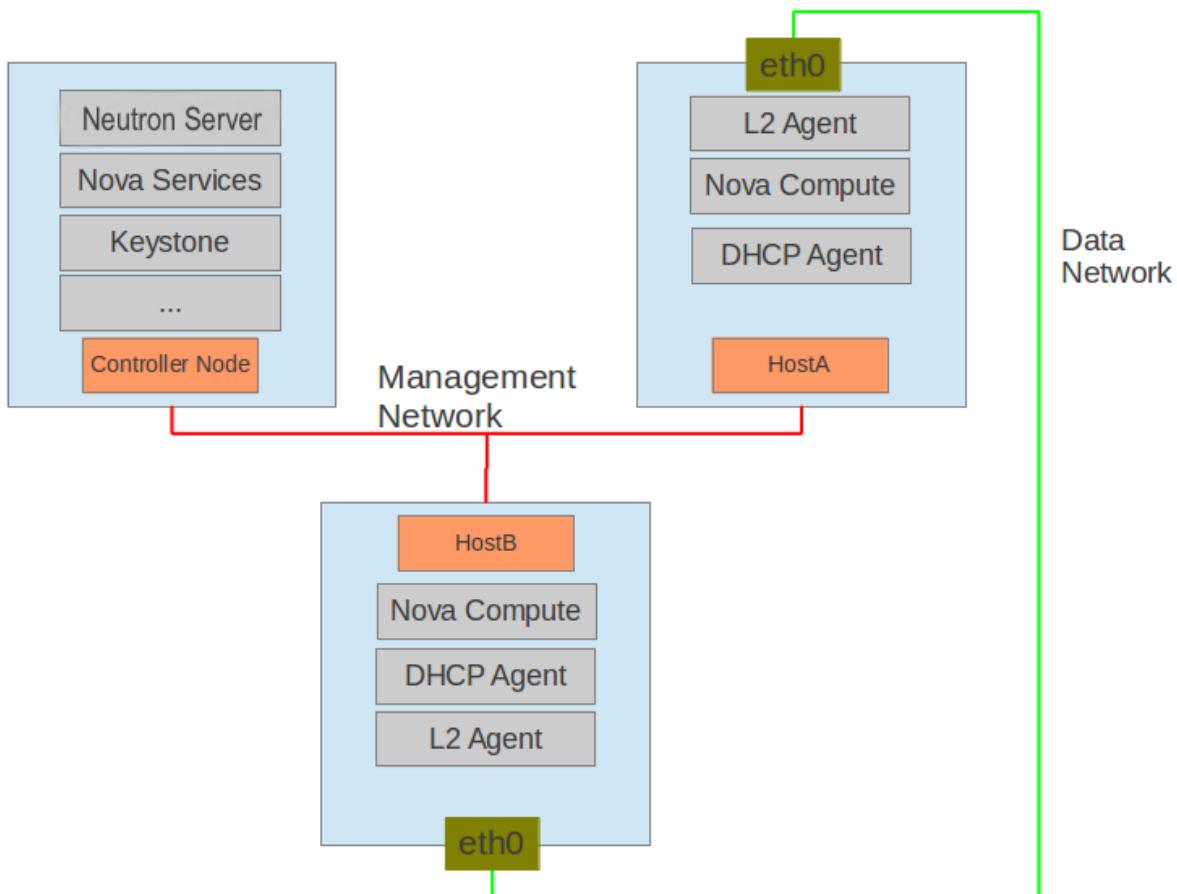
This section describes how to use the agent management (alias agent) and scheduler (alias agent_scheduler) extensions for DHCP agents scalability and HA.

Note: Use the **neutron ext-list** client command to check if these extensions are enabled. Check agent and agent_scheduler are included in the output.

\$ openstack extension list --network -c Name -c Alias	
Name	Alias
Default Subnetpools	default-subnetpools

Network IP Availability	network-ip-availability
Network Availability Zone	network_availability_zone
Auto Allocated Topology Services	auto-allocated-topology
Neutron L3 Configurable external gateway mode	ext-gw-mode
Port Binding	binding
Neutron Metering	metering
agent	agent
Subnet Allocation	subnet_allocation
L3 Agent Scheduler	l3_agent_scheduler
Tag support	tag
Neutron external network	external-net
Neutron Service Flavors	flavors
Network MTU	net-mtu
Availability Zone	availability_zone
Quota management support	quotas
HA Router extension	l3-ha
Provider Network	provider
Multi Provider Network	multi-provider
Address scope	address-scope
Neutron Extra Route	extraroute
Subnet service types	subnet-service-types
Resource timestamps	standard-attr-timestamp
Neutron Service Type Management	service-type
Router Flavor Extension	l3-flavors
Tag support for resources: subnet, subnetpool, port, router	tag-ext
Neutron Extra DHCP opts	extra_dhcp_opt
Resource revision numbers	standard-attr-revisions
Pagination support	pagination
Sorting support	sorting
security-group	security-group
DHCP Agent Scheduler	dhcp_agent_scheduler
Router Availability Zone	router_availability_zone
RBAC Policies	rbac-policies
standard-attr-description	standard-attr-description
Neutron L3 Router	router
Allowed Address Pairs	allowed-address-pairs
project_id field enabled	project-id
Distributed Virtual Router	dvr
+-----+	+-----+

Demo setup



There will be three hosts in the setup.

Host	Description
OpenStack controller host - controlnode	Runs the Networking, Identity, and Compute services that are required to deploy VMs. The node must have at least one network interface that is connected to the Management Network. Note that nova-network should not be running because it is replaced by Neutron.
HostA	Runs nova-compute, the Neutron L2 agent and DHCP agent
HostB	Same as HostA

Configuration

controlnode: neutron server

1. Neutron configuration file /etc/neutron/neutron.conf:

```
[DEFAULT]
core_plugin = linuxbridge
rabbit_host = controlnode
allow_overlapping_ips = True
host = controlnode
agent_down_time = 5
dhcp_agents_per_network = 1
```

Note: In the above configuration, we use `dhcp_agents_per_network = 1` for this demonstration. In usual deployments, we suggest setting `dhcp_agents_per_network` to more than one to match the number of DHCP agents in your deployment. See [Enabling DHCP high availability by default](#).

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

HostA and HostB: L2 agent

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
rabbit_host = controlnode
rabbit_password = openstack
# host = HostB on hostb
host = HostA
```

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

3. Update the nova configuration file `/etc/nova/nova.conf`:

```
[DEFAULT]
use_neutron=True
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
admin_username=neutron
admin_password=servicepassword
admin_auth_url=http://controlnode:35357/v2.0/
auth_strategy=keystone
admin_tenant_name=servicetenant
url=http://203.0.113.10:9696/
```

HostA and HostB: DHCP agent

- Update the DHCP configuration file `/etc/neutron/dhcp_agent.ini`:

[DEFAULT]

```
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

Prerequisites for demonstration

Admin role is required to use the agent management and scheduler extensions. Ensure you run the following commands under a project with an admin role.

To experiment, you need VMs and a neutron network:

```
$ openstack server list
+-----+-----+-----+-----+
| ID | Name | Status | Networks | Image Name |
+-----+-----+-----+-----+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=192.0.2.3 | cirros |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=192.0.2.4 | ubuntu |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=192.0.2.5 | centos |
+-----+-----+-----+-----+
$ openstack network list
+-----+-----+
| ID | Name | Subnets |
+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-d5cf646db9d1 |
+-----+-----+
```

Managing agents in neutron deployment

1. List all agents:

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability |
+-----+-----+-----+
| 22467163-01ea-4231-ba45-3bd316f425e6 | Linux bridge agent | HostA | None |
| 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b | DHCP agent | HostA | None |
| 3066d20c-9f8f-440c-ae7c-a40ffb4256b6 | Linux bridge agent | HostB | nova |
| 55569f4e-6f31-41a6-be9d-526efce1f7fe | DHCP agent | HostB | nova |
+-----+-----+-----+
```

Every agent that supports these extensions will register itself with the neutron server when it starts up.

The output shows information for four agents. The alive field shows : -) if the agent reported its state within the period defined by the agent_down_time option in the neutron.conf file. Otherwise the alive is xxx.

2. List DHCP agents that host a specified network:

```
$ neutron 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.

```
$ neutron net-list-on-dhcp-agent a0c1c21c-d4f4-4577-9ec7-908f2d48622d
+-----+-----+
| id | name | subnets |
+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-d5cf646db9d1
| 192.0.2.0/24 |
+-----+-----+
```

4. Show agent details.

The **openstack network agent show** command shows details for a specified agent:

```
$ openstack network agent show 22467163-01ea-4231-ba45-3bd316f425e6
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| agent_type | Metering agent |
| alive | False |
| availability_zone | None |
| binary | neutron-metering-agent |
| configurations | measure_interval='30', metering_driver='neutron.services.metering.driver' |
| | rs.noop.noop_driver.NoopMeteringDriver', report_interval='300' |
| created_at | 2016-10-08 15:17:14 |
| description | None |
| heartbeat_timestamp | 2016-10-24 13:53:35 |
| host | HostA |
```

id	22467163-01ea-4231-ba45-3bd316f425e6	□
↵		□
started_at	2016-10-08 15:17:14	□
↵		□
topic	dhcp_agent	□
↵		□
+-----+-----+		
↵		□

In this output, heartbeat_timestamp is the time on the neutron server. You do not need to synchronize all agents to this time for this extension to run correctly. configurations describes the static configuration for the agent or run time data. This agent is a DHCP agent and it hosts one network, one subnet, and three ports.

Different types of agents show different details. The following output shows information for a Linux bridge agent:

\$ openstack network agent show 22467163-01ea-4231-ba45-3bd316f425e6		
+-----+-----+		
Field	Value	□
↵		□
+-----+-----+		
admin_state_up	UP	□
↵		□
agent_type	Metering agent	□
↵		□
alive	False	□
↵		□
availability_zone	None	□
↵		□
binary	neutron-linuxbridge-agent	□
↵		□
configurations	measure_interval='30', metering_driver='neutron.services.metering.driver'	□
↵	rs.noop.noop_driver.NoopMeteringDriver', report_interval='300	□
created_at	2016-10-08 15:17:14	□
↵		□
description	None	□
↵		□
heartbeat_timestamp	2016-10-24 13:53:35	□
↵		□
host	HostB	□
↵		□
id	22467163-01ea-4231-ba45-3bd316f425e6	□
↵		□
started_at	2016-10-08 15:17:14	□
↵		□
topic	dhcp_agent	□
↵		□
+-----+-----+		
↵		□

The output shows bridge-mapping and the number of virtual network devices on this L2 agent.

Managing assignment of networks to DHCP agent

A single network can be assigned to more than one DHCP agents and one DHCP agent can host more than one network. You can add a network to a DHCP agent and remove one from it.

1. Default scheduling.

When you create a network with one port, the network will be scheduled to an active DHCP agent. If many active DHCP agents are running, select one randomly. You can design more sophisticated scheduling algorithms in the same way as nova-schedule later on.

```
$ neutron net-create net2
$ neutron subnet-create net2 198.51.100.0/24 --name subnet2
$ neutron port-create net2
$ openstack network agent list --agent-type dhcp --host qiaomin-free
+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability Zone |
+-----+-----+-----+-----+
| e838ef5c-75b1-4b12-84da-7bdbd62f1040 | DHCP agent | HostA | nova |
| True | UP | neutron-dhcp-agent |
+-----+-----+-----+-----+
```

It is allocated to DHCP agent on HostA. If you want to validate the behavior through the **dnsmasq** command, you must create a subnet for the network because the DHCP agent starts the dnsmasq service only if there is a DHCP.

2. Assign a network to a given DHCP agent.

To add another DHCP agent to host the network, run this command:

```
$ neutron dhcp-agent-network-add f28aa126-6edb-4ea5-a81e-8850876bc0a8 net2
Added network net2 to dhcp agent
$ openstack network agent list --agent-type dhcp --host qiaomin-free
+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability Zone |
+-----+-----+-----+-----+
| e838ef5c-75b1-4b12-84da-7bdbd62f1040 | DHCP agent | HostA | nova |
| True | UP | neutron-dhcp-agent |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent | HostB | nova |
| True | UP | neutron-dhcp-agent |
+-----+-----+-----+-----+
```

Both DHCP agents host the net2 network.

3. Remove a network from a specified DHCP agent.

This command is the sibling command for the previous one. Remove net2 from the DHCP agent for HostA:

```
$ neutron dhcp-agent-network-remove a0c1c21c-d4f4-4577-9ec7-908f2d48622d \
net2
Removed network net2 to dhcp agent
$ neutron dhcp-agent-list-hosting-net net2
+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True | :-) |
+-----+-----+-----+
```

You can see that only the DHCP agent for HostB is hosting the net2 network.

HA of DHCP agents

Boot a VM on net2. Let both DHCP agents host net2. Fail the agents in turn to see if the VM can still get the desired IP.

1. Boot a VM on net2:

```
$ openstack network list
+-----+-----+-----+
| ID | Name | Subnets |
+-----+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-d5cf646db9d1 |
| 9b96b14f-71b8-4918-90aa-c5d705606b1a | net2 | 6979b71a-0ae8-448c-aa87-65f68eedcaa |
+-----+-----+-----+
$ openstack server create --image tty --flavor 1 myserver4 \
--nic net-id=9b96b14f-71b8-4918-90aa-c5d705606b1a
...
$ openstack server list
+-----+-----+-----+
| ID | Name | Status | Networks |
+-----+-----+-----+
| c394fc0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=192.0.2.3 |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=192.0.2.4 |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=192.0.2.5 |
| f62f4731-5591-46b1-9d74-f0c901de567f | myserver4 | ACTIVE | net2=198.51.100.2 |
+-----+-----+-----+
```

2. Make sure both DHCP agents hosting net2:

Use the previous commands to assign the network to agents.

```
$ neutron 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 :-)
-----+-----+-----+-----+

To test the HA of DHCP agent:

1. Log in to the `myserver4` VM, and run `udhcpc`, `dhclient` or other DHCP client.
2. Stop the DHCP agent on HostA. Besides stopping the `neutron-dhcp-agent` binary, you must stop the `dnsmasq` processes.
3. Run a DHCP client in VM to see if it can get the wanted IP.
4. Stop the DHCP agent on HostB too.
5. Run `udhcpc` in the VM; it cannot get the wanted IP.
6. Start DHCP agent on HostB. The VM gets the wanted IP again.

Disabling and removing an agent

An administrator might want to disable an agent if a system hardware or software upgrade is planned. Some agents that support scheduling also support disabling and enabling agents, such as L3 and DHCP agents. After the agent is disabled, the scheduler does not schedule new resources to the agent.

After the agent is disabled, you can safely remove the agent. Even after disabling the agent, resources on the agent are kept assigned. Ensure you remove the resources on the agent before you delete the agent.

Disable the DHCP agent on HostA before you stop it:

\$ neutron agent-update a0c1c21c-d4f4-4577-9ec7-908f2d48622d --admin-state-up False
\$ neutron agent-list
\$ openstack network agent list
-----+-----+-----+-----+
ID Agent Type Host Availability Zone
-----+-----+-----+-----+
Alive State Binary
-----+-----+-----+-----+
22467163-01ea-4231-ba45-3bd316f425e6 Linux bridge agent HostA None
True UP neutron-metering-agent
2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b DHCP agent HostA None
True UP neutron-openvswitch-agent
3066d20c-9f8f-440c-ae7c-a40ffb4256b6 Linux bridge agent HostB nova
True UP neutron-l3-agent
55569f4e-6f31-41a6-be9d-526efce1f7fe DHCP agent HostB nova
True UP neutron-l3-agent
-----+-----+-----+-----+
-----+-----+-----+-----+

After you stop the DHCP agent on HostA, you can delete it by the following command:

\$ openstack network agent delete 2444c54d-0d28-460c-ab0f-cd1e6b5d3c7b
\$ openstack network agent list
-----+-----+-----+-----+
ID Agent Type Host Availability Zone
-----+-----+-----+-----+
Alive State Binary
-----+-----+-----+-----+
-----+-----+-----+-----+

22467163-01ea-4231-ba45-3bd316f425e6 Linux bridge agent HostA None	
 True UP neutron-metering-agent	
3066d20c-9f8f-440c-ae7c-a40ffb4256b6 Linux bridge agent HostB nova	
 True UP neutron-l3-agent	
55569f4e-6f31-41a6-be9d-526efce1f7fe DHCP agent HostB nova	
 True UP neutron-l3-agent	
+-----+-----+-----+-----+	
 -----+-----+-----+-----+	

After deletion, if you restart the DHCP agent, it appears on the agent list again.

Enabling DHCP high availability by default

You can control the default number of DHCP agents assigned to a network by setting the following configuration option in the file /etc/neutron/neutron.conf.

```
dhcp_agents_per_network = 3
```

DNS integration

This page serves as a guide for how to use the DNS integration functionality of the Networking service. The functionality described covers DNS from two points of view:

- The internal DNS functionality offered by the Networking service and its interaction with the Compute service.
- Integration of the Compute service and the Networking service with an external DNSaaS (DNS-as-a-Service).

Users can control the behavior of the Networking service in regards to DNS using two attributes associated with ports, networks, and floating IPs. The following table shows the attributes available for each one of these resources:

Resource	dns_name	dns_domain
Ports	Yes	No
Networks	No	Yes
Floating IPs	Yes	Yes

The Networking service internal DNS resolution

The Networking service enables users to control the name assigned to ports by the internal DNS. To enable this functionality, do the following:

1. Edit the /etc/neutron/neutron.conf file and assign a value different to openstacklocal (its default value) to the dns_domain parameter in the [default] section. As an example:

```
dns_domain = example.org.
```

2. Add dns to extension_drivers in the [ml2] section of /etc/neutron/plugins/ml2/ml2_conf.ini. The following is an example:

```
[ml2]
extension_drivers = port_security,dns
```

After re-starting the neutron-server, users will be able to assign a dns_name attribute to their ports.

Note: The enablement of this functionality is prerequisite for the enablement of the Networking service integration with an external DNS service, which is described in detail in [Configuring OpenStack Networking for integration with an external DNS service](#).

The following illustrates the creation of a port with my-port in its dns_name attribute.

Note: The name assigned to the port by the Networking service internal DNS is now visible in the response in the dns_assignment attribute.

```
$ neutron port-create my-net --dns_name my-port
Created a new port:
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| allowed_address_pairs | |
| binding:vnic_type | normal |
| device_id | |
| device_owner | |
| dns_assignment | {"hostname": "my-port", "ip_address": "10.0.1.3", "fqdn": "my-
port.example.org."} |
| dns_name | my-port |
| fixed_ips | {"subnet_id": "6141b474-56cd-430f-b731-71660bb79b79", "ip_address
": "10.0.1.3"} |
| id | fb3c10f4-017e-420c-9be1-8f8c557ae21f |
| mac_address | fa:16:3e:aa:9b:e1 |
| name | |
| network_id | bf2802a0-99a0-4e8c-91e4-107d03f158ea |
| port_security_enabled | True |
| security_groups | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635 |
| status | DOWN |
| tenant_id | d5660cb1e6934612a01b4fb2fb630725 |
```

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

When this functionality is enabled, it is leveraged by the Compute service when creating instances. When allocating ports for an instance during boot, the Compute service populates the `dns_name` attributes of these ports with the `hostname` attribute of the instance, which is a DNS sanitized version of its display name. As a consequence, at the end of the boot process, the allocated ports will be known in the `dnsmasq` associated to their networks by their instance `hostname`.

The following is an example of an instance creation, showing how its `hostname` populates the `dns_name` attribute of the allocated port:

\$ openstack server create --image cirros --flavor 42 \
--nic net-id=37aaff3a-6047-45ac-bf4f-a825e56fd2b3 my_vm
+-----+-----+
-----+-----+
Field Value
-----+-----+-----+-----+
OS-DCF:diskConfig MANUAL
-----+-----+-----+-----+
OS-EXT-AZ:availability_zone
-----+-----+-----+-----+
OS-EXT-STS:power_state 0
-----+-----+-----+-----+
OS-EXT-STS:task_state scheduling
-----+-----+-----+-----+
OS-EXT-STS:vm_state building
-----+-----+-----+-----+
OS-SRV-USG:launched_at -
-----+-----+-----+-----+
OS-SRV-USG:terminated_at -
-----+-----+-----+-----+
accessIPv4
-----+-----+-----+-----+
accessIPv6
-----+-----+-----+-----+
adminPass db45Zvo8Jpfe
-----+-----+-----+-----+
config_drive
-----+-----+-----+-----+
created 2016-02-05T21:35:04Z
-----+-----+-----+-----+
flavor m1.nano (42)
-----+-----+-----+-----+
hostId
-----+-----+-----+-----+
id 66c13cb4-3002-4ab3-8400-7efc2659c363
-----+-----+-----+-----+
image cirros-0.3.5-x86_64-uec(b9d981eb-d21c-4ce2-9dbc- dd38f3d9015f)
-----+-----+-----+-----+
key_name -
-----+-----+-----+-----+
locked False
-----+-----+-----+-----+
metadata {}
-----+-----+-----+-----+

```

| name | my_vm
| os-extended-volumes:volumes_attached | []
| progress | 0
| security_groups | default
| status | BUILD
| tenant_id | d5660cb1e6934612a01b4fb2fb630725
| updated | 2016-02-05T21:35:04Z
| user_id | 8bb6e578cba24e7db9d3810633124525
+-----+
| id | name | mac_address | fixed_ips
+-----+
| b3ecc464-1263-44a7-8c38-2d8a52751773 | fa:16:3e:a8:ce:b8 | {"subnet_id": "277eca5d-9869-474b-960e-6da5951d09f7", "ip_address": "172.24.5.8"} |
| | | | {"subnet_id": "eab47748-3f0a-4775-a09f-b0c24bb64bc4", "ip_address": "2001:db8:10::8"} |
+-----+
| Field | Value
+-----+
| admin_state_up | True
| allowed_address_pairs |
| binding:vnic_type | normal
| device_id | 66c13cb4-3002-4ab3-8400-7efc2659c363
| device_owner | compute:None
| dns_assignment | {"hostname": "my-vm", "ip_address": "172.24.5.8", "fqdn": "my-vm.example.org."}
| | {"hostname": "my-vm", "ip_address": "2001:db8:10::8", "fqdn": "my-vm.example.org."}
| dns_name | my-vm
| extra_dhcp_opts |
+-----+
$ neutron port-list --device_id 66c13cb4-3002-4ab3-8400-7efc2659c363
+-----+-----+-----+
| id | name | mac_address | fixed_ips
+-----+
| b3ecc464-1263-44a7-8c38-2d8a52751773 | fa:16:3e:a8:ce:b8 | {"subnet_id": "277eca5d-9869-474b-960e-6da5951d09f7", "ip_address": "172.24.5.8"} |
| | | | {"subnet_id": "eab47748-3f0a-4775-a09f-b0c24bb64bc4", "ip_address": "2001:db8:10::8"} |
+-----+
$ neutron port-show b3ecc464-1263-44a7-8c38-2d8a52751773
+-----+
| Field | Value
+-----+
| admin_state_up | True
| allowed_address_pairs |
| binding:vnic_type | normal
| device_id | 66c13cb4-3002-4ab3-8400-7efc2659c363
| device_owner | compute:None
| dns_assignment | {"hostname": "my-vm", "ip_address": "172.24.5.8", "fqdn": "my-vm.example.org."}
| | {"hostname": "my-vm", "ip_address": "2001:db8:10::8", "fqdn": "my-vm.example.org."}
| dns_name | my-vm
| extra_dhcp_opts |
+-----+

```

fixed_ips	{"subnet_id": "277eca5d-9869-474b-960e-6da5951d09f7", "ip_address
↵": "172.24.5.8"}	
{"subnet_id": "eab47748-3f0a-4775-a09f-b0c24bb64bc4", "ip_address	
↵": "2001:db8:10::8"}	
id	b3ecc464-1263-44a7-8c38-2d8a52751773
↵	
mac_address	fa:16:3e:a8:ce:b8
↵	
name	
↵	
network_id	37aaff3a-6047-45ac-bf4f-a825e56fd2b3
↵	
port_security_enabled	True
↵	
security_groups	1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635
↵	
status	ACTIVE
↵	
tenant_id	d5660cb1e6934612a01b4fb2fb630725
↵	
+-----+-----+	
↵-----+-----+	

In the above example notice that:

- The name given to the instance by the user, `my_vm`, is sanitized by the Compute service and becomes `my-vm` as the port's `dns_name`.
- The port's `dns_assignment` attribute shows that its FQDN is `my-vm.example.org`. in the Networking service internal DNS, which is the result of concatenating the port's `dns_name` with the value configured in the `dns_domain` parameter in `neutron.conf`, as explained previously.
- The `dns_assignment` attribute also shows that the port's `hostname` in the Networking service internal DNS is `my-vm`.
- Instead of having the Compute service create the port for the instance, the user might have created it and assigned a value to its `dns_name` attribute. In this case, the value assigned to the `dns_name` attribute must be equal to the value that Compute service will assign to the instance's `hostname`, in this example `my-vm`. Otherwise, the instance boot will fail.

Integration with an external DNS service

Users can also integrate the Networking and Compute services with an external DNS. To accomplish this, the users have to:

1. Enable the functionality described in [The Networking service internal DNS resolution](#).
2. Configure an external DNS driver. The Networking service provides a driver reference implementation based on the OpenStack DNS service. It is expected that third party vendors will provide other implementations in the future. For detailed configuration instructions, see [Configuring OpenStack Networking for integration with an external DNS service](#).

Once the `neutron-server` has been configured and restarted, users will have functionality that covers three use cases, described in the following sections. In each of the use cases described below:

- The examples assume the OpenStack DNS service as the external DNS.

- A, AAAA and PTR records will be created in the DNS service.
- Before executing any of the use cases, the user must create in the DNS service under his project a DNS zone where the A and AAAA records will be created. For the description of the use cases below, it is assumed the zone `example.org.` was created previously.
- The PTR records will be created in zones owned by a project with admin privileges. See [Configuring OpenStack Networking for integration with an external DNS service](#) for more details.

Use case 1: Ports are published directly in the external DNS service

In this case, the user is creating ports or booting instances on a network that is accessible externally. The steps to publish the port in the external DNS service are the following:

1. Assign a valid domain name to the network's `dns_domain` attribute. This name must end with a period (.).
2. Boot an instance specifying the externally accessible network. Alternatively, create a port on the externally accessible network specifying a valid value to its `dns_name` attribute. If the port is going to be used for an instance boot, the value assigned to `dns_name` must be equal to the `hostname` that the Compute service will assign to the instance. Otherwise, the boot will fail.

Once these steps are executed, the port's DNS data will be published in the external DNS service. This is an example:

```
$ neutron net-list
+-----+-----+-----+
| id | name | subnets |
+-----+-----+-----+
| 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a | public | a67cfdf7-9d5d-406f-8a19-3f38e4fc3e74 |
| 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 | external | 277eca5d-9869-474b-960e-6da5951d09f7 |
| bf2802a0-99a0-4e8c-91e4-107d03f158ea | my-net | 6141b474-56cd-430f-b731-71660bb79b79 |
| 38c5e950-b450-4c30-83d4-ee181c28aad3 | private | 43414c53-62ae-49bc-aa6c-c9dd7705818a |
+-----+-----+-----+
$ neutron net-update 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 --dns_domain example.org.
Updated network: 37aaff3a-6047-45ac-bf4f-a825e56fd2b3

$ neutron net-show 37aaff3a-6047-45ac-bf4f-a825e56fd2b3
+-----+-----+-----+
| Field | Value |
+-----+-----+-----+
| admin_state_up | True |
```

```

| availability_zone_hints | |
| availability_zones     | nova
| dns_domain             | example.org.
| id                     | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3
| mtu                   | 1450
| name                  | external
| port_security_enabled | True
| provider:network_type | vlan
| provider:physical_network | |
| provider:segmentation_id | 2016
| router:external        | False
| shared                 | True
| status                 | ACTIVE
| subnets                | eab47748-3f0a-4775-a09f-b0c24bb64bc4 |
|                         | 277eca5d-9869-474b-960e-6da5951d09f7 |
| tenant_id              | 04fc2f83966245dba907efb783f8eab9
+-----+-----+-----+
$ designate record-list example.org.
+-----+-----+-----+
| id           | type | name      | data
|             |      |           |
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org. | ns1.devstack.org. malavall.
| us.ibm.com. 1454729414 3600 600 86400 3600 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org. | ns1.devstack.org.
|                                         |
+-----+-----+-----+
|                                         |

$ neutron port-create 37aaff3a-6047-45ac-bf4f-a825e56fd2b3 --dns_name my-vm
Created a new port:
+-----+-----+
| Field          | Value
|             |           |
+-----+-----+
| admin_state_up | True
|             |           |
| allowed_address_pairs | |
|             |           |
| binding:vnic_type | normal
|             |           |
| device_id       |           |
|             |           |
| device_owner    |           |
|             |           |
| dns_assignment  | {"hostname": "my-vm", "ip_address": "172.24.5.9", "fqdn": "my-vm.example.org."}
|             | {"hostname": "my-vm", "ip_address": "2001:db8:10::9", "fqdn": "my-vm.example.org."}
| dns_name        | my-vm
|             |           |
| fixed_ips      | {"subnet_id": "277eca5d-9869-474b-960e-6da5951d09f7", "ip_address": "172.24.5.9"}
|             |           |

```

```

|           | {"subnet_id": "eab47748-3f0a-4775-a09f-b0c24bb64bc4", "ip_address
|   ":" 2001:db8:10::9"} |
| id          | 04be331b-dc5e-410a-9103-9c8983aeb186
| mac_address | fa:16:3e:0f:4b:e4
| name        |
| network_id  | 37aaff3a-6047-45ac-bf4f-a825e56fd2b3
| port_security_enabled | True
| security_groups | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635
| status       | DOWN
| tenant_id   | d5660cb1e6934612a01b4fb2fb630725
+-----+
|           |
$ designate record-list example.org.
+-----+-----+-----+
| id          | type | name          | data
|           |      |               |
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org.    | ns1.devstack.org.❶
| malavall.us.ibm.com. 1455563035 3600 600 86400 3600 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org.    | ns1.devstack.org.❶
|           |      |               |
| 3593591b-181f-4beb-9ab7-67fad7413b37 | A    | my-vm.example.org. | 172.24.5.9
|           |      |               |
| 5649c68f-7a88-48f5-9f87-ccb1f6ae67ca | AAAA | my-vm.example.org. | 2001:db8:10::9
|           |      |               |
+-----+-----+-----+
|           |
$ openstack server create --image cirros --flavor 42 \
--nic port-id=04be331b-dc5e-410a-9103-9c8983aeb186 my_vm
+-----+
| Field          | Value
|           |      |
+-----+
| OS-DCF:diskConfig | MANUAL
|           |      |
| OS-EXT-AZ:availability_zone |      |
|           |      |
| OS-EXT-STS:power_state | 0
|           |      |
| OS-EXT-STS:task_state | scheduling
|           |      |
| OS-EXT-STS:vm_state | building
|           |

```

OS-SRV-USG:launched_at	-			
OS-SRV-USG:terminated_at	-			
accessIPv4				
accessIPv6				
adminPass	TDc9EpBT3B9W			
config_drive				
created	2016-02-15T19:10:43Z			
flavor	m1.nano (42)			
hostId				
id	62c19691-d1c7-4d7b-a88e-9cc4d95d4f41			
image	cirros-0.3.5-x86_64-uec (b9d981eb-d21c-4ce2-9dbc- dd38f3d9015f)			
key_name	-			
locked	False			
metadata	{}			
name	my_vm			
os-extended-volumes:volumes_attached	[]			
progress	0			
security_groups	default			
status	BUILD			
tenant_id	d5660cb1e6934612a01b4fb2fb630725			
updated	2016-02-15T19:10:43Z			
user_id	8bb6e578cba24e7db9d3810633124525			
<hr/>				
<hr/>				
\$ openstack server list				
+-----+-----+-----+-----+-----+				
ID	Name	Status	Task State	Power State
Networks	Image			
+-----+-----+-----+-----+-----+				
62c19691-d1c7-4d7b-a88e-9cc4d95d4f41 my_vm ACTIVE - Running				
external=172.24.5.9, 2001:db8:10::9 cirros				
+-----+-----+-----+-----+-----+				
-----+-----				

In this example the port is created manually by the user and then used to boot an instance. Notice that:

- The port's data was visible in the DNS service as soon as it was created.
 - See [Performance considerations](#) for an explanation of the potential performance impact associated with this use case.

Following are the PTR records created for this example. Note that for IPv4, the value of ipv4_ptr_zone_prefix_size is 24. In the case of IPv6, the value of ipv6_ptr_zone_prefix_size is 116. For more details, see [Configuring OpenStack Networking for integration with an external DNS service](#):

See [Configuring OpenStack Networking for integration with an external DNS service](#) for detailed instructions on how to create the externally accessible network.

Use case 2: Floating IPs are published with associated port DNS attributes

In this use case, the address of a floating IP is published in the external DNS service in conjunction with the dns_name of its associated port and the dns_domain of the port's network. The steps to execute in this use case are the following:

1. Assign a valid domain name to the network's dns_domain attribute. This name must end with a period (.).
2. Boot an instance or alternatively, create a port specifying a valid value to its dns_name attribute. If the port is going to be used for an instance boot, the value assigned to dns_name must be equal to the hostname that the Compute service will assign to the instance. Otherwise, the boot will fail.
3. Create a floating IP and associate it to the port.

Following is an example of these steps:

```
$ neutron net-update 38c5e950-b450-4c30-83d4-ee181c28aad3 --dns_domain example.org.
Updated network: 38c5e950-b450-4c30-83d4-ee181c28aad3

$ neutron net-show 38c5e950-b450-4c30-83d4-ee181c28aad3
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| availability_zone_hints | |
| availability_zones | nova |
| dns_domain | example.org. |
| id | 38c5e950-b450-4c30-83d4-ee181c28aad3 |
| mtu | 1450 |
| name | private |
| port_security_enabled | True |
| router:external | False |
| shared | False |
| status | ACTIVE |
| subnets | 43414c53-62ae-49bc-aa6c-c9dd7705818a |
| | 5b9282a1-0be1-4ade-b478-7868ad2a16ff |
| tenant_id | d5660cb1e6934612a01b4fb2fb630725 |
+-----+-----+

$ openstack server create --image cirros --flavor 42 \
--nic net-id=38c5e950-b450-4c30-83d4-ee181c28aad3 my_vm
+-----+
| Field | Value |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | |
| OS-EXT-STS:power_state | 0 |
| OS-EXT-STS:task_state | scheduling |
| OS-EXT-STS:vm_state | building |
```

OS-SRV-USG:launched_at	-			
OS-SRV-USG:terminated_at	-			
accessIPv4				
accessIPv6				
adminPass	oTLQLR3Kezmt			
config_drive				
created	2016-02-15T19:27:34Z			
flavor	m1.nano (42)			
hostId				
id	43f328bb-b2d1-4cf1-a36f-3b2593397cb1			
image	cirros-0.3.5-x86_64-uec (b9d981eb-d21c-4ce2-9dbc- dd38f3d9015f)			
key_name	-			
locked	False			
metadata	{}			
name	my_vm			
os-extended-volumes:volumes_attached	[]			
progress	0			
security_groups	default			
status	BUILD			
tenant_id	d5660cb1e6934612a01b4fb2fb630725			
updated	2016-02-15T19:27:34Z			
user_id	8bb6e578cba24e7db9d3810633124525			
<hr/>				
<hr/>				
\$ openstack server list				
+-----+-----+-----+-----+-----+				
ID	Name	Status	Task State	Power State
Networks				Image Name
+-----+-----+-----+-----+-----+				
43f328bb-b2d1-4cf1-a36f-3b2593397cb1 my_vm ACTIVE - Running				
private=fda4:653e:71b0:0:f816:3eff:fe16:b5f2, 10.0.0.15 cirros				
+-----+-----+-----+-----+-----+				

```
$ neutron port-list --device_id 43f328bb-b2d1-4cf1-a36f-3b2593397cb1
+-----+-----+-----+
| id | name | mac_address | fixed_ips |
+-----+-----+-----+
| da0b1f75-c895-460f-9fc1-4d6ec84cf85f | fa:16:3e:16:b5:f2 | {"subnet_id": "5b9282a1-0be1-4ade-b478-7868ad2a16ff", "ip_address": "10.0.0.15"} | {"subnet_id": "43414c53-62ae-49bc-aa6c-c9dd7705818a", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe16:b5f2"} |
+-----+-----+-----+
$ neutron port-show da0b1f75-c895-460f-9fc1-4d6ec84cf85f
+-----+
| Field | Value |
+-----+
| admin_state_up | True |
| allowed_address_pairs | |
| binding:vnic_type | normal |
| device_id | 43f328bb-b2d1-4cf1-a36f-3b2593397cb1 |
| device_owner | compute:None |
| dns_assignment | {"hostname": "my-vm", "ip_address": "10.0.0.15", "fqdn": "my-vm.example.org."} |
| dns_assignment | {"hostname": "my-vm", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe16:b5f2", "fqdn": "my-vm.example.org."} |
| dns_name | my-vm |
| extra_dhcp_opts | |
| fixed_ips | {"subnet_id": "5b9282a1-0be1-4ade-b478-7868ad2a16ff", "ip_address": "10.0.0.15"} |
| fixed_ips | {"subnet_id": "43414c53-62ae-49bc-aa6c-c9dd7705818a", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe16:b5f2"} |
| id | da0b1f75-c895-460f-9fc1-4d6ec84cf85f |
| mac_address | fa:16:3e:16:b5:f2 |
| name | |
| network_id | 38c5e950-b450-4c30-83d4-ee181c28aad3 |
| port_security_enabled | True |
| security_groups | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635 |
| status | ACTIVE |
```

```
| tenant_id          | d5660cb1e6934612a01b4fb2fb630725 |
|                     |
+-----+-----+
| id               | type | name      | data   |
|                     |      |           |
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org. | ns1.devstack.org. malavall.us.ibm.com. 1455563783 3600 600 86400 3600 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org. | ns1.devstack.org. |
|                     |      |
+-----+-----+-----+
| id               | type | name      | data   |
|                     |      |           |
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org. | ns1.devstack.org. ns1.my-vm.example.org. 172.24.4.4 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org. | ns1.devstack.org. ns1.my-vm.example.org. 172.24.4.4 |
| 5ff53fd0-3746-48da-b9c9-77ed3004ec67 | A    | my-vm.example.org. | 172.24.4.4 |
```

In this example, notice that the data is published in the DNS service when the floating IP is associated to the port.

Following are the PTR records created for this example. Note that for IPv4, the value of

`ipv4_ptr_zone_prefix_size` is 24. For more details, see [Configuring OpenStack Networking for integration with an external DNS service](#):

\$ designate record-list 4.24.172.in-addr.arpa.				
id	type	name	data	
2dd0b894-25fa-4563-9d32-9f13bd67f329	NS	4.24.172.in-addr.arpa.	ns1.devstack.org.	
47b920f1-5eff-4dfa-9616-7cb5b7cb7ca6	SOA	4.24.172.in-addr.arpa.	ns1.devstack.org.	
admin.example.org. 1455564862 3600 600 86400 3600				
fb1edf42-abba-410c-8397-831f45fd0cd7	PTR	4.4.24.172.in-addr.arpa.	my-vm.example.org.	

Use case 3: Floating IPs are published in the external DNS service

In this use case, the user assigns `dns_name` and `dns_domain` attributes to a floating IP when it is created. The floating IP data becomes visible in the external DNS service as soon as it is created. The floating IP can be associated with a port on creation or later on. The following example shows a user booting an instance and then creating a floating IP associated to the port allocated for the instance:

\$ neutron net-show 38c5e950-b450-4c30-83d4-ee181c28aad3	
Field	Value
admin_state_up	True
availability_zone_hints	
availability_zones	nova
dns_domain	example.org.
id	38c5e950-b450-4c30-83d4-ee181c28aad3
mtu	1450
name	private
port_security_enabled	True
router:external	False
shared	False
status	ACTIVE
subnets	43414c53-62ae-49bc-aa6c-c9dd7705818a
	5b9282a1-0be1-4ade-b478-7868ad2a16ff
tenant_id	d5660cb1e6934612a01b4fb2fb630725

\$ openstack server create --image cirros --flavor 42 \ --nic net-id=38c5e950-b450-4c30-83d4-ee181c28aad3 my_vm	
Field	Value
OS-DCF:diskConfig	MANUAL

OS-EXT-AZ:availability_zone		□
		□
OS-EXT-STS:power_state	0	□
		□
OS-EXT-STS:task_state	scheduling	□
		□
OS-EXT-STS:vm_state	building	□
		□
OS-SRV-USG:launched_at	-	□
		□
OS-SRV-USG:terminated_at	-	□
		□
accessIPv4		□
		□
accessIPv6		□
		□
adminPass	HLXGznYqXM4J	□
		□
config_drive		□
		□
created	2016-02-15T19:42:44Z	□
		□
flavor	m1.nano (42)	□
		□
hostId		□
		□
id	71fb4ac8-eed8-4644-8113-0641962bb125	□
		□
image	cirros-0.3.5-x86_64-uec (b9d981eb-d21c-4ce2-9dbc-	□
dd38f3d9015f)		□
key_name	-	□
		□
locked	False	□
		□
metadata	{}	□
		□
name	my_vm	□
		□
os-extended-volumes:volumes_attached	[]	□
		□
progress	0	□
		□
security_groups	default	□
		□
status	BUILD	□
		□
tenant_id	d5660cb1e6934612a01b4fb2fb630725	□
		□
updated	2016-02-15T19:42:44Z	□
		□
user_id	8bb6e578cba24e7db9d3810633124525	□
		□
+	-----+-----+-----+-----+-----+-----+	
	-----+-----+-----+-----+-----+-----+	
\$ openstack server list	+-----+-----+-----+-----+-----+-----+	
+	-----+-----+-----+-----+-----+-----+	

```
| ID | Name | Status | Task State | Power State |  
+-----+-----+-----+-----+-----+  
| 71fb4ac8-eed8-4644-8113-0641962bb125 | my_vm | ACTIVE | - | Running |  
| private=fda4:653e:71b0:0:f816:3eff:fe24:8614, 10.0.0.16 | cirros |  
  
$ neutron port-list --device_id 71fb4ac8-eed8-4644-8113-0641962bb125  
+-----+-----+-----+-----+  
| id | name | mac_address | fixed_ips |  
+-----+-----+-----+-----+  
| 1e7033fb-8e9d-458b-89ed-8312cafcdcb | fa:16:3e:24:86:14 | {"subnet_id": "5b9282a1-0be1-4ade-b478-7868ad2a16ff", "ip_address": "10.0.0.16"} |  
| | | | {"subnet_id": "43414c53-62ae-49bc-aa6c-c9dd7705818a", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe24:8614"} |  
  
$ neutron port-show 1e7033fb-8e9d-458b-89ed-8312cafcdcb  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| admin_state_up | True |  
| allowed_address_pairs |  
| binding:vnic_type | normal |  
| device_id | 71fb4ac8-eed8-4644-8113-0641962bb125 |  
| device_owner | compute:None |  
| dns_assignment | {"hostname": "my-vm", "ip_address": "10.0.0.16", "fqdn": "my-vm.example.org."} |  
| | {"hostname": "my-vm", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe24:8614", "fqdn": "my-vm.example.org."} |  
| dns_name | my-vm |  
| extra_dhcp_opts |  
| fixed_ip | {"subnet_id": "5b9282a1-0be1-4ade-b478-7868ad2a16ff", "ip_address": "10.0.0.16"} |  
| | {"subnet_id": "43414c53-62ae-49bc-aa6c-c9dd7705818a", "ip_address": "fda4:653e:71b0:0:f816:3eff:fe24:8614"} |  
| id | 1e7033fb-8e9d-458b-89ed-8312cafcdcb |  
| mac_address | fa:16:3e:24:86:14 |  
| name |
```

```
| network_id          | 38c5e950-b450-4c30-83d4-ee181c28aad3
| port_security_enabled | True
| security_groups     | 1f0ddd73-7e3c-48bd-a64c-7ded4fe0e635
| status              | ACTIVE
| tenant_id           | d5660cb1e6934612a01b4fb2fb630725
+-----+
$ designate record-list example.org.
+-----+-----+-----+
| id                  | type | name      | data
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org. | ns1.devstack.org. malavall.
| us.ibm.com. 1455566486 3600 600 86400 3600 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org. | ns1.devstack.org.
+-----+-----+-----+
$ neutron floatingip-create 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a \
--dns_domain example.org. --dns_name my-floatingip
Created a new floatingip:
+-----+-----+
| Field        | Value
+-----+-----+
| dns_domain    | example.org.
| dns_name      | my-floatingip
| fixed_ip_address | 
| floating_ip_address | 172.24.4.5
| floating_network_id | 41fa3995-9e4a-4cd9-bb51-3e5424f2ff2a
| id            | 9f23a9c6-eceb-42eb-9f45-beb58c473728
| port_id       | 
| router_id     | 
| status         | DOWN
| tenant_id     | d5660cb1e6934612a01b4fb2fb630725
+-----+
$ designate record-list example.org.
+-----+-----+-----+
| id                  | type | name      | data
+-----+-----+-----+
| 10a36008-6ecf-47c3-b321-05652a929b04 | SOA  | example.org. | ns1.devstack.
| us.ibm.com. 1455566486 3600 600 86400 3600 |
| 56ca0b88-e343-4c98-8faa-19746e169baf | NS   | example.org. | ns1.devstack.
| org.
| 8884c56f-3ef5-446e-ae4d-8053cc8bc2b4 | A    | my-floatingip.example.org. | 172.24.4.5
+-----+
```

Note that in this use case:

- The `dns_name` and `dns_domain` attributes of a floating IP must be specified together on creation. They cannot be assigned to the floating IP separately.
- The `dns_name` and `dns_domain` of a floating IP have precedence, for purposes of being published in the external DNS service, over the `dns_name` of its associated port and the `dns_domain` of the port's network, whether they are specified or not. Only the `dns_name` and the `dns_domain` of the floating IP are published in the external DNS service.

Following are the PTR records created for this example. Note that for IPv4, the value of `ipv4_ptr_zone_prefix_size` is 24. For more details, see [Configuring OpenStack Networking for integration with an external DNS service](#):

\$ designate record-list 4.24.172.in-addr.arpa.			
id	type	name	data
2dd0b894-25fa-4563-9d32-9f13bd67f329	NS	4.24.172.in-addr.arpa.	ns1.devstack.org.
47b920f1-5eff-4dfa-9616-7cb5b7cb7ca6	SOA	4.24.172.in-addr.arpa.	ns1.devstack.org.
admin.example.org.	1455566487	3600 600 86400 3600	
589a0171-e77a-4ab6-ba6e-23114f2b9366	PTR	5.4.24.172.in-addr.arpa.	my-floatingip.example.org.

Performance considerations

Only for [Use case 1: Ports are published directly in the external DNS service](#), if the port binding extension is enabled in the Networking service, the Compute service will execute one additional port update operation when allocating the port for the instance during the boot process. This may have a noticeable adverse effect in the performance of the boot process that must be evaluated before adoption of this use case.

Configuring OpenStack Networking for integration with an external DNS service

The first step to configure the integration with an external DNS service is to enable the functionality described in [The Networking service internal DNS resolution](#). Once this is done, the user has to take the following steps and restart `neutron-server`.

1. Edit the `[default]` section of `/etc/neutron/neutron.conf` and specify the external DNS service driver to be used in parameter `external_dns_driver`. The valid options are defined in namespace `neutron.services.external_dns_drivers`. The following example shows how to set up the driver for the OpenStack DNS service:

```
external_dns_driver = designate
```

2. If the OpenStack DNS service is the target external DNS, the [designate] section of /etc/neutron/neutron.conf must define the following parameters:
 - `url`: the OpenStack DNS service public endpoint URL.
 - `allow_reverse_dns_lookup`: a boolean value specifying whether to enable or not the creation of reverse lookup (PTR) records.
 - `admin_auth_url`: the Identity service admin authorization endpoint url. This endpoint will be used by the Networking service to authenticate as an admin user to create and update reverse lookup (PTR) zones.
 - `admin_username`: the admin user to be used by the Networking service to create and update reverse lookup (PTR) zones.
 - `admin_password`: the password of the admin user to be used by Networking service to create and update reverse lookup (PTR) zones.
 - `admin_tenant_name`: the project of the admin user to be used by the Networking service to create and update reverse lookup (PTR) zones.
 - `ipv4_ptr_zone_prefix_size`: the size in bits of the prefix for the IPv4 reverse lookup (PTR) zones.
 - `ipv6_ptr_zone_prefix_size`: the size in bits of the prefix for the IPv6 reverse lookup (PTR) zones.
 - `insecure`: Disable SSL certificate validation. By default, certificates are validated.
 - `cafile`: Path to a valid Certificate Authority (CA) certificate.

The following is an example:

```
[designate]
url = http://55.114.111.93:9001/v2
admin_auth_url = http://55.114.111.93:35357/v2.0
admin_username = neutron
admin_password = x5G90074
admin_tenant_name = service
allow_reverse_dns_lookup = True
ipv4_ptr_zone_prefix_size = 24
ipv6_ptr_zone_prefix_size = 116
cafile = /etc/ssl/certs/my_ca_cert
```

Configuration of the externally accessible network for use case 1

In *Use case 1: Ports are published directly in the external DNS service*, the externally accessible network must meet the following requirements:

- The network cannot have attribute `router:external` set to True.
- The network type can be FLAT, VLAN, GRE, VXLAN or GENEVE.
- For network types VLAN, GRE, VXLAN or GENEVE, the segmentation ID must be outside the ranges assigned to tenant networks.

Name resolution for instances

The Networking service offers several methods to configure name resolution (DNS) for instances. Most deployments should implement case 1 or 2. Case 3 requires security considerations to prevent leaking internal DNS information to instances.

Case 1: Each virtual network uses unique DNS resolver(s)

In this case, the DHCP agent offers one or more unique DNS resolvers to instances via DHCP on each virtual network. You can configure a DNS resolver when creating or updating a subnet. To configure more than one DNS resolver, use a comma between each value.

- Configure a DNS resolver when creating a subnet.

```
$ neutron subnet-create --dns-nameserver DNS_RESOLVER
```

Replace `DNS_RESOLVER` with the IP address of a DNS resolver reachable from the virtual network. For example:

```
$ neutron subnet-create --dns-nameserver 8.8.8.8,8.8.4.4
```

Note: This command requires other options outside the scope of this content.

- Configure a DNS resolver on an existing subnet.

```
$ neutron subnet-update --dns-nameserver DNS_RESOLVER SUBNET_ID_OR_NAME
```

Replace `DNS_RESOLVER` with the IP address of a DNS resolver reachable from the virtual network and `SUBNET_ID_OR_NAME` with the UUID or name of the subnet. For example, using the `selfservice` subnet:

```
$ neutron subnet-update --dns-nameserver 8.8.8.8,8.8.4.4 selfservice
```

Case 2: All virtual networks use same DNS resolver(s)

In this case, the DHCP agent offers the same DNS resolver(s) to instances via DHCP on all virtual networks.

- In the `dhcp_agent.ini` file, configure one or more DNS resolvers. To configure more than one DNS resolver, use a comma between each value.

```
[DEFAULT]
dnsmasq_dns_servers = DNS_RESOLVER
```

Replace `DNS_RESOLVER` with the IP address of a DNS resolver reachable from all virtual networks. For example:

```
[DEFAULT]
dnsmasq_dns_servers = 8.8.8.8, 8.8.4.4
```

Note: You must configure this option for all eligible DHCP agents and restart them to activate the values.

Case 3: All virtual networks use DNS resolver(s) on the host

In this case, the DHCP agent offers the DNS resolver(s) in the `resolv.conf` file on the host running the DHCP agent via DHCP to instances on all virtual networks.

- In the `dhcp_agent.ini` file, enable advertisement of the DNS resolver(s) on the host.

[DEFAULT]

```
dnsmasq_local_resolv = True
```

Note: You must configure this option for all eligible DHCP agents and restart them to activate the values.

Distributed Virtual Routing with VRRP

[Open vSwitch: High availability using DVR](#) supports augmentation using Virtual Router Redundancy Protocol (VRRP). Using this configuration, virtual routers support both the `--distributed` and `--ha` options.

Similar to legacy HA routers, DVR/SNAT HA routers provide a quick fail over of the SNAT service to a backup DVR/SNAT router on an l3-agent running on a different node.

SNAT high availability is implemented in a manner similar to the [Linux bridge: High availability using VRRP](#) and [Open vSwitch: High availability using VRRP](#) examples where `keepalived` uses VRRP to provide quick failover of SNAT services.

During normal operation, the master router periodically transmits *heartbeat* packets over a hidden project network that connects all HA routers for a particular project.

If the DVR/SNAT backup router stops receiving these packets, it assumes failure of the master DVR/SNAT router and promotes itself to master router by configuring IP addresses on the interfaces in the `snat` namespace. In environments with more than one backup router, the rules of VRRP are followed to select a new master router.

Warning: There is a known bug with `keepalived` v1.2.15 and earlier which can cause packet loss when `max_l3_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to `keepalived` v1.2.16 or greater when using this feature.

Note: Experimental feature or incomplete documentation.

Configuration example

The basic deployment model consists of one controller node, two or more network nodes, and multiple compute nodes.

Controller node configuration

1. Add the following to /etc/neutron/neutron.conf:

```
[DEFAULT]
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
router_distributed = True
l3_ha = True
l3_ha_net_cidr = 169.254.192.0/18
max_l3_agents_per_router = 3
```

When the `router_distributed = True` flag is configured, routers created by all users are distributed. Without it, only privileged users can create distributed routers by using `--distributed True`.

Similarly, when the `l3_ha = True` flag is configured, routers created by all users default to HA.

It follows that with these two flags set to True in the configuration file, routers created by all users will default to distributed HA routers (DVR HA).

The same can explicitly be accomplished by a user with administrative credentials setting the flags in the `neutron router-create` command:

```
$ neutron router-create name-of-router --distributed=True --ha=True
```

Note: The `max_l3_agents_per_router` determine the number of backup DVR/SNAT routers which will be instantiated.

2. Add the following to /etc/neutron/plugins/ml2/ml2_conf.ini:

```
[ml2]
type_drivers = flat,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
extension_drivers = port_security

[ml2_type_flat]
flat_networks = external

[ml2_type_vxlan]
vni_ranges = MIN_VXLAN_ID:MAX_VXLAN_ID
```

Replace `MIN_VXLAN_ID` and `MAX_VXLAN_ID` with VXLAN ID minimum and maximum values suitable for your environment.

Note: The first value in the `tenant_network_types` option becomes the default project network type when a regular user creates a network.

Network nodes

1. Configure the Open vSwitch agent. Add the following to `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ovs]
local_ip = TUNNEL_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
enable_distributed_routing = True
tunnel_types = vxlan
l2_population = True
```

Replace `TUNNEL_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN project networks.

2. Configure the L3 agent. Add the following to `/etc/neutron/l3_agent.ini`:

```
[DEFAULT]
ha_vrrp_auth_password = password
interface_driver = openvswitch
external_network_bridge =
agent_mode = dvr_snat
```

Note: The `external_network_bridge` option intentionally contains no value.

Compute nodes

1. Configure the Open vSwitch agent. Add the following to `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ovs]
local_ip = TUNNEL_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
enable_distributed_routing = True
tunnel_types = vxlan
l2_population = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
↪OVSHybridIptablesFirewallDriver
```

2. Configure the L3 agent. Add the following to `/etc/neutron/l3_agent.ini`:

```
[DEFAULT]
interface_driver = openvswitch
external_network_bridge =
agent_mode = dvr
```

Replace TUNNEL_INTERFACE_IP_ADDRESS with the IP address of the interface that handles VXLAN project networks.

Keepalived VRRP health check

The health of your keepalived instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the `l3_agent.ini` file:

```
ha_vrrp_health_check_interval = 30
```

Where `ha_vrrp_health_check_interval` indicates how often in seconds the health check should run. The default value is 0, which indicates that the check should not run at all.

Known limitations

- Migrating a router from distributed only, HA only, or legacy to distributed HA is not supported at this time. The router must be created as distributed HA. The reverse direction is also not supported. You cannot reconfigure a distributed HA router to be only distributed, only HA, or legacy.
- There are certain scenarios where l2pop and distributed HA routers do not interact in an expected manner. These situations are the same that affect HA only routers and l2pop.

IPAM configuration

Note: Experimental feature or incomplete documentation.

Starting with the Liberty release, OpenStack Networking includes a pluggable interface for the IP Address Management (IPAM) function. This interface creates a driver framework for the allocation and de-allocation of subnets and IP addresses, enabling the integration of alternate IPAM implementations or third-party IP Address Management systems.

The basics

In Liberty and Mitaka, the IPAM implementation within OpenStack Networking provided a pluggable and non-pluggable flavor. As of Newton, the non-pluggable flavor is no longer available. Instead, it is completely replaced with a reference driver implementation of the pluggable framework. All data will be automatically migrated during the upgrade process, unless you have previously configured a pluggable IPAM driver. In that case, no migration is necessary.

To configure a driver other than the reference driver, specify it in the `neutron.conf` file. Do this after the migration is complete. There is no need to specify any value if you wish to use the reference driver.

```
ipam_driver = ipam-driver-name
```

There is no need to specify any value if you wish to use the reference driver, though specifying `internal` will explicitly choose the reference driver. The documentation for any alternate drivers will include the value to use when specifying that driver.

Known limitations

- The driver interface is designed to allow separate drivers for each subnet pool. However, the current implementation allows only a single IPAM driver system-wide.
- Third-party drivers must provide their own migration mechanisms to convert existing OpenStack installations to their IPAM.

IPv6

This section describes the following items:

- How to enable dual-stack (IPv4 and IPv6 enabled) instances.
- How those instances receive an IPv6 address.
- How those instances communicate across a router to other subnets or the internet.
- How those instances interact with other OpenStack services.

Enabling a dual-stack network in OpenStack Networking simply requires creating a subnet with the `ip_version` field set to 6, then the IPv6 attributes (`ipv6_ra_mode` and `ipv6_address_mode`) set. The `ipv6_ra_mode` and `ipv6_address_mode` will be described in detail in the next section. Finally, the subnets `cidr` needs to be provided.

This section does not include the following items:

- Single stack IPv6 project networking
- OpenStack control communication between servers and services over an IPv6 network.
- Connection to the OpenStack APIs via an IPv6 transport network
- IPv6 multicast
- IPv6 support in conjunction with any out of tree routers, switches, services or agents whether in physical or virtual form factors.

Neutron subnets and the IPv6 API attributes

As of Juno, the OpenStack Networking service (neutron) provides two new attributes to the subnet object, which allows users of the API to configure IPv6 subnets.

There are two IPv6 attributes:

- `ipv6_ra_mode`
- `ipv6_address_mode`

These attributes can be set to the following values:

- slaac
- dhcipv6-stateful
- dhcipv6-stateless

The attributes can also be left unset.

IPv6 addressing

The `ipv6_address_mode` attribute is used to control how addressing is handled by OpenStack. There are a number of different ways that guest instances can obtain an IPv6 address, and this attribute exposes these choices to users of the Networking API.

Router advertisements

The `ipv6_ra_mode` attribute is used to control router advertisements for a subnet.

The IPv6 Protocol uses Internet Control Message Protocol packets (ICMPv6) as a way to distribute information about networking. ICMPv6 packets with the type flag set to 134 are called “Router Advertisement” packets, which contain information about the router and the route that can be used by guest instances to send network traffic.

The `ipv6_ra_mode` is used to specify if the Networking service should generate Router Advertisement packets for a subnet.

ipv6_ra_mode and ipv6_address_mode combinations

ipv6 ra mode	ipv6 address mode	radvd A,M,O	External Router A,M,O	Description
N/S	N/S	Off	Not Defined	Backwards compatibility with pre-Juno IPv6 behavior.
N/S	slaac	Off	1,0,0	Guest instance obtains IPv6 address from non-OpenStack router using SLAAC.
N/S	dhcpv6-stateful	Off	0,1,1	Not currently implemented in the reference implementation.
N/S	dhcpv6-stateless	Off	1,0,1	Not currently implemented in the reference implementation.
slaac	N/S	1,0,0	Off	Not currently implemented in the reference implementation.
dhcpv6-stateful	N/S	0,1,1	Off	Not currently implemented in the reference implementation.
dhcpv6-stateless	N/S	1,0,1	Off	Not currently implemented in the reference implementation.
slaac	slaac	1,0,0	Off	Guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC.
dhcpv6-stateful	dhcpv6-stateful	0,1,1	Off	Guest instance obtains IPv6 address from dnsmasq using DHCPv6 stateful and optional info from dnsmasq using DHCPv6.
dhcpv6-stateless	dhcpv6-stateless	1,0,1	Off	Guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6.
slaac	dhcpv6-stateful			<i>Invalid combination.</i>
slaac	dhcpv6-stateless			<i>Invalid combination.</i>
dhcpv6-stateful	slaac			<i>Invalid combination.</i>
dhcpv6-stateful	dhcpv6-stateless			<i>Invalid combination.</i>
dhcpv6-stateless	slaac			<i>Invalid combination.</i>
dhcpv6-stateless	dhcpv6-stateful			<i>Invalid combination.</i>

Project network considerations

Dataplane

Both the Linux bridge and the Open vSwitch dataplane modules support forwarding IPv6 packets amongst the guests and router ports. Similar to IPv4, there is no special configuration or setup required to enable the dataplane to properly forward packets from the source to the destination using IPv6. Note that these dataplanes will forward Link-local Address (LLA) packets between hosts on the same network just fine without any participation or setup by OpenStack components after the ports are all connected and MAC addresses learned.

Addresses for subnets

There are three methods currently implemented for a subnet to get its cidr in OpenStack:

1. Direct assignment during subnet creation via command line or Horizon
2. Referencing a subnet pool during subnet creation
3. Using a Prefix Delegation (PD) client to request a prefix for a subnet from a PD server

In the future, additional techniques could be used to allocate subnets to projects, for example, use of an external IPAM module.

Address modes for ports

Note: An external DHCPv6 server in theory could override the full address OpenStack assigns based on the EUI-64 address, but that would not be wise as it would not be consistent through the system.

IPv6 supports three different addressing schemes for address configuration and for providing optional network information.

Stateless Address Auto Configuration (SLAAC) Address configuration using Router Advertisement (RA).

DHCPv6-stateless Address configuration using RA and optional information using DHCPv6.

DHCPv6-stateful Address configuration and optional information using DHCPv6.

OpenStack can be setup such that OpenStack Networking directly provides RA, DHCP relay and DHCPv6 address and optional information for their networks or this can be delegated to external routers and services based on the drivers that are in use. There are two neutron subnet attributes - `ipv6_ra_mode` and `ipv6_address_mode` – that determine how IPv6 addressing and network information is provided to project instances:

- `ipv6_ra_mode`: Determines who sends RA.
- `ipv6_address_mode`: Determines how instances obtain IPv6 address, default gateway, or optional information.

For the above two attributes to be effective, `enable_dhcp` of the subnet object must be set to True.

Using SLAAC for addressing

When using SLAAC, the currently supported combinations for `ipv6_ra_mode` and `ipv6_address_mode` are as follows.

<code>ipv6_ra_mode</code>	<code>ipv6_address_mode</code>	Result
Not specified.	SLAAC	Addresses are assigned using EUI-64, and an external router will be used for routing.
SLAAC	SLAAC	Address are assigned using EUI-64, and OpenStack Networking provides routing.

Setting `ipv6_ra_mode` to `slaac` will result in OpenStack Networking routers being configured to send RA packets, when they are created. This results in the following values set for the address configuration flags in the RA messages:

- Auto Configuration Flag = 1
- Managed Configuration Flag = 0
- Other Configuration Flag = 0

New or existing neutron networks that contain a SLAAC enabled IPv6 subnet will result in all neutron ports attached to the network receiving IPv6 addresses. This is because when RA broadcast messages are sent out on a neutron network, they are received by all IPv6 capable ports on the network, and each port will then configure an IPv6 address based on the information contained in the RA packet. In some cases, an IPv6 SLAAC address will be added to a port, in addition to other IPv4 and IPv6 addresses that the port already has been assigned.

DHCPv6

For DHCPv6, the currently supported combinations are as follows:

ipv6_ra_mode	ipv6_address_mode	result
DHCPv6-stateless	DHCPv6-stateless	Addresses are assigned through RAs (see SLAAC above) and optional information is delivered through DHCPv6.
DHCPv6-stateful	DHCPv6-stateful	Addresses and optional information are assigned using DHCPv6.

Setting DHCPv6-stateless for `ipv6_ra_mode` configures the neutron router with radvd agent to send RAs. The list below captures the values set for the address configuration flags in the RA packet in this scenario. Similarly, setting DHCPv6-stateless for `ipv6_address_mode` configures neutron DHCP implementation to provide the additional network information.

- Auto Configuration Flag = 1
- Managed Configuration Flag = 0
- Other Configuration Flag = 1

Setting DHCPv6-stateful for `ipv6_ra_mode` configures the neutron router with radvd agent to send RAs. The list below captures the values set for the address configuration flags in the RA packet in this scenario. Similarly, setting DHCPv6-stateful for `ipv6_address_mode` configures neutron DHCP implementation to provide addresses and additional network information through DHCPv6.

- Auto Configuration Flag = 0
- Managed Configuration Flag = 1
- Other Configuration Flag = 1

Router support

The behavior of the neutron router for IPv6 is different than for IPv4 in a few ways.

Internal router ports, that act as default gateway ports for a network, will share a common port for all IPv6 subnets associated with the network. This implies that there will be an IPv6 internal router interface with multiple IPv6 addresses from each of the IPv6 subnets associated with the network and a separate IPv4 internal router interface for the IPv4 subnet. On the other hand, external router ports are allowed to have a dual-stack configuration with both an IPv4 and an IPv6 address assigned to them.

Neutron project networks that are assigned Global Unicast Address (GUA) prefixes and addresses don't require NAT on the neutron router external gateway port to access the outside world. As a consequence of the lack of NAT the external router port doesn't require a GUA to send and receive to the external networks. This implies

a GUA IPv6 subnet prefix is not necessarily needed for the neutron external network. By default, a IPv6 LLA associated with the external gateway port can be used for routing purposes. To handle this scenario, the implementation of router-gateway-set API in neutron has been modified so that an IPv6 subnet is not required for the external network that is associated with the neutron router. The LLA address of the upstream router can be learned in two ways.

1. In the absence of an upstream RA support, `ipv6_gateway` flag can be set with the external router gateway LLA in the neutron L3 agent configuration file. This also requires that no subnet is associated with that port.
2. The upstream router can send an RA and the neutron router will automatically learn the next-hop LLA, provided again that no subnet is assigned and the `ipv6_gateway` flag is not set.

Effectively the `ipv6_gateway` flag takes precedence over an RA that is received from the upstream router. If it is desired to use a GUA next hop that is accomplished by allocating a subnet to the external router port and assigning the upstream routers GUA address as the gateway for the subnet.

Note: It should be possible for projects to communicate with each other on an isolated network (a network without a router port) using LLA with little to no participation on the part of OpenStack. The authors of this section have not proven that to be true for all scenarios.

Note: When using the neutron L3 agent in a configuration where it is auto-configuring an IPv6 address via SLAAC, and the agent is learning its default IPv6 route from the ICMPv6 Router Advertisement, it may be necessary to set the `net.ipv6.conf.<physical_interface>.accept_ra` sysctl to the value 2 in order for routing to function correctly. For a more detailed description, please see the [bug](#).

Neutron's Distributed Router feature and IPv6

IPv6 does work when the Distributed Virtual Router functionality is enabled, but all ingress/egress traffic is via the centralized router (hence, not distributed). More work is required to fully enable this functionality.

Advanced services

VPNaas

VPNaas supports IPv6, but support in Kilo and prior releases will have some bugs that may limit how it can be used. More thorough and complete testing and bug fixing is being done as part of the Liberty release. IPv6-based VPN-as-a-Service is configured similar to the IPv4 configuration. Either or both the `peer_address` and the `peer_cidr` can specified as an IPv6 address. The choice of addressing modes and router modes described above should not impact support.

LBaaS

TODO

FWaaS

FWaaS allows creation of IPv6 based rules.

NAT & Floating IPs

At the current time OpenStack Networking does not provide any facility to support any flavor of NAT with IPv6. Unlike IPv4 there is no current embedded support for floating IPs with IPv6. It is assumed that the IPv6 addressing amongst the projects is using GUAs with no overlap across the projects.

Security considerations

Configuring interfaces of the guest

OpenStack currently doesn't support the privacy extensions defined by RFC 4941. The interface identifier and DUID used must be directly derived from the MAC as described in RFC 2373. The compute hosts must not be setup to utilize the privacy extensions when generating their interface identifier.

There is no provisions for an IPv6-based metadata service similar to what is provided for IPv4. In the case of dual stacked guests though it is always possible to use the IPv4 metadata service instead.

Unlike IPv4 the MTU of a given network can be conveyed in the RA messages sent by the router as well as in the DHCP messages.

OpenStack control & management network considerations

As of the Kilo release, considerable effort has gone in to ensuring the project network can handle dual stack IPv6 and IPv4 transport across the variety of configurations described above. OpenStack control network can be run in a dual stack configuration and OpenStack API endpoints can be accessed via an IPv6 network. At this time, Open vSwitch (OVS) tunnel types - STT, VXLAN, GRE, support both IPv4 and IPv6 endpoints.

Prefix delegation

From the Liberty release onwards, OpenStack Networking supports IPv6 prefix delegation. This section describes the configuration and workflow steps necessary to use IPv6 prefix delegation to provide automatic allocation of subnet CIDRs. This allows you as the OpenStack administrator to rely on an external (to the OpenStack Networking service) DHCPv6 server to manage your project network prefixes.

Note: Prefix delegation became available in the Liberty release, it is not available in the Kilo release. HA and DVR routers are not currently supported by this feature.

Configuring OpenStack Networking for prefix delegation

To enable prefix delegation, edit the `/etc/neutron/neutron.conf` file. If you are running OpenStack Liberty, make the following change:

```
default_ipv6_subnet_pool = prefix_delegation
```

Otherwise if you are running OpenStack Mitaka, make this change:

```
ipv6_pd_enabled = True
```

Note: If you are not using the default dibbler-based driver for prefix delegation, then you also need to set the driver in /etc/neutron/neutron.conf:

```
pd_dhcp_driver = <class path to driver>
```

Drivers other than the default one may require extra configuration, please refer to [Extra configuration](#)

This tells OpenStack Networking to use the prefix delegation mechanism for subnet allocation when the user does not provide a CIDR or subnet pool id when creating a subnet.

Requirements

To use this feature, you need a prefix delegation capable DHCPv6 server that is reachable from your OpenStack Networking node(s). This could be software running on the OpenStack Networking node(s) or elsewhere, or a physical router. For the purposes of this guide we are using the open-source DHCPv6 server, Dibbler. Dibbler is available in many Linux package managers, or from source at [tomaszmrugalski/dibbler](#).

When using the reference implementation of the OpenStack Networking prefix delegation driver, Dibbler must also be installed on your OpenStack Networking node(s) to serve as a DHCPv6 client. Version 1.0.1 or higher is required.

This guide assumes that you are running a Dibbler server on the network node where the external network bridge exists. If you already have a prefix delegation capable DHCPv6 server in place, then you can skip the following section.

Configuring the Dibbler server

After installing Dibbler, edit the /etc/dibbler/server.conf file:

```
script "/var/lib/dibbler/pd-server.sh"

iface "br-ex" {
    pd-class {
        pd-pool 2001:db8:2222::/48
        pd-length 64
    }
}
```

The options used in the configuration file above are:

- **script** Points to a script to be run when a prefix is delegated or released. This is only needed if you want instances on your subnets to have external network access. More on this below.
- **iface** The name of the network interface on which to listen for prefix delegation messages.

- pd-pool The larger prefix from which you want your delegated prefixes to come. The example given is sufficient if you do not need external network access, otherwise a unique globally routable prefix is necessary.
- pd-length The length that delegated prefixes will be. This must be 64 to work with the current OpenStack Networking reference implementation.

To provide external network access to your instances, your Dibbler server also needs to create new routes for each delegated prefix. This is done using the script file named in the config file above. Edit the `/var/lib/dibbler/pd-server.sh` file:

```
if [ "$PREFIX1" != "" ]; then
    if [ "$1" == "add" ]; then
        sudo ip -6 route add ${PREFIX1}/64 via $REMOTE_ADDR dev $IFACE
    fi
    if [ "$1" == "delete" ]; then
        sudo ip -6 route del ${PREFIX1}/64 via $REMOTE_ADDR dev $IFACE
    fi
fi
```

The variables used in the script file above are:

- `$PREFIX1` The prefix being added/deleted by the Dibbler server.
- `$1` The operation being performed.
- `$REMOTE_ADDR` The IP address of the requesting Dibbler client.
- `$IFACE` The network interface upon which the request was received.

The above is all you need in this scenario, but more information on installing, configuring, and running Dibbler is available in the Dibbler user guide, at [Dibbler – a portable DHCPv6](#).

To start your Dibbler server, run:

```
# dibbler-server run
```

Or to run in headless mode:

```
# dibbler-server start
```

When using DevStack, it is important to start your server after the `stack.sh` script has finished to ensure that the required network interfaces have been created.

User workflow

First, create a network and IPv6 subnet:

```
$ openstack network create ipv6-pd
+-----+-----+
| Field          | Value           |
+-----+-----+
| admin_state_up | UP              |
| availability_zone_hints |           |
| availability_zones |           |
| created_at     | 2017-01-25T19:26:01Z |
| description     |           |
```

```

| headers           | |
| id               | 4b782725-6abe-4a2d-b061-763def1bb029 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| mtu              | 1450 |
| name              | ipv6-pd |
| port_security_enabled | True |
| project_id        | 61b7eba037fd41f29cfba757c010faff |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 46 |
| revision_number   | 3 |
| router:external    | Internal |
| shared             | False |
| status              | ACTIVE |
| subnets            | |
| tags               | [] |
| updated_at         | 2017-01-25T19:26:01Z |
+-----+-----+-----+
$ openstack subnet create --ip-version 6 --ipv6-ra-mode slaac \
--ipv6-address-mode slaac --use-default-subnet-pool \
--network ipv6-pd ipv6-pd-1
+-----+-----+-----+
| Field          | Value |
+-----+-----+-----+
| allocation_pools | ::2-::ffff:ffff:ffff:ffff |
| cidr            | ::/64 |
| created_at      | 2017-01-25T19:31:53Z |
| description      | |
| dns_nameservers | |
| enable_dhcp     | True |
| gateway_ip       | ::1 |
| headers          | |
| host_routes      | |
| id               | 1319510d-c92c-4532-bf5d-8bcf3da761a1 |
| ip_version       | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode     | slaac |
| name              | ipv6-pd-1 |
| network_id        | 4b782725-6abe-4a2d-b061-763def1bb029 |
| project_id        | 61b7eba037fd41f29cfba757c010faff |
| revision_number   | 2 |
| service_types     | |
| subnetpool_id     | prefix_delegation |
| updated_at         | 2017-01-25T19:31:53Z |
| use_default_subnetpool | True |
+-----+-----+-----+

```

The subnet is initially created with a temporary CIDR before one can be assigned by prefix delegation. Any number of subnets with this temporary CIDR can exist without raising an overlap error. The subnetpool_id is automatically set to `prefix_delegation`.

To trigger the prefix delegation process, create a router interface between this subnet and a router with an active interface on the external network:

```
$ openstack router add subnet router1 ipv6-pd-1
```

The prefix delegation mechanism then sends a request via the external network to your prefix delegation server, which replies with the delegated prefix. The subnet is then updated with the new prefix, including issuing new IP addresses to all ports:

```
$ openstack subnet show ipv6-pd-1
+-----+-----+
| Field      | Value
+-----+-----+
| allocation_pools | 2001:db8:2222:6977::2-2001:db8:2222:
|                   | 6977:ffff:ffff:ffff:ffff
| cidr        | 2001:db8:2222:6977::/64
| created_at   | 2017-01-25T19:31:53Z
| description   |
| dns_nameservers |
| enable_dhcp    | True
| gateway_ip     | 2001:db8:2222:6977::1
| host_routes    |
| id            | 1319510d-c92c-4532-bf5d-8bcf3da761a1
| ip_version     | 6
| ipv6_address_mode | slaac
| ipv6_ra_mode    | slaac
| name           | ipv6-pd-1
| network_id      | 4b782725-6abe-4a2d-b061-763def1bb029
| project_id      | 61b7eba037fd41f29cfba757c010faff
| revision_number | 4
| service_types   |
| subnetpool_id   | prefix_delegation
| updated_at       | 2017-01-25T19:35:26Z
+-----+
```

If the prefix delegation server is configured to delegate globally routable prefixes and setup routes, then any instance with a port on this subnet should now have external network access.

Deleting the router interface causes the subnet to be reverted to the temporary CIDR, and all ports have their IPs updated. Prefix leases are released and renewed automatically as necessary.

References

The following link provides a great step by step tutorial on setting up IPv6 with OpenStack: [Tenant IPV6 deployment in OpenStack Kilo release](#).

Extra configuration

Neutron dhcipv6_pd_agent

To enable the driver for the dhcipv6_pd_agent, set pd_dhcp_driver to this in /etc/neutron/neutron.conf:

```
pd_dhcp_driver = neutron_pd_agent
```

To allow the neutron-pd-agent to communicate with prefix delegation servers, you must set which network interface to use for external communication. In DevStack the default for this is br-ex:

```
pd_interface = br-ex
```

Once you have stacked run the command below to start the neutron-pd-agent:

```
neutron-pd-agent --config-file /etc/neutron/neutron.conf
```

Load Balancer as a Service (LBaaS)

The Networking service offers a load balancer feature called “LBaaS v2” through the `neutron-lbaas` service plug-in.

LBaaS v2 adds the concept of listeners to the LBaaS v1 load balancers. LBaaS v2 allows you to configure multiple listener ports on a single load balancer IP address.

There are two reference implementations of LBaaS v2. The one is an agent based implementation with HAProxy. The agents handle the HAProxy configuration and manage the HAProxy daemon. Another LBaaS v2 implementation, [Octavia](#), has a separate API and separate worker processes that build load balancers within virtual machines on hypervisors that are managed by the Compute service. You do not need an agent for Octavia.

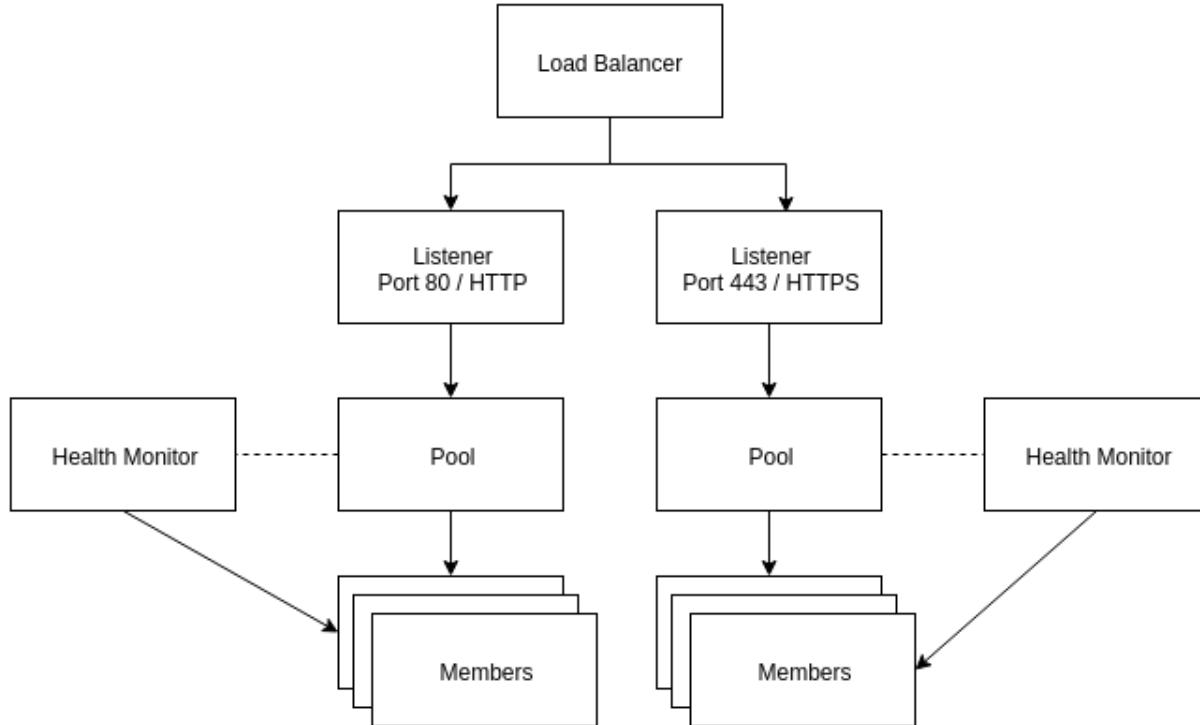
Note: LBaaS v1 was removed in the Newton release. These links provide more details about how LBaaS v1 works and how to configure it:

- [Load-Balancer-as-a-Service \(LBaaS\) overview](#)
 - [Basic Load-Balancer-as-a-Service operations](#)
-

Warning: Currently, no migration path exists between v1 and v2 load balancers. If you choose to switch from v1 to v2, you must recreate all load balancers, pools, and health monitors.

LBaaS v2 Concepts

LBaaS v2 has several new concepts to understand:



Load balancer The load balancer occupies a neutron network port and has an IP address assigned from a subnet.

Listener Load balancers can listen for requests on multiple ports. Each one of those ports is specified by a listener.

Pool A pool holds a list of members that serve content through the load balancer.

Member Members are servers that serve traffic behind a load balancer. Each member is specified by the IP address and port that it uses to serve traffic.

Health monitor Members may go offline from time to time and health monitors divert traffic away from members that are not responding properly. Health monitors are associated with pools.

LBaaS v2 has multiple implementations via different service plug-ins. The two most common implementations use either an agent or the Octavia services. Both implementations use the [LBaaS v2 API](#).

Configurations

Configuring LBaaS v2 with an agent

- Add the LBaaS v2 service plug-in to the `service_plugins` configuration directive in `/etc/neutron/neutron.conf`. The plug-in list is comma-separated:

```

service_plugins = [existing service plugins],neutron_lbaas.services.loadbalancer.
  ↪plugin.LoadBalancerPluginv2
  
```

- Add the LBaaS v2 service provider to the `service_provider` configuration directive within the `[service_providers]` section in `/etc/neutron/neutron_lbaas.conf`:

```

service_provider = LOADBALANCERV2:Haproxy:neutron_lbaas.drivers.haproxy.plugin_driver.
  ↪HaproxyOnHostPluginDriver:default
  
```

If you have existing service providers for other networking service plug-ins, such as VPNaas or FWaaS, add the `service_provider` line shown above in the `[service_providers]` section as a separate line. These configuration directives are repeatable and are not comma-separated.

3. Select the driver that manages virtual interfaces in `/etc/neutron/lbaas_agent.ini`:

```
[DEFAULT]
interface_driver = INTERFACE_DRIVER
```

Replace `INTERFACE_DRIVER` with the interface driver that the layer-2 agent in your environment uses. For example, `openvswitch` for Open vSwitch or `linuxbridge` for Linux bridge.

4. Run the `neutron-lbaas` database migration:

```
neutron-db-manage --subproject neutron-lbaas upgrade head
```

5. If you have deployed LBaaS v1, **stop the LBaaS v1 agent now**. The v1 and v2 agents **cannot** run simultaneously.

6. Start the LBaaS v2 agent:

```
neutron-lbaasv2-agent \
--config-file /etc/neutron/neutron.conf \
--config-file /etc/neutron/lbaas_agent.ini
```

7. Restart the Network service to activate the new configuration. You are now ready to create load balancers with the LBaaS v2 agent.

Configuring LBaaS v2 with Octavia

Octavia provides additional capabilities for load balancers, including using a compute driver to build instances that operate as load balancers. The [Hands on Lab - Install and Configure OpenStack Octavia](#) session at the OpenStack Summit in Tokyo provides an overview of Octavia.

The DevStack documentation offers a [simple method to deploy Octavia](#) and test the service with redundant load balancer instances. If you already have Octavia installed and configured within your environment, you can configure the Network service to use Octavia:

1. Add the LBaaS v2 service plug-in to the `service_plugins` configuration directive in `/etc/neutron/neutron.conf`. The plug-in list is comma-separated:

```
service_plugins = [existing service plugins],neutron_lbaas.services.loadbalancer.
↳plugin.LoadBalancerPluginv2
```

2. Add the Octavia service provider to the `service_provider` configuration directive within the `[service_providers]` section in `/etc/neutron/neutron_lbaas.conf`:

```
service_provider = LOADBALANCERV2:Octavia:neutron_lbaas.drivers.octavia.driver.
↳OctaviaDriver:default
```

Ensure that the LBaaS v1 and v2 service providers are removed from the `[service_providers]` section. They are not used with Octavia. **Verify that all LBaaS agents are stopped**.

3. Restart the Network service to activate the new configuration. You are now ready to create and manage load balancers with Octavia.

Add LBaaS panels to Dashboard

The Dashboard panels for managing LBaaS v2 are available starting with the Mitaka release.

1. Clone the [neutron-lbaas-dashboard repository](#) and check out the release branch that matches the installed version of Dashboard:

```
$ git clone https://git.openstack.org/openstack/neutron-lbaas-dashboard  
$ cd neutron-lbaas-dashboard  
$ git checkout OPENSTACK_RELEASE
```

2. Install the Dashboard panel plug-in:

```
$ python setup.py install
```

3. Copy the `_1481_project_ng_loadbalancersv2_panel.py` file from the `neutron-lbaas-dashboard/enabled` directory into the Dashboard `openstack_dashboard/local/enabled` directory.

This step ensures that Dashboard can find the plug-in when it enumerates all of its available panels.

4. Enable the plug-in in Dashboard by editing the `local_settings.py` file and setting `enable_lb` to `True` in the `OPENSTACK_NEUTRON_NETWORK` dictionary.
5. If Dashboard is configured to compress static files for better performance (usually set through `COMPRESS_OFFLINE` in `local_settings.py`), optimize the static files again:

```
$ ./manage.py collectstatic  
$ ./manage.py compress
```

6. Restart Apache to activate the new panel:

```
$ sudo service apache2 restart
```

To find the panel, click on *Project* in Dashboard, then click the *Network* drop-down menu and select *Load Balancers*.

LBaaS v2 operations

The same neutron commands are used for LBaaS v2 with an agent or with Octavia.

Building an LBaaS v2 load balancer

1. Start by creating a load balancer on a network. In this example, the private network is an isolated network with two web server instances:

```
$ neutron lbaas-loadbalancer-create --name test-lb private-subnet
```

2. You can view the load balancer status and IP address with the `neutron lbaas-loadbalancer-show` command:

```
$ neutron lbaas-loadbalancer-show test-lb  
+-----+-----+  
| Field | Value |
```

admin_state_up	True
description	
id	7780f9dd-e5dd-43a9-af81-0d2d1bd9c386
listeners	{"id": "23442d6a-4d82-40ee-8d08-243750dbc191"}
	{"id": "7e0d084d-6d67-47e6-9f77-0115e6cf9ba8"}
name	test-lb
operating_status	ONLINE
provider	haproxy
provisioning_status	ACTIVE
tenant_id	fbfce4cb346c4f9097a977c54904cafd
vip_address	192.0.2.22
vip_port_id	9f8f8a75-a731-4a34-b622-864907e1d556
vip_subnet_id	f1e7827d-1bfe-40b6-b8f0-2d9fd946f59b

3. Update the security group to allow traffic to reach the new load balancer. Create a new security group along with ingress rules to allow traffic into the new load balancer. The neutron port for the load balancer is shown as `vip_port_id` above.

Create a security group and rules to allow TCP port 80, TCP port 443, and all ICMP traffic:

```
$ neutron security-group-create lbaas
$ neutron security-group-rule-create \
    --direction ingress \
    --protocol tcp \
    --port-range-min 80 \
    --port-range-max 80 \
    --remote-ip-prefix 0.0.0.0/0 \
    lbaas
$ neutron security-group-rule-create \
    --direction ingress \
    --protocol tcp \
    --port-range-min 443 \
    --port-range-max 443 \
    --remote-ip-prefix 0.0.0.0/0 \
    lbaas
$ neutron security-group-rule-create \
    --direction ingress \
    --protocol icmp \
    lbaas
```

Apply the security group to the load balancer's network port using `vip_port_id` from the **neutron lbaas-loadbalancer-show** command:

```
$ neutron port-update \
    --security-group lbaas \
    9f8f8a75-a731-4a34-b622-864907e1d556
```

Adding an HTTP listener

1. With the load balancer online, you can add a listener for plaintext HTTP traffic on port 80:

```
$ neutron lbaas-listener-create \
    --name test-lb-http \
```

```
--loadbalancer test-lb \
--protocol HTTP \
--protocol-port 80
```

This load balancer is active and ready to serve traffic on 192.0.2.22.

- Verify that the load balancer is responding to pings before moving further:

```
$ ping -c 4 192.0.2.22
PING 192.0.2.22 (192.0.2.22) 56(84) bytes of data.
64 bytes from 192.0.2.22: icmp_seq=1 ttl=62 time=0.410 ms
64 bytes from 192.0.2.22: icmp_seq=2 ttl=62 time=0.407 ms
64 bytes from 192.0.2.22: icmp_seq=3 ttl=62 time=0.396 ms
64 bytes from 192.0.2.22: icmp_seq=4 ttl=62 time=0.397 ms

--- 192.0.2.22 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.396/0.402/0.410/0.020 ms
```

- You can begin building a pool and adding members to the pool to serve HTTP content on port 80. For this example, the web servers are 192.0.2.16 and 192.0.2.17:

```
$ neutron lbaas-pool-create \
--name test-lb-pool-http \
--lb-algorithm ROUND_ROBIN \
--listener test-lb-http \
--protocol HTTP
$ neutron lbaas-member-create \
--name test-lb-http-member-1 \
--subnet private-subnet \
--address 192.0.2.16 \
--protocol-port 80 \
test-lb-pool-http
$ neutron lbaas-member-create \
--name test-lb-http-member-2 \
--subnet private-subnet \
--address 192.0.2.17 \
--protocol-port 80 \
test-lb-pool-http
```

- You can use curl to verify connectivity through the load balancers to your web servers:

```
$ curl 192.0.2.22
web2
$ curl 192.0.2.22
web1
$ curl 192.0.2.22
web2
$ curl 192.0.2.22
web1
```

In this example, the load balancer uses the round robin algorithm and the traffic alternates between the web servers on the backend.

- You can add a health monitor so that unresponsive servers are removed from the pool:

```
$ neutron lbaas-healthmonitor-create \
--name test-lb-http-monitor \
--delay 5 \
--max-retries 2 \
--timeout 10 \
--type HTTP \
--pool test-lb-pool-http
```

In this example, the health monitor removes the server from the pool if it fails a health check at two five-second intervals. When the server recovers and begins responding to health checks again, it is added to the pool once again.

Adding an HTTPS listener

You can add another listener on port 443 for HTTPS traffic. LBaaS v2 offers SSL/TLS termination at the load balancer, but this example takes a simpler approach and allows encrypted connections to terminate at each member server.

1. Start by creating a listener, attaching a pool, and then adding members:

```
$ neutron lbaas-listener-create \
--name test-lb-https \
--loadbalancer test-lb \
--protocol HTTPS \
--protocol-port 443
$ neutron lbaas-pool-create \
--name test-lb-pool-https \
--lb-algorithm LEAST_CONNECTIONS \
--listener test-lb-https \
--protocol HTTPS
$ neutron lbaas-member-create \
--name test-lb-https-member-1 \
--subnet private-subnet \
--address 192.0.2.16 \
--protocol-port 443 \
test-lb-pool-https
$ neutron lbaas-member-create \
--name test-lb-https-member-2 \
--subnet private-subnet \
--address 192.0.2.17 \
--protocol-port 443 \
test-lb-pool-https
```

2. You can also add a health monitor for the HTTPS pool:

```
$ neutron lbaas-healthmonitor-create \
--name test-lb-https-monitor \
--delay 5 \
--max-retries 2 \
--timeout 10 \
--type HTTPS \
--pool test-lb-pool-https
```

The load balancer now handles traffic on ports 80 and 443.

Associating a floating IP address

Load balancers that are deployed on a public or provider network that are accessible to external clients do not need a floating IP address assigned. External clients can directly access the virtual IP address (VIP) of those load balancers.

However, load balancers deployed onto private or isolated networks need a floating IP address assigned if they must be accessible to external clients. To complete this step, you must have a router between the private and public networks and an available floating IP address.

You can use the **neutron lbaas-loadbalancer-show** command from the beginning of this section to locate the `vip_port_id`. The `vip_port_id` is the ID of the network port that is assigned to the load balancer. You can associate a free floating IP address to the load balancer using **neutron floatingip-associate**:

```
$ neutron floatingip-associate FLOATINGIP_ID LOAD_BALANCER_PORT_ID
```

Setting quotas for LBaaS v2

Quotas are available for limiting the number of load balancers and load balancer pools. By default, both quotas are set to 10.

You can adjust quotas using the **neutron quota-update** command:

```
$ neutron quota-update --tenant-id TENANT_UUID --loadbalancer 25
$ neutron quota-update --tenant-id TENANT_UUID --pool 50
```

A setting of -1 disables the quota for a tenant.

Retrieving load balancer statistics

The LBaaS v2 agent collects four types of statistics for each load balancer every six seconds. Users can query these statistics with the **neutron lbaas-loadbalancer-stats** command:

```
$ neutron lbaas-loadbalancer-stats test-lb
+-----+-----+
| Field      | Value   |
+-----+-----+
| active_connections | 0      |
| bytes_in      | 40264557 |
| bytes_out      | 71701666 |
| total_connections | 384601 |
+-----+-----+
```

The `active_connections` count is the total number of connections that were active at the time the agent polled the load balancer. The other three statistics are cumulative since the load balancer was last started. For example, if the load balancer restarts due to a system error or a configuration change, these statistics will be reset.

Macvtap mechanism driver

The Macvtap mechanism driver for the ML2 plug-in generally increases network performance of instances.

Consider the following attributes of this mechanism driver to determine practicality in your environment:

- Supports only instance ports. Ports for DHCP and layer-3 (routing) services must use another mechanism driver such as Linux bridge or Open vSwitch (OVS).
- Supports only untagged (flat) and tagged (VLAN) networks.
- Lacks support for security groups including basic (sanity) and anti-spoofing rules.
- Lacks support for layer-3 high-availability mechanisms such as Virtual Router Redundancy Protocol (VRRP) and Distributed Virtual Routing (DVR).
- Only compute resources can be attached via macvtap. Attaching other resources like DHCP, Routers and others is not supported. Therefore run either OVS or linux bridge in VLAN or flat mode on the controller node.
- Instance migration requires the same values for the `physical_interface_mapping` configuration option on each compute node. For more information, see <https://bugs.launchpad.net/neutron/+bug/1550400>.

Prerequisites

You can add this mechanism driver to an existing environment using either the Linux bridge or OVS mechanism drivers with only provider networks or provider and self-service networks. You can change the configuration of existing compute nodes or add compute nodes with the Macvtap mechanism driver. The example configuration assumes addition of compute nodes with the Macvtap mechanism driver to the [Linux bridge: Self-service networks](#) or [Open vSwitch: Self-service networks](#) deployment examples.

Add one or more compute nodes with the following components:

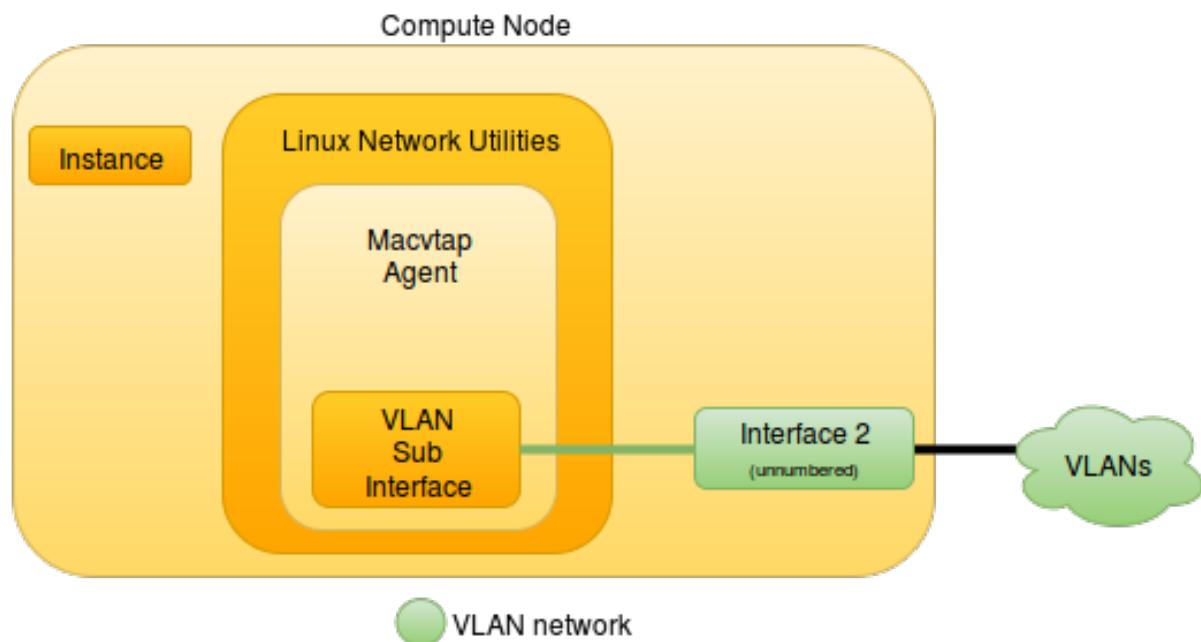
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking Macvtap layer-2 agent and any dependencies.

Note: To support integration with the deployment examples, this content configures the Macvtap mechanism driver to use the overlay network for untagged (flat) or tagged (VLAN) networks in addition to overlay networks such as VXLAN. Your physical network infrastructure must support VLAN (802.1q) tagging on the overlay network.

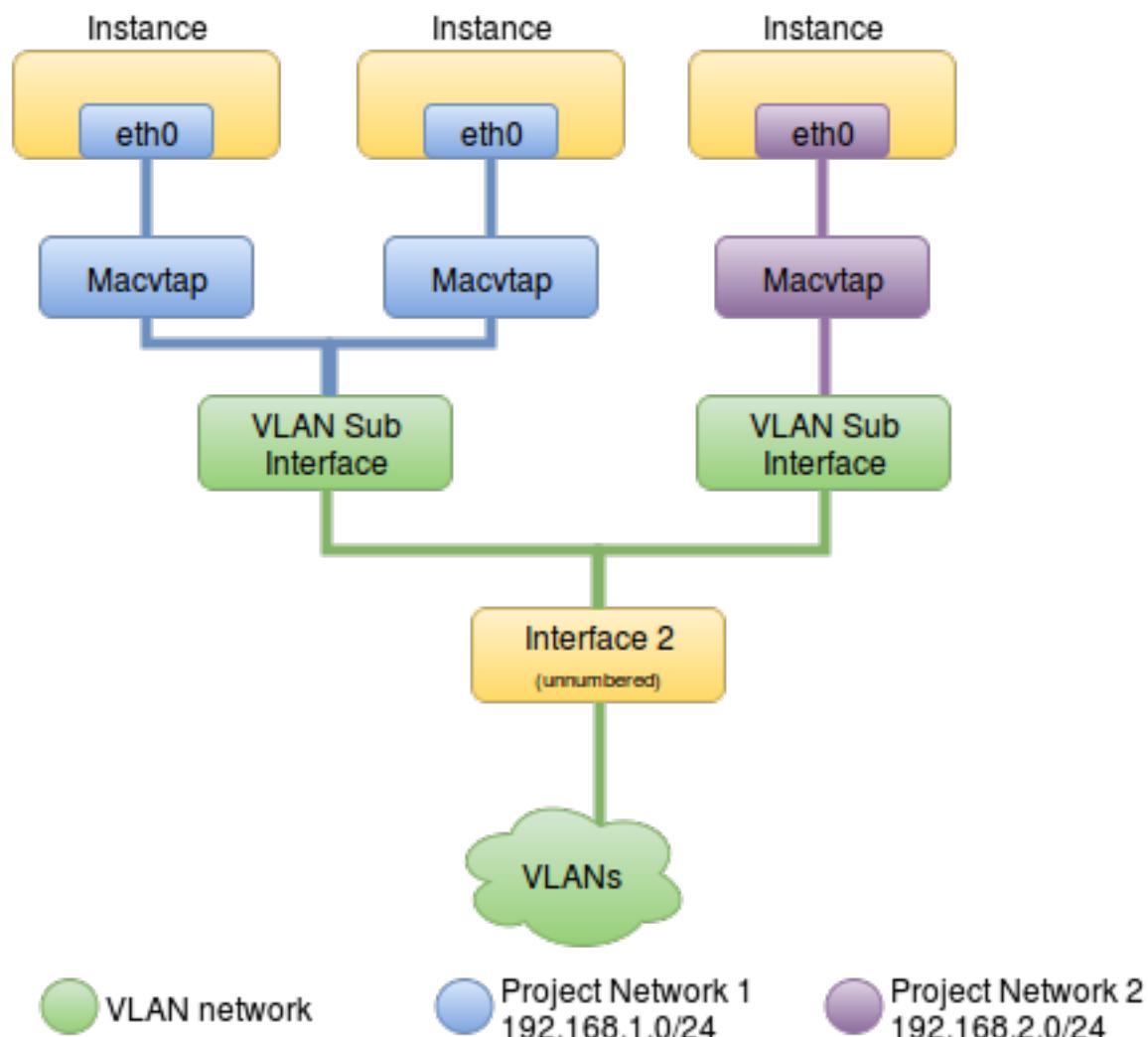
Architecture

The Macvtap mechanism driver only applies to compute nodes. Otherwise, the environment resembles the prerequisite deployment example.

Compute Node Overview



Compute Node Components



Example configuration

Use the following example configuration as a template to add support for the Macvtap mechanism driver to an existing operational environment.

Controller node

1. In the `ml2_conf.ini` file:
 - Add `macvtap` to mechanism drivers.

```
[ml2]
mechanism_drivers = macvtap
```

- Configure network mappings.

```
[ml2_type_flat]
flat_networks = provider,macvtap

[ml2_type_vlan]
network_vlan_ranges = provider,macvtap:VLAN_ID_START:VLAN_ID_END
```

Note: Use of `macvtap` is arbitrary. Only the self-service deployment examples require VLAN ID ranges. Replace `VLAN_ID_START` and `VLAN_ID_END` with appropriate numerical values.

2. Restart the following services:

- Server

Network nodes

No changes.

Compute nodes

1. Install the Networking service Macvtap layer-2 agent.
2. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the `[DEFAULT]`, `[database]`, `[keystone_auth_token]`, `[nova]`, and `[agent]` sections.

3. In the `macvtap_agent.ini` file, configure the layer-2 agent.

```
[macvtap]
physical_interface_mappings = macvtap:MACVTAP_INTERFACE

[securitygroup]
firewall_driver = noop
```

Replace MACVTAP_INTERFACE with the name of the underlying interface that handles Macvtap mechanism driver interfaces. If using a prerequisite deployment example, replace MACVTAP_INTERFACE with the name of the underlying interface that handles overlay networks. For example, eth1.

4. Start the following services:

- Macvtap agent

Verify service operation

1. Source the administrative project credentials.

2. Verify presence and operation of the agents:

\$ openstack network agent list				
ID	Agent Type	Host	Availability	
Zone	Alive	State	Binary	
31e1bc1b-c872-4429-8fc3-2c8eba52634e	Metadata agent	compute1	None	■
True UP neutron-metadata-agent				
378f5550-fee-e42aa-a1cb-e548b7c2601f	Open vSwitch agent	compute1	None	■
True UP neutron-openvswitch-agent				
7d2577d0-e640-42a3-b303-cb1eb077f2b6	L3 agent	compute1	nova	■
True UP neutron-l3-agent				
d5d7522c-ad14-4c63-ab45-f6420d6a81dd	Metering agent	compute1	None	■
True UP neutron-metering-agent				
e838ef5c-75b1-4b12-84da-7bdbd62f1040	DHCP agent	compute1	nova	■
True UP neutron-dhcp-agent				

Create initial networks

This mechanism driver simply changes the virtual network interface driver for instances. Thus, you can reference the [Create initial networks](#) content for the prerequisite deployment example.

Verify network operation

This mechanism driver simply changes the virtual network interface driver for instances. Thus, you can reference the [Verify network operation](#) content for the prerequisite deployment example.

Network traffic flow

This mechanism driver simply removes the Linux bridge handling security groups on the compute nodes. Thus, you can reference the network traffic flow scenarios for the prerequisite deployment example.

MTU considerations

The Networking service uses the MTU of the underlying physical network to calculate the MTU for virtual network components including instance network interfaces. By default, it assumes a standard 1500-byte MTU for the underlying physical network.

The Networking service only references the underlying physical network MTU. Changing the underlying physical network device MTU requires configuration of physical network devices such as switches and routers.

Jumbo frames

The Networking service supports underlying physical networks using jumbo frames and also enables instances to use jumbo frames minus any overlay protocol overhead. For example, an underlying physical network with a 9000-byte MTU yields a 8950-byte MTU for instances using a VXLAN network with IPv4 endpoints. Using IPv6 endpoints for overlay networks adds 20 bytes of overhead for any protocol.

The Networking service supports the following underlying physical network architectures. Case 1 refers to the most common architecture. In general, architectures should avoid cases 2 and 3.

Note: You can trigger MTU recalculation for existing networks by changing the MTU configuration and restarting the `neutron-server` service. However, propagating MTU calculations to the data plane may require users to delete and recreate ports on the network.

When using the Open vSwitch or Linux bridge drivers, new MTU calculations will be propagated automatically after restarting the `l3-agent` service.

Case 1

For typical underlying physical network architectures that implement a single MTU value, you can leverage jumbo frames using two options, one in the `neutron.conf` file and the other in the `m12_conf.ini` file. Most environments should use this configuration.

For example, referencing an underlying physical network with a 9000-byte MTU:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `m12_conf.ini` file:

```
[m12]
path_mtu = 9000
```

Case 2

Some underlying physical network architectures contain multiple layer-2 networks with different MTU values. You can configure each flat or VLAN provider network in the bridge or interface mapping options of the layer-2 agent to reference a unique MTU value.

For example, referencing a 4000-byte MTU for provider2, a 1500-byte MTU for provider3, and a 9000-byte MTU for other networks using the Open vSwitch agent:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `openvswitch_agent.ini` file:

```
[ovs]
bridge_mappings = provider1:eth1,provider2:eth2,provider3:eth3
```

3. In the `ml2_conf.ini` file:

```
[ml2]
physical_network_mtus = provider2:4000,provider3:1500
path_mtu = 9000
```

Case 3

Some underlying physical network architectures contain a unique layer-2 network for overlay networks using protocols such as VXLAN and GRE.

For example, referencing a 4000-byte MTU for overlay networks and a 9000-byte MTU for other networks:

1. In the `neutron.conf` file:

```
[DEFAULT]
global_physnet_mtu = 9000
```

2. In the `ml2_conf.ini` file:

```
[ml2]
path_mtu = 4000
```

Note: Other networks including provider networks and flat or VLAN self-service networks assume the value of the `global_physnet_mtu` option.

Instance network interfaces (VIFs)

The DHCP agent provides an appropriate MTU value to instances using IPv4, while the L3 agent provides an appropriate MTU value to instances using IPv6. IPv6 uses RA via the L3 agent because the DHCP agent only supports IPv4. Instances using IPv4 and IPv6 should obtain the same MTU value regardless of method.

Open vSwitch with DPDK datapath

This page serves as a guide for how to use the OVS with DPDK datapath functionality available in the Networking service as of the Mitaka release.

The basics

Open vSwitch (OVS) provides support for a Data Plane Development Kit (DPDK) datapath since OVS 2.2, and a DPDK-backed `vhost-user` virtual interface since OVS 2.4. The DPDK datapath provides lower latency and higher performance than the standard kernel OVS datapath, while DPDK-backed `vhost-user` interfaces can connect guests to this datapath. For more information on DPDK, refer to the [DPDK website](#).

OVS with DPDK, or OVS-DPDK, can be used to provide high-performance networking between instances on OpenStack compute nodes.

Prerequisites

Using DPDK in OVS requires the following minimum software versions:

- OVS 2.4
- DPDK 2.0
- QEMU 2.1.0
- libvirt 1.2.13

Multiqueue support is available if the following newer versions are used:

- OVS 2.5
- DPDK 2.2
- QEMU 2.5
- libvirt 1.2.17

In both cases, install and configure Open vSwitch with DPDK support for each node. For more information, see the [OVS-DPDK](#) installation guide.

Using `vhost-user` interfaces

Once OVS is correctly configured with DPDK support, `vhost-user` interfaces are completely transparent to the guest. However, guests must request large pages. This can be done through flavors. For example:

```
$ openstack flavor set m1.large --property hw:mem_page_size=large
```

For more information about the syntax for `hw:mem_page_size`, refer to the [Flavors](#) guide.

Note: `vhost-user` requires file descriptor-backed shared memory. Currently, the only way to request this is by requesting large pages. This is why instances spawned on hosts with OVS-DPDK must request large pages. The aggregate flavor affinity filter can be used to associate flavors with large page support to hosts with OVS-DPDK support.

Create and add `vhost-user` network interfaces to instances in the same fashion as conventional interfaces. These interfaces can use the kernel `virtio-net` driver or a DPDK-compatible driver in the guest

```
$ openstack server create --nic net-id=$net_id ... testserver
```

Known limitations

- This feature is only supported when using the libvirt compute driver, and the KVM/QEMU hypervisor.
- Large pages are required for each instance running on hosts with OVS-DPDK. If large pages are not present in the guest, the interface will appear but will not function.
- Expect performance degradation of services using tap devices: these devices do not support DPDK. Example services include DVR, FWaaS, or LBaaS.

Native Open vSwitch firewall driver

Note: Experimental feature or incomplete documentation.

Historically, Open vSwitch (OVS) could not interact directly with *iptables* to implement security groups. Thus, the OVS agent and Compute service use a Linux bridge between each instance (VM) and the OVS integration bridge `br-int` to implement security groups. The Linux bridge device contains the *iptables* rules pertaining to the instance. In general, additional components between instances and physical network infrastructure cause scalability and performance problems. To alleviate such problems, the OVS agent includes an optional firewall driver that natively implements security groups as flows in OVS rather than Linux bridge and *iptables*, thus increasing scalability and performance.

Prerequisites

The native OVS firewall implementation requires kernel and user space support for *conntrack*, thus requiring minimum versions of the Linux kernel and Open vSwitch. All cases require Open vSwitch version 2.5 or newer.

- Kernel version 4.3 or newer includes *conntrack* support.
- Kernel version 3.3, but less than 4.3, does not include *conntrack* support and requires building the OVS modules.

Enable the native OVS firewall driver

- On nodes running the Open vSwitch agent, edit the `openvswitch_agent.ini` file and enable the firewall driver.

```
[securitygroup]
firewall_driver = openvswitch
```

For more information, see the [developer documentation](#) and the [video](#).

Quality of Service (QoS)

QoS is defined as the ability to guarantee certain network requirements like bandwidth, latency, jitter, and reliability in order to satisfy a Service Level Agreement (SLA) between an application provider and end users.

Network devices such as switches and routers can mark traffic so that it is handled with a higher priority to fulfill the QoS conditions agreed under the SLA. In other cases, certain network traffic such as Voice over IP (VoIP) and video streaming needs to be transmitted with minimal bandwidth constraints. On a system without network

QoS management, all traffic will be transmitted in a “best-effort” manner making it impossible to guarantee service delivery to customers.

QoS is an advanced service plug-in. QoS is decoupled from the rest of the OpenStack Networking code on multiple levels and it is available through the ml2 extension driver.

Details about the DB models, API extension, and use cases are out of the scope of this guide but can be found in the [Neutron QoS specification](#).

Supported QoS rule types

Any plug-in or ml2 mechanism driver can claim support for some QoS rule types by providing a plug-in/driver class property called `supported_qos_rule_types` that returns a list of strings that correspond to [QoS rule types](#).

Note: Bandwidth limit is supported on OVS, Linux bridge, and SR-IOV mechanism drivers. For the Newton release onward DSCP marking is supported on the OVS and Linux bridge mechanism drivers, and the minimum bandwidth rules on the SR-IOV NIC mechanism driver.

In the most simple case, the property can be represented by a simple Python list defined on the class.

For an ml2 plug-in, the list of supported QoS rule types is defined as a common subset of rules supported by all active mechanism drivers.

Note: The list of supported rule types reported by core plug-in is not enforced when accessing QoS rule resources. This is mostly because then we would not be able to create any rules while at least one ml2 driver lacks support for QoS (at the moment of writing, only macvtap is such a driver).

Configuration

To enable the service, follow the steps below:

On network nodes:

1. Add the QoS service to the `service_plugins` setting in `/etc/neutron/neutron.conf`. For example:

```
service_plugins = \
    neutron.services.l3_router.l3_router_plugin.L3RouterPlugin,
    neutron.services.metering.metering_plugin.MeteringPlugin,
    neutron.services.qos.qos_plugin.QoSPlugin
```

2. Optionally, set the needed `notification_drivers` in the `[qos]` section in `/etc/neutron/neutron.conf` (`message_queue` is the default).
3. In `/etc/neutron/plugins/ml2/ml2_conf.ini`, add `qos` to `extension_drivers` in the `[ml2]` section. For example:

```
[ml2]
extension_drivers = port_security, qos
```

4. If the Open vSwitch agent is being used, set `extensions` to `qos` in the `[agent]` section of `/etc/neutron/plugins/ml2/openvswitch_agent.ini`. For example:

```
[agent]
extensions = qos
```

On compute nodes:

1. In /etc/neutron/plugins/ml2/openvswitch_agent.ini, add qos to the extensions setting in the [agent] section. For example:

```
[agent]
extensions = qos
```

Note: QoS currently works with ml2 only (SR-IOV, Open vSwitch, and linuxbridge are drivers that are enabled for QoS in Mitaka release).

Trusted projects policy.json configuration

If projects are trusted to administrate their own QoS policies in your cloud, neutron's file policy.json can be modified to allow this.

Modify /etc/neutron/policy.json policy entries as follows:

```
"get_policy": "rule:regular_user",
"create_policy": "rule:regular_user",
"update_policy": "rule:regular_user",
"delete_policy": "rule:regular_user",
"get_rule_type": "rule:regular_user",
```

To enable bandwidth limit rule:

```
"get_policy_bandwidth_limit_rule": "rule:regular_user",
"create_policy_bandwidth_limit_rule": "rule:regular_user",
"delete_policy_bandwidth_limit_rule": "rule:regular_user",
"update_policy_bandwidth_limit_rule": "rule:regular_user",
```

To enable DSCP marking rule:

```
"get_policy_dscp_marking_rule": "rule:regular_user",
"create_dscp_marking_rule": "rule:regular_user",
"delete_dscp_marking_rule": "rule:regular_user",
"update_dscp_marking_rule": "rule:regular_user",
```

To enable minimum bandwidth rule:

```
"get_policy_minimum_bandwidth_rule": "rule:regular_user",
"create_policy_minimum_bandwidth_rule": "rule:regular_user",
"delete_policy_minimum_bandwidth_rule": "rule:regular_user",
"update_policy_minimum_bandwidth_rule": "rule:regular_user",
```

User workflow

QoS policies are only created by admins with the default policy.json. Therefore, you should have the cloud operator set them up on behalf of the cloud projects.

If projects are trusted to create their own policies, check the trusted projects `policy.json` configuration section.

First, create a QoS policy and its bandwidth limit rule:

```
$ neutron qos-policy-create bw-limiter

Created a new policy:
+-----+-----+
| Field | Value |
+-----+-----+
| description | |
| id | 0ee1c673-5671-40ca-b55f-4cd4bbd999c7 |
| name | bw-limiter |
| rules | |
| shared | False |
| tenant_id | 85b859134de2428d94f6ee910dc545d8 |
+-----+-----+

$ neutron qos-bandwidth-limit-rule-create bw-limiter --max-kbps 3000 \
--max-burst-kbps 300

Created a new bandwidth_limit_rule:
+-----+-----+
| Field | Value |
+-----+-----+
| id | 92ceb52f-170f-49d0-9528-976e2fee2d6f |
| max_burst_kbps | 300 |
| max_kbps | 3000 |
+-----+-----+
```

Note: The burst value is given in kilobits, not in kilobits per second as the name of the parameter might suggest. This is an amount of data which can be sent before the bandwidth limit applies.

Note: The QoS implementation requires a burst value to ensure proper behavior of bandwidth limit rules in the Open vSwitch and Linux bridge agents. If you do not provide a value, it defaults to 80% of the bandwidth limit which works for typical TCP traffic.

Second, associate the created policy with an existing neutron port. In order to do this, user extracts the port id to be associated to the already created policy. In the next example, we will assign the `bw-limiter` policy to the VM with IP address `192.0.2.1`.

```
$ neutron port-list

+-----+-----+
| id | fixed_ips |
+-----+-----+
| 0271d1d9-1b16-4410-bd74-82cdf6dc5b3 | { ... , "ip_address": "192.0.2.1"} |
| 88101e57-76fa-4d12-b0e0-4fc7634b874a | { ... , "ip_address": "192.0.2.3"} |
| e04aab6a-5c6c-4bd9-a600-33333551a668 | { ... , "ip_address": "192.0.2.2"} |
+-----+-----+

$ neutron port-update 88101e57-76fa-4d12-b0e0-4fc7634b874a --qos-policy bw-limiter
Updated port: 88101e57-76fa-4d12-b0e0-4fc7634b874a
```

In order to detach a port from the QoS policy, simply update again the port configuration.

```
$ neutron port-update 88101e57-76fa-4d12-b0e0-4fc7634b874a --no-qos-policy
Updated port: 88101e57-76fa-4d12-b0e0-4fc7634b874a
```

Ports can be created with a policy attached to them too.

```
$ neutron port-create private --qos-policy-id bw-limiter

Created a new port:
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| allowed_address_pairs | |
| binding:vnic_type | normal |
| device_id | |
| device_owner | |
| dns_assignment | {"hostname": "host-192-0-2-4", ... } |
| dns_name | |
| fixed_ips | {"subnet_id": "fabaf9b6-7a84-43b6-9d23-543591b531b8", "ip_address": "192.0.2.4"} |
| id | c3cb8faa-db36-429d-bd25-6003faf63c5 |
| mac_address | fa:16:3e:02:65:15 |
| name | |
| network_id | 4920548d-1a6c-4d67-8de4-06501211587c |
| port_security_enabled | True |
| qos_policy_id | 0ee1c673-5671-40ca-b55f-4cd4bbd999c7 |
| security_groups | b9cecbc5-a136-4032-b196-fb3eb091fff2 |
| status | DOWN |
| tenant_id | 85b859134de2428d94f6ee910dc545d8 |
+-----+-----+
```

You can attach networks to a QoS policy. The meaning of this is that any compute port connected to the network will use the network policy by default unless the port has a specific policy attached to it. Network owned ports like DHCP and router ports are excluded from network policy application.

In order to attach a QoS policy to a network, update an existing network, or initially create the network attached to the policy.

```
$ neutron net-update private --qos-policy bw-limiter
Updated network: private
```

Note: Configuring the proper burst value is very important. If the burst value is set too low, bandwidth usage will be throttled even with a proper bandwidth limit setting. This issue is discussed in various documentation sources, for example in [Juniper's documentation](#). Burst value for TCP traffic can be set as 80% of desired bandwidth limit value. For example, if the bandwidth limit is set to 1000kbps then enough burst value will be 800kbit. If the configured burst value is too low, achieved bandwidth limit will be lower than expected. If the configured burst value is too high, too few packets could be limited and achieved bandwidth limit would be higher than expected.

Administrator enforcement

Administrators are able to enforce policies on project ports or networks. As long as the policy is not shared, the project is not be able to detach any policy attached to a network or port.

If the policy is shared, the project is able to attach or detach such policy from its own ports and networks.

Rule modification

You can modify rules at runtime. Rule modifications will be propagated to any attached port.

```
$ neutron qos-bandwidth-limit-rule-update \
  92ceb52f-170f-49d0-9528-976e2fee2d6f bw-limiter \
  --max-kbps 2000 --max-burst-kbps 200
Updated bandwidth_limit_rule: 92ceb52f-170f-49d0-9528-976e2fee2d6f

$ neutron qos-bandwidth-limit-rule-show \
  92ceb52f-170f-49d0-9528-976e2fee2d6f bw-limiter

+-----+-----+
| Field      | Value           |
+-----+-----+
| id         | 92ceb52f-170f-49d0-9528-976e2fee2d6f |
| max_burst_kbps | 200          |
| max_kbps    | 2000          |
+-----+-----+
```

Just like with bandwidth limiting, create a policy for DSCP marking rule:

```
$ neutron qos-policy-create dscp-marking

Created a new policy:
+-----+-----+
| Field      | Value           |
+-----+-----+
| description |               |
| id         | 8569fb4d-3d63-483e-b49a-9f9290d794f4 |
| name       | dscp-marking   |
| rules      |               |
| shared     | False          |
| tenant_id  | 85b859134de2428d94f6ee910dc545d8 |
+-----+-----+
```

You can create, update, list, delete, and show DSCP markings with the neutron client:

```
$ neutron qos-dscp-marking-rule-create dscp-marking --dscp-mark 26

Created a new dscp marking rule
+-----+-----+
| Field      | Value           |
+-----+-----+
| id         | 115e4f70-8034-4176-8fe9-2c47f8878a7d |
| dscp_mark  | 26            |
+-----+-----+
```

```
$ neutron qos-dscp-marking-rule-update \
    115e4f70-8034-4176-8fe9-2c47f8878a7d dscp-marking --dscp-mark 22
Updated dscp_rule: 115e4f70-8034-4176-8fe9-2c47f8878a7d

$ neutron qos-dscp-marking-rule-list dscp-marking

+-----+-----+
| id | dscp_mark |
+-----+-----+
| 115e4f70-8034-4176-8fe9-2c47f8878a7d | 22 |
+-----+-----+

$ neutron qos-dscp-marking-rule-show \
    115e4f70-8034-4176-8fe9-2c47f8878a7d dscp-marking

+-----+-----+
| Field | Value |
+-----+-----+
| id | 115e4f70-8034-4176-8fe9-2c47f8878a7d |
| dscp_mark | 22 |
+-----+-----+

$ neutron qos-dscp-marking-rule-delete \
    115e4f70-8034-4176-8fe9-2c47f8878a7d dscp-marking
Deleted dscp_rule: 115e4f70-8034-4176-8fe9-2c47f8878a7d
```

You can also include minimum bandwidth rules in your policy:

```
$ openstack network qos policy create bandwidth-control
+-----+-----+
| Field | Value |
+-----+-----+
| description | |
| id | 8491547e-add1-4c6c-a50e-42121237256c |
| name | bandwidth-control |
| project_id | 7cc5a84e415d48e69d2b06aa67b317d8 |
| rules | [] |
| shared | False |
+-----+-----+

$ openstack network qos rule create bandwidth-control \
    --type minimum-bandwidth --min-kbps 1000 --egress
+-----+-----+
| Field | Value |
+-----+-----+
| direction | egress |
| id | da858b32-44bc-43c9-b92b-cf6e2fa836ab |
| min_kbps | 1000 |
| name | None |
| project_id | |
+-----+-----+
```

A policy with a minimum bandwidth ensures best efforts are made to provide no less than the specified bandwidth to each port on which the rule is applied. However, as this feature is not yet integrated with the Compute scheduler, minimum bandwidth cannot be guaranteed.

It is also possible to combine several rules in one policy:

```
$ openstack network qos rule create bandwidth-control \
--type bandwidth-limit --max-kbps 50000 --max-burst-kbits 50000
+-----+
| Field      | Value
+-----+
| id          | 0db48906-a762-4d32-8694-3f65214c34a6 |
| max_burst_kbps | 50000 |
| max_kbps    | 50000 |
| name         | None |
| project_id   | |
+-----+

$ openstack network qos policy show bandwidth-control
+-----+
| Field      | Value
+-----+
| description | |
| id          | 8491547e-add1-4c6c-a50e-42121237256c |
| name        | bandwidth-control |
| project_id  | 7cc5a84e415d48e69d2b06aa67b317d8 |
| rules       | [{"u'max_kbps': 50000, u'type': u'bandwidth_limit',
|           |   u'id': u'0db48906-a762-4d32-8694-3f65214c34a6',
|           |   u'max_burst_kbps': 50000,
|           |   u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'},
|           |   {u'direction':
|           |     u'egress', u'min_kbps': 1000, u'type': u'minimum_bandwidth',
|           |     u'id': u'da858b32-44bc-43c9-b92b-cf6e2fa836ab',
|           |     u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'}] |
| shared      | False |
+-----+
```

Role-Based Access Control (RBAC)

The Role-Based Access Control (RBAC) policy framework enables both operators and users to grant access to resources for specific projects.

Supported objects for sharing with specific projects

Currently, the access that can be granted using this feature is supported by:

- Regular port creation permissions on networks (since Liberty).
- Binding QoS policies permissions to networks or ports (since Mitaka).
- Attaching router gateways to networks (since Mitaka).

Sharing an object with specific projects

Sharing an object with a specific project is accomplished by creating a policy entry that permits the target project the `access_as_shared` action on that object.

Sharing a network with specific projects

Create a network to share:

\$ openstack network create secret_network	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2017-01-25T20:16:40Z
description	
dns_domain	None
id	f55961b9-3eb8-42eb-ac96-b97038b568de
ipv4_address_scope	None
ipv6_address_scope	None
is_default	None
mtu	1450
name	secret_network
port_security_enabled	True
project_id	61b7eba037fd41f29cfba757c010faff
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	9
qos_policy_id	None
revision_number	3
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
updated_at	2017-01-25T20:16:40Z

Create the policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is b87b2fc13e0248a4a031d38e06dc191d):

\$ openstack network rbac create --target-project \	
Field	Value
action	access_as_shared
id	f93efdbf-f1e0-41d2-b093-8328959d469e
name	None
object_id	f55961b9-3eb8-42eb-ac96-b97038b568de
object_type	network
project_id	61b7eba037fd41f29cfba757c010faff
target_project_id	b87b2fc13e0248a4a031d38e06dc191d

The **target-project** parameter specifies the project that requires access to the network. The **action** parameter specifies what the project is allowed to do. The **type** parameter says that the target object is a network. The final parameter is the ID of the network we are granting access to.

Project b87b2fc13e0248a4a031d38e06dc191d will now be able to see the network when running **openstack network list** and **openstack network show** and will also be able to create ports on that network. No other users (other than admins and the owner) will be able to see the network.

To remove access for that project, delete the policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete f93efdbf-f1e0-41d2-b093-8328959d469e
```

If that project has ports on the network, the server will prevent the policy from being deleted until the ports have been deleted:

```
$ openstack network rbac delete f93efdbf-f1e0-41d2-b093-8328959d469e
RBAC policy on object f93efdbf-f1e0-41d2-b093-8328959d469e
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a network with an arbitrary number of projects.

Sharing a QoS policy with specific projects

Create a QoS policy to share:

```
$ openstack network qos policy create secret_policy
+-----+-----+
| Field      | Value
+-----+-----+
| description | 
| id          | 1f730d69-1c45-4ade-a8f2-89070ac4f046
| name        | secret_policy
| project_id  | 61b7eba037fd41f29cfba757c010faff
| rules       | []
| shared      | False
+-----+-----+
```

Create the RBAC policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is be98b82f8fdf46b696e9e01cebc33fd9):

```
$ openstack network rbac create --target-project \
be98b82f8fdf46b696e9e01cebc33fd9 --action access_as_shared \
--type qos_policy 1f730d69-1c45-4ade-a8f2-89070ac4f046
+-----+-----+
| Field      | Value
+-----+-----+
| action     | access_as_shared
| id          | 8828e38d-a0df-4c78-963b-e5f215d3d550
| name        | None
| object_id   | 1f730d69-1c45-4ade-a8f2-89070ac4f046
| object_type | qos_policy
| project_id  | 61b7eba037fd41f29cfba757c010faff
| target_project_id | be98b82f8fdf46b696e9e01cebc33fd9
+-----+-----+
```

The **target-project** parameter specifies the project that requires access to the QoS policy. The **action** parameter specifies what the project is allowed to do. The **type** parameter says that the target object is a QoS policy. The final parameter is the ID of the QoS policy we are granting access to.

Project be98b82f8fdf46b696e9e01cebc33fd9 will now be able to see the QoS policy when running **openstack network qos policy list** and **openstack network qos policy show** and will also be able to bind it to its ports or networks. No other users (other than admins and the owner) will be able to see the QoS policy.

To remove access for that project, delete the RBAC policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
```

If that project has ports or networks with the QoS policy applied to them, the server will not delete the RBAC policy until the QoS policy is no longer in use:

```
$ openstack network rbac delete 8828e38d-a0df-4c78-963b-e5f215d3d550
RBAC policy on object 8828e38d-a0df-4c78-963b-e5f215d3d550
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to share a qos-policy with an arbitrary number of projects.

How the ‘shared’ flag relates to these entries

As introduced in other guide entries, neutron provides a means of making an object (network, qos-policy) available to every project. This is accomplished using the shared flag on the supported object:

```
$ openstack network create global_network --share
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2017-01-25T20:32:06Z |
| description | |
| dns_domain | None |
| id | 84a7e627-573b-49da-af66-c9a65244f3ce |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | None |
| mtu | 1450 |
| name | global_network |
| port_security_enabled | True |
| project_id | 61b7eba037fd41f29cfba757c010faff |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 7 |
| qos_policy_id | None |
| revision_number | 3 |
| router:external | Internal |
| segments | None |
| shared | True |
| status | ACTIVE |
| subnets | |
| updated_at | 2017-01-25T20:32:07Z |
+-----+-----+
```

This is the equivalent of creating a policy on the network that permits every project to perform the action `access_as_shared` on that network. Neutron treats them as the same thing, so the policy entry for that network should be visible using the `openstack network rbac list` command:

\$ openstack network rbac list		
ID	Object Type	Object ID
58a5ee31-2ad6-467d-	qos_policy	1f730d69-1c45-4ade-
8bb8-8c2ae3dd1382		a8f2-89070ac4f046
27efbd79-f384-4d89-9dfc-	network	84a7e627-573b-49da-
6c4a606ceec6		af66-c9a65244f3ce

Use the `neutron rbac-show` command to see the details:

\$ openstack network rbac show 27efbd79-f384-4d89-9dfc-6c4a606ceec6	
Field	Value
action	access_as_shared
id	27efbd79-f384-4d89-9dfc-6c4a606ceec6
name	None
object_id	84a7e627-573b-49da-af66-c9a65244f3ce
object_type	network
project_id	61b7eba037fd41f29cfba757c010faff
target_project_id	*

The output shows that the entry allows the action `access_as_shared` on object `84a7e627-573b-49da-af66-c9a65244f3ce` of type `network` to target_tenant `*`, which is a wildcard that represents all projects.

Currently, the shared flag is just a mapping to the underlying RBAC policies for a network. Setting the flag to `True` on a network creates a wildcard RBAC entry. Setting it to `False` removes the wildcard entry.

When you run `openstack network list` or `openstack network show`, the shared flag is calculated by the server based on the calling project and the RBAC entries for each network. For QoS objects use `openstack network qos policy list` or `openstack network qos policy show` respectively. If there is a wildcard entry, the shared flag is always set to `True`. If there are only entries that share with specific projects, only the projects the object is shared to will see the flag as `True` and the rest will see the flag as `False`.

Allowing a network to be used as an external network

To make a network available as an external network for specific projects rather than all projects, use the `access_as_external` action.

1. Create a network that you want to be available as an external network:

\$ openstack network create secret_external_network	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2017-01-25T20:36:59Z

description	
dns_domain	None
id	802d4e9e-4649-43e6-9ee2-8d052a880cfb
ipv4_address_scope	None
ipv6_address_scope	None
is_default	None
mtu	1450
name	secret_external_network
port_security_enabled	True
project_id	61b7eba037fd41f29cfba757c010faff
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	21
qos_policy_id	None
revision_number	3
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
updated_at	2017-01-25T20:36:59Z

2. Create a policy entry using the **openstack network rbac create** command (in this example, the ID of the project we want to share with is 838030a7bf3c4d04b4b054c0f0b2b17c):

\$ openstack network rbac create --target-project \\\n838030a7bf3c4d04b4b054c0f0b2b17c --action access_as_external \\\n--type network 802d4e9e-4649-43e6-9ee2-8d052a880cfb	
<hr/>	
Field	Value
+-----+-----+	
action	access_as_external
id	afdd5b8d-b6f5-4a15-9817-5231434057be
name	None
object_id	802d4e9e-4649-43e6-9ee2-8d052a880cfb
object_type	network
project_id	61b7eba037fd41f29cfba757c010faff
target_project_id	838030a7bf3c4d04b4b054c0f0b2b17c
+-----+-----+	

The **target-project** parameter specifies the project that requires access to the network. The **action** parameter specifies what the project is allowed to do. The **type** parameter indicates that the target object is a network. The final parameter is the ID of the network we are granting external access to.

Now project 838030a7bf3c4d04b4b054c0f0b2b17c is able to see the network when running **openstack network list** and **openstack network show** and can attach router gateway ports to that network. No other users (other than admins and the owner) are able to see the network.

To remove access for that project, delete the policy that allows it using the **openstack network rbac delete** command:

```
$ openstack network rbac delete afdd5b8d-b6f5-4a15-9817-5231434057be
```

If that project has router gateway ports attached to that network, the server prevents the policy from being deleted until the ports have been deleted:

```
$ openstack network rbac delete afdd5b8d-b6f5-4a15-9817-5231434057be
RBAC policy on object afdd5b8d-b6f5-4a15-9817-5231434057be
cannot be removed because other objects depend on it.
```

This process can be repeated any number of times to make a network available as external to an arbitrary number of projects.

If a network is marked as external during creation, it now implicitly creates a wildcard RBAC policy granting everyone access to preserve previous behavior before this feature was added.

```
$ openstack network create global_external_network --external
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2017-01-25T20:41:44Z |
| description | |
| dns_domain | None |
| id | 72a257a2-a56e-4ac7-880f-94a4233abec6 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | None |
| mtu | 1450 |
| name | global_external_network |
| port_security_enabled | True |
| project_id | 61b7eba037fd41f29cfba757c010faff |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 69 |
| qos_policy_id | None |
| revision_number | 4 |
| router:external | External |
| segments | None |
| shared | False |
| status | ACTIVE |
| subnets | |
| updated_at | 2017-01-25T20:41:44Z |
+-----+-----+
```

In the output above the standard `router:external` attribute is `External` as expected. Now a wildcard policy is visible in the RBAC policy listings:

```
$ openstack network rbac list --long -c ID -c Action
+-----+-----+
| ID | Action |
+-----+-----+
| b694e541-bdca-480d-94ec-eda59ab7d71a | access_as_external |
+-----+-----+
```

You can modify or delete this policy with the same constraints as any other RBAC `access_as_external` policy.

Preventing regular users from sharing objects with each other

The default `policy.json` file will not allow regular users to share objects with every other project using a wildcard; however, it will allow them to share objects with specific project IDs.

If an operator wants to prevent normal users from doing this, the "`create_rbac_policy`": entry in `policy.json` can be adjusted from "" to "`rule:admin_only`".

Routed provider networks

Note: Experimental feature. Use of this feature requires the OpenStack client version 3.3 or newer.

Before routed provider networks, the Networking service could not present a multi-segment layer-3 network as a single entity. Thus, each operator typically chose one of the following architectures:

- Single large layer-2 network
- Multiple smaller layer-2 networks

Single large layer-2 networks become complex at scale and involve significant failure domains.

Multiple smaller layer-2 networks scale better and shrink failure domains, but leave network selection to the user. Without additional information, users cannot easily differentiate these networks.

A routed provider network enables a single provider network to represent multiple layer-2 networks (broadcast domains) or segments and enables the operator to present one network to users. However, the particular IP addresses available to an instance depend on the segment of the network available on the particular compute node.

Similar to conventional networking, layer-2 (switching) handles transit of traffic between ports on the same segment and layer-3 (routing) handles transit of traffic between segments.

Each segment requires at least one subnet that explicitly belongs to that segment. The association between a segment and a subnet distinguishes a routed provider network from other types of networks. The Networking service enforces that either zero or all subnets on a particular network associate with a segment. For example, attempting to create a subnet without a segment on a network containing subnets with segments generates an error.

The Networking service does not provide layer-3 services between segments. Instead, it relies on physical network infrastructure to route subnets. Thus, both the Networking service and physical network infrastructure must contain configuration for routed provider networks, similar to conventional provider networks. In the future, implementation of dynamic routing protocols may ease configuration of routed networks.

Limitations

The Newton implementation contains the following limitations:

- The Compute scheduler lacks awareness of IP address resources on a routed network. Thus, the scheduler could exhaust the IP addresses in one segment before assigning IP addresses from another segment. The Ocata release of the Compute scheduler should provide the capability of scheduling around segments without available IP addresses. In Newton, the Compute scheduler selects any compute node on the provider network. If the segment on that compute node lacks available IP addresses, port binding fails and the Compute scheduler chooses another compute node without regard to segments. Rescheduling

continues up to the maximum number of retries. Operators should monitor IP usage and add subnets to segments prior to exhaustion of IP addresses. For more information, see the [blueprint](#).

- A routed provider network cannot also function as a router : external networks which prevents compatibility with floating IPv4 addresses. Additional routing, possibly using *BGP dynamic routing*, could address this issue in the future.

Prerequisites

Routed provider networks require additional prerequisites over conventional provider networks. We recommend using the following procedure:

1. Begin with segments. The Networking service defines a segment using the following components:
 - Unique physical network name
 - Segmentation type
 - Segmentation ID

For example, provider1, VLAN, and 2016. See the [API reference](#) for more information.

Within a network, use a unique physical network name for each segment which enables reuse of the same segmentation details between subnets. For example, using the same VLAN ID across all segments of a particular provider network. Similar to conventional provider networks, the operator must provision the layer-2 physical network infrastructure accordingly.

2. Implement routing between segments.

The Networking service does not provision routing among segments. The operator must implement routing among segments of a provider network. Each subnet on a segment must contain the gateway address of the router interface on that particular subnet. For example:

Segment	Version	Addresses	Gateway
segment1	4	203.0.113.0/24	203.0.113.1
segment1	6	fd00:203:0:113::/64	fd00:203:0:113::1
segment2	4	198.51.100.0/24	198.51.100.1
segment2	6	fd00:198:51:100::/64	fd00:198:51:100::1

3. Map segments to compute nodes.

Routed provider networks imply that compute nodes reside on different segments. The operator must ensure that every compute host that is supposed to participate in a router provider network has direct connectivity to one of its segments.

Host	Rack	Physical Network
compute0001	rack 1	segment 1
compute0002	rack 1	segment 1
...
compute0101	rack 2	segment 2
compute0102	rack 2	segment 2
compute0102	rack 2	segment 2
...

4. Deploy DHCP agents.

Unlike conventional provider networks, a DHCP agent cannot support more than one segment within a network. The operator must deploy at least one DHCP agent per segment. Consider deploying DHCP

agents on compute nodes containing the segments rather than one or more network nodes to reduce node count.

Host	Rack	Physical Network
network0001	rack 1	segment 1
network0002	rack 2	segment 2
...

5. Configure communication of the Networking service with the Compute scheduler.

An instance with an interface with an IPv4 address in a routed provider network must be placed by the Compute scheduler in a host that has access to a segment with available IPv4 addresses. To make this possible, the Networking service communicates to the Compute scheduler the inventory of IPv4 addresses associated with each segment of a routed provider network. The operator must configure the authentication credentials that the Networking service will use to communicate with the Compute scheduler's placement API. Please see below an example configuration.

Note: Coordination between the Networking service and the Compute scheduler is not necessary for IPv6 subnets as a consequence of their large address spaces.

Note: The coordination between the Networking service and the Compute scheduler requires the following minimum API micro-versions.

- Compute service API: 2.41
 - Placement API: 1.1
-

Example configuration

Controller node

1. Enable the segments service plug-in by appending segments to the list of service_plugins in the neutron.conf file on all nodes running the neutron-server service:

```
[DEFAULT]
# ...
service_plugins = ..., segments
```

2. Add a placement section to the neutron.conf file with authentication credentials for the Compute service placement API:

```
[placement]
auth_uri = http://192.0.2.72/identity
project_domain_name = Default
project_name = service
user_domain_name = Default
password = apassword
username = nova
auth_url = http://192.0.2.72/identity_admin
auth_type = password
region_name = RegionOne
```

3. Restart the neutron-server service.

Network or compute nodes

- Configure the layer-2 agent on each node to map one or more segments to the appropriate physical network bridge or interface and restart the agent.

Create a routed provider network

The following steps create a routed provider network with two segments. Each segment contains one IPv4 subnet and one IPv6 subnet.

1. Source the administrative project credentials.
2. Create a VLAN provider network which includes a default segment. In this example, the network uses the provider1 physical network with VLAN ID 2016.

```
$ openstack network create --share --provider-physical-network provider1 \
    --provider-network-type vlan --provider-segment 2016 multisegment1
+-----+-----+
| Field          | Value
+-----+-----+
| admin_state_up | UP
| id             | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9
| ipv4_address_scope | None
| ipv6_address_scope | None
| l2_adjacency   | True
| mtu            | 1500
| name           | multisegment1
| port_security_enabled | True
| provider:network_type | vlan
| provider:physical_network | provider1
| provider:segmentation_id | 2016
| router:external | Internal
| shared          | True
| status          | ACTIVE
| subnets         |
| tags            | []
+-----+-----+
```

3. Rename the default segment to segment1.

```
$ openstack network segment list --network multisegment1
+-----+-----+
| ID          | Name      | Network
| Network Type | Segment |
+-----+-----+
| 43e16869-ad31-48e4-87ce-acf756709e18 | None     | 6ab19caa-dda9-4b3d-abc4-
| 5b8f435b98d9 | vlan     | 2016    |
+-----+-----+
```

```
$ openstack network segment set --name segment1 43e16869-ad31-48e4-87ce-acf756709e18
```

Note: This command provides no output.

4. Create a second segment on the provider network. In this example, the segment uses the provider2 physical network with VLAN ID 2016.

```
$ openstack network segment create --physical-network provider2 \
    --network-type vlan --segment 2016 --network multisenegment1 segment2
+-----+-----+
| Field      | Value          |
+-----+-----+
| description | None           |
| headers     |                 |
| id          | 053b7925-9a89-4489-9992-e164c8cc8763 |
| name        | segment2        |
| network_id  | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| network_type| vlan            |
| physical_network | provider2 |
| segmentation_id | 2016           |
+-----+-----+
```

5. Verify that the network contains the segment1 and segment2 segments.

```
$ openstack network segment list --network multisenegment1
+-----+-----+
| ID          | Network Type | Segment | Name       | Network |
+-----+-----+
| 053b7925-9a89-4489-9992-e164c8cc8763 | vlan       | 2016    | segment2 | 6ab19caa-dda9-4b3d-abc4-
| 43e16869-ad31-48e4-87ce-acf756709e18 | vlan       | 2016    | segment1 | 6ab19caa-dda9-4b3d-abc4-
+-----+-----+
```

6. Create subnets on the segment1 segment. In this example, the IPv4 subnet uses 203.0.113.0/24 and the IPv6 subnet uses fd00:203:0:113::/64.

```
$ openstack subnet create \
    --network multisenegment1 --network-segment segment1 \
    --ip-version 4 --subnet-range 203.0.113.0/24 \
    multisenegment1-segment1-v4
+-----+-----+
| Field      | Value          |
+-----+-----+
| allocation_pools | 203.0.113.2-203.0.113.254 |
| cidr        | 203.0.113.0/24   |
| enable_dhcp  | True            |
| gateway_ip   | 203.0.113.1     |
| id          | c428797a-6f8e-4cb1-b394-c404318a2762 |
| ip_version   | 4                |
| name        | multisenegment1-segment1-v4 |
+-----+-----+
```

```
| network_id | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| segment_id | 43e16869-ad31-48e4-87ce-acf756709e18 |
+-----+-----+
$ openstack subnet create \
    --network multisegment1 --network-segment segment1 \
    --ip-version 6 --subnet-range fd00:203:0:113::/64 \
    --ipv6-address-mode slaac multisegment1-segment1-v6
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr | fd00:203:0:113::/64 |
| enable_dhcp | True |
| gateway_ip | fd00:203:0:113::1 |
| id | e41cb069-9902-4c01-9e1c-268c8252256a |
| ip_version | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode | None |
| name | multisegment1-segment1-v6 |
| network_id | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| segment_id | 43e16869-ad31-48e4-87ce-acf756709e18 |
+-----+-----+
```

Note: By default, IPv6 subnets on provider networks rely on physical network infrastructure for stateless address autoconfiguration (SLAAC) and router advertisement.

7. Create subnets on the segment2 segment. In this example, the IPv4 subnet uses 198.51.100.0/24 and the IPv6 subnet uses fd00:198:51:100::/64.

```
$ openstack subnet create \
    --network multisegment1 --network-segment segment2 \
    --ip-version 4 --subnet-range 198.51.100.0/24 \
    multisegment1-segment2-v4
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 198.51.100.2-198.51.100.254 |
| cidr | 198.51.100.0/24 |
| enable_dhcp | True |
| gateway_ip | 198.51.100.1 |
| id | 242755c2-f5fd-4e7d-bd7a-342ca95e50b2 |
| ip_version | 4 |
| name | multisegment1-segment2-v4 |
| network_id | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| segment_id | 053b7925-9a89-4489-9992-e164c8cc8763 |
+-----+-----+
$ openstack subnet create \
    --network multisegment1 --network-segment segment2 \
    --ip-version 6 --subnet-range fd00:198:51:100::/64 \
    --ipv6-address-mode slaac multisegment1-segment2-v6
+-----+-----+
| Field | Value |
+-----+-----+
```

allocation_pools	fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff
cidr	fd00:198:51:100::/64
enable_dhcp	True
gateway_ip	fd00:198:51:100::1
id	b884c40e-9cfe-4d1b-a085-0a15488e9441
ip_version	6
ipv6_address_mode	slaac
ipv6_ra_mode	None
name	multisegment1-segment2-v6
network_id	6ab19caa-dda9-4b3d-abc4-5b8f435b98d9
segment_id	053b7925-9a89-4489-9992-e164c8cc8763
+-----+-----+-----+-----+	+-----+-----+-----+-----+

8. Verify that each IPv4 subnet associates with at least one DHCP agent.

```
$ neutron dhcp-agent-list-hosting-net multisegment1
+-----+-----+-----+-----+
| id | host | admin_state_up | alive |
+-----+-----+-----+-----+
| c904ed10-922c-4c1a-84fd-d928abaf8f55 | compute0001 | True | :-) |
| e0b22cc0-d2a6-4f1c-b17c-27558e20b454 | compute0101 | True | :-) |
+-----+-----+-----+-----+
```

9. Verify that inventories were created for each segment IPv4 subnet in the Compute service placement API (for the sake of brevity, only one of the segments is shown in this example).

```
$ SEGMENT_ID=053b7925-9a89-4489-9992-e164c8cc8763
$ curl -s -X GET \
  http://localhost/placement/resource_providers/$SEGMENT_ID/inventories \
  -H "Content-type: application/json" \
  -H "X-Auth-Token: $TOKEN" \
  -H "Openstack-Api-Version: placement 1.1"
{
  "resource_provider_generation": 1,
  "inventories": {
    "allocation_ratio": 1,
    "total": 254,
    "reserved": 2,
    "step_size": 1,
    "min_unit": 1,
    "max_unit": 1
  }
}
```

Note: As of the writing of this guide, there is not placement API CLI client, so the `curl` command is used for this example.

10. Verify that host aggregates were created for each segment in the Compute service (for the sake of brevity, only one of the segments is shown in this example).

```
$ nova aggregate-list
+-----+-----+-----+
| Id | Name | Availability Zone |
+-----+-----+-----+
```

10 Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763	
+-----+	+-----+

11. Launch one or more instances. Each instance obtains IP addresses according to the segment it uses on the particular compute node.

Note: Creating a port and passing it to an instance yields a different behavior than conventional networks. The Networking service defers assignment of IP addresses to the port until the particular compute node becomes apparent. For example:

\$ openstack port create --network multisegment1 port1	
Field	Value
+-----+-----+	
admin_state_up	UP
binding_vnic_type	normal
id	6181fb47-7a74-4add-9b6b-f9837c1c90c4
ip_allocation	deferred
mac_address	fa:16:3e:34:de:9b
name	port1
network_id	6ab19caa-dda9-4b3d-abc4-5b8f435b98d9
port_security_enabled	True
security_groups	e4fcfef0d-e2c5-40c3-a385-9c33ac9289c5
status	DOWN
+-----+-----+	

Service function chaining

Service function chain (SFC) essentially refers to the *software-defined networking (SDN)* version of *policy-based routing (PBR)*. In many cases, SFC involves security, although it can include a variety of other features.

Fundamentally, SFC routes packets through one or more service functions instead of conventional routing that routes packets using destination IP address. Service functions essentially emulate a series of physical network devices with cables linking them together.

A basic example of SFC involves routing packets from one location to another through a firewall that lacks a “next hop” IP address from a conventional routing perspective. A more complex example involves an ordered series of service functions, each implemented using multiple instances (VMs). Packets must flow through one instance and a hashing algorithm distributes flows across multiple instances at each hop.

Architecture

All OpenStack Networking services and OpenStack Compute instances connect to a virtual network via ports making it possible to create a traffic steering model for service chaining using only ports. Including these ports in a port chain enables steering of traffic through one or more instances providing service functions.

A port chain, or service function path, consists of the following:

- A set of ports that define the sequence of service functions.
- A set of flow classifiers that specify the classified traffic flows entering the chain.

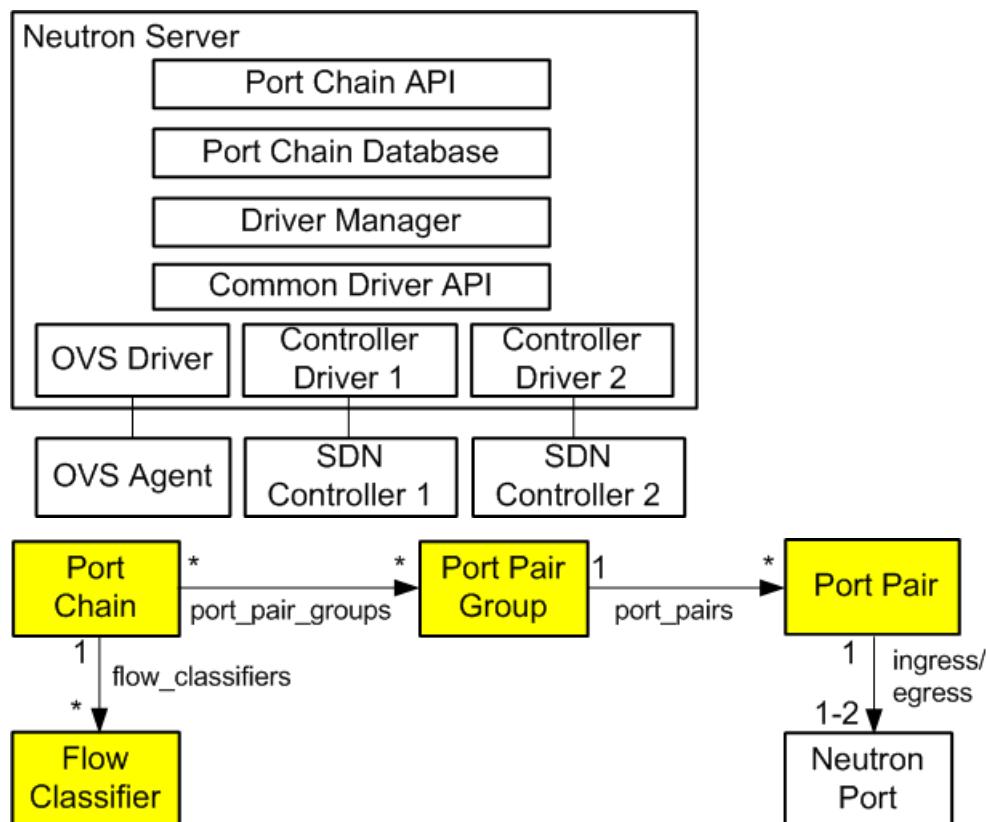
If a service function involves a pair of ports, the first port acts as the ingress port of the service function and the second port acts as the egress port. If both ports use the same value, they function as a single virtual bidirectional port.

A port chain is a unidirectional service chain. The first port acts as the head of the service function chain and the second port acts as the tail of the service function chain. A bidirectional service function chain consists of two unidirectional port chains.

A flow classifier can only belong to one port chain to prevent ambiguity as to which chain should handle packets in the flow. A check prevents such ambiguity. However, you can associate multiple flow classifiers with a port chain because multiple flows can request the same service function path.

Currently, SFC lacks support for multi-project service functions.

The port chain plug-in supports backing service providers including the OVS driver and a variety of SDN controller drivers. The common driver API enables different drivers to provide different implementations for the service chain path rendering.



See the [developer documentation](#) for more information.

Resources

Port chain

- `id` - Port chain ID
- `tenant_id` - Project ID
- `name` - Readable name
- `description` - Readable description

- `port_pair_groups` - List of port pair group IDs
- `flow_classifiers` - List of flow classifier IDs
- `chain_parameters` - Dictionary of chain parameters

A port chain consists of a sequence of port pair groups. Each port pair group is a hop in the port chain. A group of port pairs represents service functions providing equivalent functionality. For example, a group of firewall service functions.

A flow classifier identifies a flow. A port chain can contain multiple flow classifiers. Omitting the flow classifier effectively prevents steering of traffic through the port chain.

The `chain_parameters` attribute contains one or more parameters for the port chain. Currently, it only supports a correlation parameter that defaults to `mpls` for consistency with [Open vSwitch](#) (OVS) capabilities. Future values for the correlation parameter may include the [*network service header \(NSH\)*](#).

Port pair group

- `id` - Port pair group ID
- `tenant_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `port_pairs` - List of service function port pairs

A port pair group may contain one or more port pairs. Multiple port pairs enable load balancing/distribution over a set of functionally equivalent service functions.

Port pair

- `id` - Port pair ID
- `tenant_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `ingress` - Ingress port
- `egress` - Egress port
- `service_function_parameters` - Dictionary of service function parameters

A port pair represents a service function instance that includes an ingress and egress port. A service function containing a bidirectional port uses the same ingress and egress port.

The `service_function_parameters` attribute includes one or more parameters for the service function. Currently, it only supports a correlation parameter that determines association of a packet with a chain. This parameter defaults to `none` for legacy service functions that lack support for correlation such as the NSH. If set to `none`, the data plane implementation must provide service function proxy functionality.

Flow classifier

- `id` - Flow classifier ID
- `tenant_id` - Project ID
- `name` - Readable name
- `description` - Readable description
- `ethertype` - Ethertype (IPv4/IPv6)
- `protocol` - IP protocol
- `source_port_range_min` - Minimum source protocol port
- `source_port_range_max` - Maximum source protocol port
- `destination_port_range_min` - Minimum destination protocol port
- `destination_port_range_max` - Maximum destination protocol port
- `source_ip_prefix` - Source IP address or prefix
- `destination_ip_prefix` - Destination IP address or prefix
- `logical_source_port` - Source port
- `logical_destination_port` - Destination port
- `l7_parameters` - Dictionary of L7 parameters

A combination of the source attributes defines the source of the flow. A combination of the destination attributes defines the destination of the flow. The `l7_parameters` attribute is a place holder that may be used to support flow classification using layer 7 fields, such as a URL. If unspecified, the `logical_source_port` and `logical_destination_port` attributes default to none, the `ethertype` attribute defaults to IPv4, and all other attributes default to a wildcard value.

Operations

Create a port chain

The following example uses the neutron command-line interface (CLI) to create a port chain consisting of three service function instances to handle HTTP (TCP) traffic flows from 192.0.2.11:1000 to 198.51.100.11:80.

- Instance 1
 - Name: vm1
 - Function: Firewall
 - Port pair: [p1, p2]
- Instance 2
 - Name: vm2
 - Function: Firewall
 - Port pair: [p3, p4]
- Instance 3

- Name: vm3
- Function: Intrusion detection system (IDS)
- Port pair: [p5, p6]

Note: The example network net1 must exist before creating ports on it.

1. Source the credentials of the project that owns the net1 network.
2. Create ports on network net1 and record the UUID values.

```
$ openstack port create p1 --network net1
$ openstack port create p2 --network net1
$ openstack port create p3 --network net1
$ openstack port create p4 --network net1
$ openstack port create p5 --network net1
$ openstack port create p6 --network net1
```

3. Launch service function instance vm1 using ports p1 and p2, vm2 using ports p3 and p4, and vm3 using ports p5 and p6.

```
$ openstack server create --nic port-id=P1_ID --nic port-id=P2_ID vm1
$ openstack server create --nic port-id=P3_ID --nic port-id=P4_ID vm2
$ openstack server create --nic port-id=P5_ID --nic port-id=P6_ID vm3
```

Replace P1_ID, P2_ID, P3_ID, P4_ID, P5_ID, and P6_ID with the UUIDs of the respective ports.

Note: This command requires additional options to successfully launch an instance. See the [CLI reference](#) for more information.

Alternatively, you can launch each instance with one network interface and attach additional ports later.

4. Create flow classifier FC1 that matches the appropriate packet headers.

```
$ neutron flow-classifier-create \
--description "HTTP traffic from 192.0.2.11 to 198.51.100.11" \
--ethertype IPv4 \
--source-ip-prefix 192.0.2.11/32 \
--destination-ip-prefix 198.51.100.11/32 \
--protocol tcp \
--source-port 1000:1000 \
--destination-port 80:80 FC1
```

5. Create port pair PP1 with ports p1 and p2, PP2 with ports p3 and p4, and PP3 with ports p5 and p6.

```
$ neutron port-pair-create \
--description "Firewall SF instance 1" \
--ingress p1 \
--egress p2 PP1

$ neutron port-pair-create \
--description "Firewall SF instance 2" \
--ingress p3 \
--egress p4 PP2
```

```
$ neutron port-pair-create \
--description "IDS SF instance" \
--ingress p5 \
--egress p6 PP3
```

6. Create port pair group PPG1 with port pair PP1 and PP2 and PPG2 with port pair PP3.

```
$ neutron port-pair-group-create \
--port-pair PP1 --port-pair PP2 PPG1
$ neutron port-pair-group-create \
--port-pair PP3 PPG2
```

Note: You can repeat the `--port-pair` option for multiple port pairs of functionally equivalent service functions.

7. Create port chain PC1 with port pair groups PPG1 and PPG2 and flow classifier FC1.

```
$ neutron port-chain-create \
--port-pair-group PPG1 --port-pair-group PPG2 \
--flow-classifier FC1 PC1
```

Note: You can repeat the `--port-pair-group` option to specify additional port pair groups in the port chain. A port chain must contain at least one port pair group.

You can repeat the `--flow-classifier` option to specify multiple flow classifiers for a port chain. Each flow classifier identifies a flow.

Update a port chain or port pair group

- Use the **neutron port-chain-update** command to dynamically add or remove port pair groups or flow classifiers on a port chain.

- For example, add port pair group PPG3 to port chain PC1:

```
$ neutron port-chain-update \
--port-pair-group PPG1 --port-pair-group PPG2 --port-pair-group PPG3 \
--flow-classifier FC1 PC1
```

- For example, add flow classifier FC2 to port chain PC1:

```
$ neutron port-chain-update \
--port-pair-group PPG1 --port-pair-group PPG2 \
--flow-classifier FC1 --flow-classifier FC2 PC1
```

SFC steers traffic matching the additional flow classifier to the port pair groups in the port chain.

- Use the **neutron port-pair-group-update** command to perform dynamic scale-out or scale-in operations by adding or removing port pairs on a port pair group.

```
$ neutron port-pair-group-update \
--port-pair PP1 --port-pair PP2 --port-pair PP4 PPG1
```

SFC performs load balancing/distribution over the additional service functions in the port pair group.

SR-IOV

The purpose of this page is to describe how to enable SR-IOV functionality available in OpenStack (using OpenStack Networking). This functionality was first introduced in the OpenStack Juno release. This page intends to serve as a guide for how to configure OpenStack Networking and OpenStack Compute to create SR-IOV ports.

The basics

PCI-SIG Single Root I/O Virtualization and Sharing (SR-IOV) functionality is available in OpenStack since the Juno release. The SR-IOV specification defines a standardized mechanism to virtualize PCIe devices. This mechanism can virtualize a single PCIe Ethernet controller to appear as multiple PCIe devices. Each device can be directly assigned to an instance, bypassing the hypervisor and virtual switch layer. As a result, users are able to achieve low latency and near-line wire speed.

The following terms are used throughout this document:

Term	Definition
PF	Physical Function. The physical Ethernet controller that supports SR-IOV.
VF	Virtual Function. The virtual PCIe device created from a physical Ethernet controller.

SR-IOV agent

The SR-IOV agent allows you to set the admin state of ports, configure port security (enable and disable spoof checking), and configure QoS rate limiting. You must include the SR-IOV agent on each compute node using SR-IOV ports.

Note: The SR-IOV agent was optional before Mitaka, and was not enabled by default before Liberty.

Note: The ability to control port security and QoS rate limit settings was added in Liberty.

Supported Ethernet controllers

The following manufacturers are known to work:

- Intel
- Mellanox
- QLogic

For information on **Mellanox SR-IOV Ethernet ConnectX-3/ConnectX-3 Pro cards**, see [Mellanox: How To Configure SR-IOV VFs](#).

For information on **QLogic SR-IOV Ethernet cards**, see [User's Guide OpenStack Deployment with SR-IOV Configuration](#).

Using SR-IOV interfaces

In order to enable SR-IOV, the following steps are required:

1. Create Virtual Functions (Compute)
2. Whitelist PCI devices in nova-compute (Compute)
3. Configure neutron-server (Controller)
4. Configure nova-scheduler (Controller)
5. Enable neutron sriov-agent (Compute)

We recommend using VLAN provider networks for segregation. This way you can combine instances without SR-IOV ports and instances with SR-IOV ports on a single network.

Note: Throughout this guide, `eth3` is used as the PF and `physnet2` is used as the provider network configured as a VLAN range. These ports may vary in different environments.

Create Virtual Functions (Compute)

Create the VFs for the network interface that will be used for SR-IOV. We use `eth3` as PF, which is also used as the interface for the VLAN provider network and has access to the private networks of all machines.

Note: The steps detail how to create VFs using Mellanox ConnectX-4 and newer/Intel SR-IOV Ethernet cards on an Intel system. Steps may differ for different hardware configurations.

1. Ensure SR-IOV and VT-d are enabled in BIOS.
2. Enable IOMMU in Linux by adding `intel_iommu=on` to the kernel parameters, for example, using GRUB.
3. On each compute node, create the VFs via the PCI SYS interface:

```
# echo '8' > /sys/class/net/eth3/device/sriov_numvfs
```

Note: On some PCI devices, observe that when changing the amount of VFs you receive the error `Device or resource busy`. In this case, you must first set `sriov_numvfs` to 0, then set it to your new value.

Warning: Alternatively, you can create VFs by passing the `max_vfs` to the kernel module of your network interface. However, the `max_vfs` parameter has been deprecated, so the PCI SYS interface is the preferred method.

You can determine the maximum number of VFs a PF can support:

```
# cat /sys/class/net/eth3/device/sriov_totalvfs  
63
```

4. Verify that the VFs have been created and are in up state:

```
# lspci | grep Ethernet  
82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network  
↳ Connection (rev 01)  
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network  
↳ Connection (rev 01)  
82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:10.4 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:10.6 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:11.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:11.4 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)  
82:11.6 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual  
↳ Function (rev 01)
```

```
# ip link show eth3  
8: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT  
↳ qlen 1000  
    link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff  
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 4 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 5 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 6 MAC 00:00:00:00:00:00, spoof checking on, link-state auto  
    vf 7 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

If the interfaces are down, set them to up before launching a guest, otherwise the instance will fail to spawn:

```
# ip link set eth3 up
```

5. Persist created VFs on reboot:

```
# echo "echo '7' > /sys/class/net/eth3/device/sriov_numvfs" >> /etc/rc.local
```

Note: The suggested way of making PCI SYS settings persistent is through the sysfsutils tool. However, this is not available by default on many major distributions.

Whitelist PCI devices nova-compute (Compute)

1. Configure which PCI devices the nova-compute service may use. Edit the `nova.conf` file:

```
[default]
pci_passthrough_whitelist = { "devname": "eth3", "physical_network": "physnet2"}
```

This tells the Compute service that all VFs belonging to eth3 are allowed to be passed through to instances and belong to the provider network physnet2.

Alternatively the `pci_passthrough_whitelist` parameter also supports whitelisting by:

- PCI address: The address uses the same syntax as in `lspci` and an asterisk (*) can be used to match anything.

```
pci_passthrough_whitelist = { "address": "[[[[<domain>]:]<bus>]:]<slot>][.[
↳<function>]]", "physical_network": "physnet2" }
```

For example, to match any domain, bus 0a, slot 00, and all functions:

```
pci_passthrough_whitelist = { "address": "*:0a:00.*", "physical_network":
↳"physnet2" }
```

- PCI vendor_id and product_id as displayed by the Linux utility `lspci`.

```
pci_passthrough_whitelist = { "vendor_id": "<id>", "product_id": "<id>",
↳"physical_network": "physnet2" }
```

If the device defined by the PCI address or devname corresponds to an SR-IOV PF, all VFs under the PF will match the entry. Multiple `pci_passthrough_whitelist` entries per host are supported.

2. Restart the nova-compute service for the changes to go into effect.

Configure neutron-server (Controller)

1. Add `sriovnicswitch` as mechanism driver. Edit the `ml2_conf.ini` file on each controller:

```
mechanism_drivers = openvswitch,sriovnicswitch
```

2. Add the `ml2_conf_sriov.ini` file as parameter to the neutron-server service. Edit the appropriate initialization script to configure the neutron-server service to load the SR-IOV configuration file:

```
--config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugin.ini
--config-file /etc/neutron/plugins/ml2/ml2_conf_sriov.ini
```

3. Restart the neutron-server service.

Configure nova-scheduler (Controller)

1. On every controller node running the nova-scheduler service, add PciPassthroughFilter to scheduler_default_filters to enable PciPassthroughFilter by default. Also ensure scheduler_available_filters parameter under the [DEFAULT] section in nova.conf is set to all_filters to enable all filters provided by the Compute service.

```
[DEFAULT]
scheduler_default_filters = RetryFilter, AvailabilityZoneFilter, RamFilter, ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter, ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter, PciPassthroughFilter
scheduler_available_filters = nova.scheduler.filters.all_filters
```

2. Restart the nova-scheduler service.

Enable neutron sriov-agent (Compute)

1. Install the SR-IOV agent.
2. Edit the sriov_agent.ini file on each compute node. For example:

```
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver

[sriov_nic]
physical_device_mappings = physnet2:eth3
exclude_devices =
```

Note: The physical_device_mappings parameter is not limited to be a 1-1 mapping between physical networks and NICs. This enables you to map the same physical network to more than one NIC. For example, if physnet2 is connected to eth3 and eth4, then physnet2:eth3,physnet2:eth4 is a valid option.

The exclude_devices parameter is empty, therefore, all the VFs associated with eth3 may be configured by the agent. To exclude specific VFs, add them to the exclude_devices parameter as follows:

```
exclude_devices = eth1:0000:07:00.2;0000:07:00.3,eth2:0000:05:00.1;0000:05:00.2
```

3. Ensure the neutron sriov-agent runs successfully:

```
# neutron-sriov-nic-agent \
--config-file /etc/neutron/neutron.conf \
--config-file /etc/neutron/plugins/ml2/sriov_agent.ini
```

4. Enable the neutron sriov-agent service.

If installing from source, you must configure a daemon file for the init system manually.

(Optional) FDB L2 agent extension

Forwarding DataBase (FDB) population is an L2 agent extension to OVS agent or Linux bridge. Its objective is to update the FDB table for existing instance using normal port. This enables communication between SR-IOV

instances and normal instances. The use cases of the FDB population extension are:

- Direct port and normal port instances reside on the same compute node.
- Direct port instance that uses floating IP address and network node are located on the same host.

For additional information describing the problem, refer to: [Virtual switching technologies and Linux bridge](#).

1. Edit the `ovs_agent.ini` or `linuxbridge_agent.ini` file on each compute node. For example:

```
[agent]
extensions = fdb
```

2. Add the FDB section and the `shared_physical_device_mappings` parameter. This parameter maps each physical port to its physical network name. Each physical network can be mapped to several ports:

```
[FDB]
shared_physical_device_mappings = physnet1:p1p1, physnet1:p1p2
```

Launching instances with SR-IOV ports

Once configuration is complete, you can launch instances with SR-IOV ports.

1. Get the `id` of the network where you want the SR-IOV port to be created:

```
$ net_id=`neutron net-show net04 | grep "\ id\ " | awk '{ print $4 }'`
```

2. Create the SR-IOV port. `vnic_type=direct` is used here, but other options include `normal`, `direct-physical`, and `macvtap`:

```
$ port_id=`neutron port-create $net_id --name sriov_port --binding:vnic_type direct | grep "\ id\ " | awk '{ print $4 }'`
```

3. Create the instance. Specify the SR-IOV port created in step two for the NIC:

```
$ openstack server create --flavor m1.large --image ubuntu_14.04 --nic port-id=$port_id test-sriov
```

Note: There are two ways to attach VFs to an instance. You can create an SR-IOV port or use the `pci_alias` in the Compute service. For more information about using `pci_alias`, refer to [nova-api configuration](#).

SR-IOV with InfiniBand

The support for SR-IOV with InfiniBand allows a Virtual PCI device (VF) to be directly mapped to the guest, allowing higher performance and advanced features such as RDMA (remote direct memory access). To use this feature, you must:

1. Use InfiniBand enabled network adapters.
2. Run InfiniBand subnet managers to enable InfiniBand fabric.

All InfiniBand networks must have a subnet manager running for the network to function. This is true even when doing a simple network of two machines with no switch and the cards are plugged in back-to-back. A subnet manager is required for the link on the cards to come up. It is possible to have more than one subnet manager. In this case, one of them will act as the master, and any other will act as a slave that will take over when the master subnet manager fails.

3. Install the ebrctl utility on the compute nodes.

Check that ebrctl is listed somewhere in /etc/nova/rootwrap.d/*:

```
$ grep 'ebrctl' /etc/nova/rootwrap.d/*
```

If ebrctl does not appear in any of the rootwrap files, add this to the /etc/nova/rootwrap.d/compute.filters file in the [Filters] section.

```
[Filters]
ebrctl: CommandFilter, ebrctl, root
```

Known limitations

- When using Quality of Service (QoS), max_burst_kbps (burst over max_kbps) is not supported. In addition, max_kbps is rounded to Mbps.
- Security groups are not supported when using SR-IOV, thus, the firewall driver must be disabled. This can be done in the neutron.conf file.

```
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

- SR-IOV is not integrated into the OpenStack Dashboard (horizon). Users must use the CLI or API to configure SR-IOV interfaces.
- Live migration is not supported for instances with SR-IOV ports.

Note: SR-IOV features may require a specific NIC driver version, depending on the vendor. Intel NICs, for example, require ixgbe version 4.4.6 or greater, and ixgbevf version 3.2.2 or greater.

Subnet pools

Subnet pools have been made available since the Kilo release. It is a simple feature that has the potential to improve your workflow considerably. It also provides a building block from which other new features will be built in to OpenStack Networking.

To see if your cloud has this feature available, you can check that it is listed in the supported aliases. You can do this with the OpenStack client.

```
$ openstack extension list | grep subnet_allocation
| Subnet Allocation | subnet_allocation | Enables allocation of subnets
from a subnet pool
|
```

Why you need them

Before Kilo, Networking had no automation around the addresses used to create a subnet. To create one, you had to come up with the addresses on your own without any help from the system. There are valid use cases for this but if you are interested in the following capabilities, then subnet pools might be for you.

First, would not it be nice if you could turn your pool of addresses over to Neutron to take care of? When you need to create a subnet, you just ask for addresses to be allocated from the pool. You do not have to worry about what you have already used and what addresses are in your pool. Subnet pools can do this.

Second, subnet pools can manage addresses across projects. The addresses are guaranteed not to overlap. If the addresses come from an externally routable pool then you know that all of the projects have addresses which are *routable* and unique. This can be useful in the following scenarios.

1. IPv6 since OpenStack Networking has no IPv6 floating IPs.
2. Routing directly to a project network from an external network.

How they work

A subnet pool manages a pool of addresses from which subnets can be allocated. It ensures that there is no overlap between any two subnets allocated from the same pool.

As a regular project in an OpenStack cloud, you can create a subnet pool of your own and use it to manage your own pool of addresses. This does not require any admin privileges. Your pool will not be visible to any other project.

If you are an admin, you can create a pool which can be accessed by any regular project. Being a shared resource, there is a quota mechanism to arbitrate access.

Quotas

Subnet pools have a quota system which is a little bit different than other quotas in Neutron. Other quotas in Neutron count discrete instances of an object against a quota. Each time you create something like a router, network, or a port, it uses one from your total quota.

With subnets, the resource is the IP address space. Some subnets take more of it than others. For example, 203.0.113.0/24 uses 256 addresses in one subnet but 198.51.100.224/28 uses only 16. If address space is limited, the quota system can encourage efficient use of the space.

With IPv4, the `default_quota` can be set to the number of absolute addresses any given project is allowed to consume from the pool. For example, with a quota of 128, I might get 203.0.113.128/26, 203.0.113.224/28, and still have room to allocate 48 more addresses in the future.

With IPv6 it is a little different. It is not practical to count individual addresses. To avoid ridiculously large numbers, the quota is expressed in the number of /64 subnets which can be allocated. For example, with a `default_quota` of 3, I might get 2001:db8:c18e:c05a::/64, 2001:db8:221c:8ef3::/64, and still have room to allocate one more prefix in the future.

Default subnet pools

Beginning with Mitaka, a subnet pool can be marked as the default. This is handled with a new extension.

```
$ openstack extension list | grep default-subnetpools
| Default Subnetpools | default-subnetpools | Provides ability to mark
and use a subnetpool as the default
|
```

An administrator can mark a pool as default. Only one pool from each address family can be marked default.

```
$ openstack subnet pool set --default 74348864-f8bf-4fc0-ab03-81229d189467
```

If there is a default, it can be requested by passing `--use-default-subnetpool` instead of `--subnet-pool SUBNETPOOL`.

Demo

If you have access to an OpenStack Kilo or later based neutron, you can play with this feature now. Give it a try. All of the following commands work equally as well with IPv6 addresses.

First, as admin, create a shared subnet pool:

```
$ openstack subnet pool create --share --pool-prefix 203.0.113.0/24 \
--default-prefix-length 26 demo-subnetpool4
+-----+-----+
| Field      | Value          |
+-----+-----+
| address_scope_id | None           |
| created_at    | 2016-12-14T07:21:26Z |
| default_prefixlen | 26             |
| default_quota  | None           |
| description   |                |
| headers       |                |
| id            | d3aefb76-2527-43d4-bc21-0ec253 |
|                | 908545         |
| ip_version    | 4              |
| is_default    | False          |
| max_prefixlen | 32             |
| min_prefixlen | 8              |
| name          | demo-subnetpool4 |
| prefixes      | 203.0.113.0/24 |
| project_id    | cfd1889ac7d64ad891d4f20aef9f8d |
|                | 7c              |
| revision_number | 1              |
| shared         | True           |
| updated_at    | 2016-12-14T07:21:26Z |
+-----+-----+
```

The `default_prefix_length` defines the subnet size you will get if you do not specify `--prefix-length` when creating a subnet.

Do essentially the same thing for IPv6 and there are now two subnet pools. Regular projects can see them. (the output is trimmed a bit for display)

```
$ openstack subnet pool list
+-----+-----+-----+
| ID      | Name          | Prefixes        |
+-----+-----+-----+
```

2b7cc19f-0114-4e	demo-subnetpool	2001:db8:a583::/48	
f4-ad86-c1bb91fc			
d1f9			
d3aefb76-2527-43	demo-subnetpool4	203.0.113.0/24	
d4-bc21-0ec25390			
8545			
-----+-----+-----+-----			

Now, use them. It is easy to create a subnet from a pool:

\$ openstack subnet create --ip-version 4 --subnet-pool \
demo-subnetpool4 --network demo-network1 demo-subnet1
+-----+-----+-----+-----+
Field Value
+-----+-----+-----+-----+
allocation_pools 203.0.113.194-203.0.113.254
cidr 203.0.113.192/26
created_at 2016-12-14T07:33:13Z
description
dns_nameservers
enable_dhcp True
gateway_ip 203.0.113.193
headers
host_routes
id 8d4fbea3-076c-4c08-b2dd-2d6175115a5e
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name demo-subnet1
network_id 6b377f77-ce00-4ff6-8676-82343817470d
project_id cfd1889ac7d64ad891d4f20aef9f8d7c
revision_number 2
service_types
subnetpool_id d3aefb76-2527-43d4-bc21-0ec253908545
updated_at 2016-12-14T07:33:13Z
+-----+-----+-----+-----+

You can request a specific subnet from the pool. You need to specify a subnet that falls within the pool's prefixes. If the subnet is not already allocated, the request succeeds. You can leave off the IP version because it is deduced from the subnet pool.

\$ openstack subnet create --subnet-pool demo-subnetpool4 \
--network demo-network1 --subnet-range 203.0.113.128/26 subnet2
+-----+-----+-----+-----+
Field Value
+-----+-----+-----+-----+
allocation_pools 203.0.113.130-203.0.113.190
cidr 203.0.113.128/26
created_at 2016-12-14T07:27:40Z
description
dns_nameservers
enable_dhcp True
gateway_ip 203.0.113.129
headers
host_routes
id d32814e3-cf46-4371-80dd-498a80badfba
ip_version 4
+-----+-----+-----+-----+

ipv6_address_mode	None	
ipv6_ra_mode	None	
name	subnet2	
network_id	6b377f77-ce00-4ff6-8676-82343817470d	
project_id	cfd1889ac7d64ad891d4f20aef9f8d7c	
revision_number	2	
service_types		
subnetpool_id	d3aefb76-2527-43d4-bc21-0ec253908545	
updated_at	2016-12-14T07:27:40Z	

If the pool becomes exhausted, load some more prefixes:

```
$ openstack subnet pool set --pool-prefix \
198.51.100.0/24 demo-subnetpool4
$ openstack subnet pool show demo-subnetpool4
+-----+-----+
| Field      | Value   |
+-----+-----+
| address_scope_id | None    |
| created_at     | 2016-12-14T07:21:26Z |
| default_prefixlen | 26      |
| default_quota  | None    |
| description    |         |
| id             | d3aefb76-2527-43d4-bc21-0ec253908545 |
| ip_version     | 4       |
| is_default     | False   |
| max_prefixlen  | 32      |
| min_prefixlen  | 8       |
| name           | demo-subnetpool4 |
| prefixes        | 198.51.100.0/24, 203.0.113.0/24 |
| project_id     | cfd1889ac7d64ad891d4f20aef9f8d7c |
| revision_number | 2       |
| shared          | True    |
| updated_at     | 2016-12-14T07:30:32Z |
+-----+-----+
```

Service subnets

Service subnets enable operators to define valid port types for each subnet on a network without limiting networks to one subnet or manually creating ports with a specific subnet ID. Using this feature, operators can ensure that ports for instances and router interfaces, for example, always use different subnets.

Operation

Define one or more service types for one or more subnets on a particular network. Each service type must correspond to a valid device owner within the port model in order for it to be used.

During IP allocation, the [IPAM](#) driver returns an address from a subnet with a service type matching the port device owner. If no subnets match, or all matching subnets lack available IP addresses, the IPAM driver attempts to use a subnet without any service types to preserve compatibility. If all subnets on a network have a service type, the IPAM driver cannot preserve compatibility. However, this feature enables strict IP allocation from subnets with a matching device owner. If multiple subnets contain the same service type, or a subnet without

a service type exists, the IPAM driver selects the first subnet with a matching service type. For example, a floating IP agent gateway port uses the following selection process:

- network:floatingip_agent_gateway
- None

Note: Ports with the device owner network:dhcp are exempt from the above IPAM logic for subnets with dhcp_enabled set to True. This preserves the existing automatic DHCP port creation behaviour for DHCP-enabled subnets.

Creating or updating a port with a specific subnet skips this selection process and explicitly uses the given subnet.

Usage

Note: Creating a subnet with a service type requires administrative privileges.

Example 1 - Proof-of-concept

This following example is not typical of an actual deployment. It is shown to allow users to experiment with configuring service subnets.

1. Create a network.

```
$ openstack network create demo-net1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| description | |
| headers | |
| id | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| mtu | 1450 |
| name | demo-net1 |
| port_security_enabled | True |
| project_id | a3db43cd0f224242a847ab84d091217d |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 110 |
| router:external | Internal |
| shared | False |
| status | ACTIVE |
| subnets | |
| tags | [] |
+-----+-----+
```

2. Create a subnet on the network with one or more service types. For example, the compute:nova service type enables instances to use this subnet.

```
$ openstack subnet create demo-subnet1 --subnet-range 192.0.2.0/24 \
--service-type 'compute:nova' --network demo-net1
+-----+-----+
| Field | Value |
+-----+-----+
| id | 6e38b23f-0b27-4e3c-8e69-fd23a3df1935 |
| ip_version | 4 |
| cidr | 192.0.2.0/24 |
| name | demo-subnet1 |
| network_id | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| service_types | ['compute:nova'] |
| tenant_id | a8b3054cc1214f18b1186b291525650f |
+-----+-----+
```

3. Optionally, create another subnet on the network with a different service type. For example, the compute:foo arbitrary service type.

```
$ openstack subnet create demo-subnet2 --subnet-range 198.51.100.0/24 \
--service-type 'compute:foo' --network demo-net1
+-----+-----+
| Field | Value |
+-----+-----+
| id | ea139dcf-17a3-4f0a-8cca-dff8b4e03f8a |
| ip_version | 4 |
| cidr | 198.51.100.0/24 |
| name | demo-subnet2 |
| network_id | b5b729d8-31cc-4d2c-8284-72b3291fec02 |
| service_types | ['compute:foo'] |
| tenant_id | a8b3054cc1214f18b1186b291525650f |
+-----+-----+
```

4. Launch an instance using the network. For example, using the cirros image and m1.tiny flavor.

```
$ openstack server create demo-instance1 --flavor m1.tiny \
--image cirros --nic net-id=b5b729d8-31cc-4d2c-8284-72b3291fec02
+-----+
| Field | Value |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | |
| OS-EXT-SRV-ATTR:host | None |
| OS-EXT-SRV-ATTR:hypervisor_hostname | None |
| OS-EXT-SRV-ATTR:instance_name | instance-00000009 |
| OS-EXT-STS:power_state | 0 |
| OS-EXT-STS:task_state | scheduling |
```

OS-EXT-STS:vm_state	building	□
↳		
OS-SRV-USG:launched_at	None	□
↳		
OS-SRV-USG:terminated_at	None	□
↳		
accessIPv4		□
↳		
accessIPv6		□
↳		
addresses		□
↳		
adminPass	Fn85skabdxBL	□
↳		
config_drive		□
↳		
created	2016-09-19T15:07:42Z	□
↳		
flavor	m1.tiny (1)	□
↳		
hostId		□
↳		
id	04222b73-1a6e-4c2a-9af4-ef3d17d521ff	□
↳		
image	cirros (4aaec87d-c655-4856-8618-	□
↳ b2dada3a2b11)		
key_name	None	□
↳		
name	demo-instance1	□
↳		
os-extended-volumes:volumes_attached	[]	□
↳		
progress	0	□
↳		
project_id	d44c19e056674381b86430575184b167	□
↳		
properties		□
↳		
security_groups	[{u'name': u'default'}]	□
↳		
status	BUILD	□
↳		
updated	2016-09-19T15:07:42Z	□
↳		
user_id	331afbeb322d4c559a181e19051ae362	□
↳		
+-----+-----+-----+-----+		
↳ +		

5. Check the instance status. The Networks field contains an IP address from the subnet having the compute:nova service type.

\$ openstack server list			
ID	Name	Status	Networks
↳			□

```
+-----+-----+-----+
| 20181f46-5cd2-4af8-9af0-f4cf5c983008 | demo-instance1 | ACTIVE | demo-net1=192.0.
| 2.3 |
+-----+-----+-----+
|
```

Example 2 - DVR configuration

The following example outlines how you can configure service subnets in a DVR-enabled deployment, with the goal of minimizing public IP address consumption. This example uses three subnets on the same external network:

- 192.0.2.0/24 for instance floating IP addresses
- 198.51.100.0/24 for floating IP agent gateway IPs configured on compute nodes
- 203.0.113.0/25 for all other IP allocations on the external network

This example uses again the private network, demo-net1 (b5b729d8-31cc-4d2c-8284-72b3291fec02) which was created in [Example 1 - Proof-of-concept](#).

1. Create an external network:

```
$ openstack network create --external demo-ext-net
```

2. Create a subnet on the external network for the instance floating IP addresses. This uses the network:floatingip service type.

```
$ openstack subnet create demo-floating-ip-subnet \
--subnet-range 192.0.2.0/24 --no-dhcp \
--service-type 'network:floatingip' --network demo-ext-net
```

3. Create a subnet on the external network for the floating IP agent gateway IP addresses, which are configured by DVR on compute nodes. This will use the network:floatingip_agent_gateway service type.

```
$ openstack subnet create demo-floating-ip-agent-gateway-subnet \
--subnet-range 198.51.100.0/24 --no-dhcp \
--service-type 'network:floatingip_agent_gateway' \
--network demo-ext-net
```

4. Create a subnet on the external network for all other IP addresses allocated on the external network. This will not use any service type. It acts as a fall back for allocations that do not match either of the above two service subnets.

```
$ openstack subnet create demo-other-subnet \
--subnet-range 203.0.113.0/25 --no-dhcp \
--network demo-ext-net
```

5. Create a router:

```
$ openstack router create demo-router
```

6. Add an interface to the router on demo-subnet1:

```
$ openstack router add subnet demo-router demo-subnet1
```

- Set the external gateway for the router, which will create an interface and allocate an IP address on demo-ext-net:

```
$ neutron router-gateway-set demo-router demo-ext-net
```

- Launch an instance on a private network and retrieve the neutron port ID that was allocated. As above, use the cirros image and m1.tiny flavor:

```
$ openstack server create demo-instance1 --flavor m1.tiny \
--image cirros --nic net-id=b5b729d8-31cc-4d2c-8284-72b3291fec02
$ openstack port list --server demo-instance1
+-----+-----+-----+
| ID | Name | MAC Address | Fixed IP |
| Addresses | Status | | |
+-----+-----+-----+
| a752bb24-9bf2-4d37-b9d6-07da69c86f19 | fa:16:3e:99:54:32 | ip_address='203.0.113.130' | |
| | ACTIVE | | |
| | | | |
| | | | |
+-----+-----+-----+
| 6e38b23f-0b27-4e3c-8e69-fd23a3df1935 | | | |
| | | | |
+-----+-----+-----+
```

- Associate a floating IP with the instance port and verify it was allocated an IP address from the correct subnet:

```
$ openstack floating ip create --port \
a752bb24-9bf2-4d37-b9d6-07da69c86f19 demo-ext-net
+-----+-----+
| Field | Value |
+-----+-----+
| fixed_ip_address | 203.0.113.130 |
| floating_ip_address | 192.0.2.12 |
| floating_network_id | 02d236d5-dad9-4082-bb6b-5245f9f84d13 |
| id | f15cae7f-5e05-4b19-bd25-4bb71edcf3de |
| port_id | a752bb24-9bf2-4d37-b9d6-07da69c86f19 |
| project_id | d44c19e056674381b86430575184b167 |
| router_id | 5a8ca19f-3703-4f81-bc29-db6bc2f528d6 |
| status | ACTIVE |
+-----+-----+
```

- As the *admin* user, verify the neutron routers are allocated IP addresses from their correct subnets. Use `openstack port list` to find ports associated with the routers.

First, the router gateway external port:

```
$ neutron port-show f148ffeb-3c26-4067-bc5f-5c3dfddae2f5
+-----+
| Field | Value |
+-----+
| admin_state_up | UP |
+-----+
```

device_id	5a8ca19f-3703-4f81-bc29-db6bc2f528d6	¶
device_owner	network:router_gateway	¶
extra_dhcp_opts		¶
fixed_ips	ip_address='203.0.113.11',	¶
	subnet_id='67c251d9-2b7a-4200-99f6-e13785b0334d'	¶
id	f148ffeb-3c26-4067-bc5f-5c3dfddae2f5	¶
mac_address	fa:16:3e:2c:0f:69	¶
network_id	02d236d5-dad9-4082-bb6b-5245f9f84d13	¶
project_id		¶
status	ACTIVE	¶
+-----+-----+-----+		

Second, the router floating IP agent gateway external port:

\$ neutron port-show a2d1e756-8ae1-4f96-9aa1-e7ea16a6a68a		
+-----+-----+	+-----+-----+	+-----+-----+
Field	Value	¶
admin_state_up	UP	¶
device_id	3d0c98eb-bca3-45cc-8aa4-90ae3deb0844	¶
device_owner	network:floatingip_agent_gateway	¶
extra_dhcp_opts		¶
fixed_ips	ip_address='198.51.100.10',	¶
	subnet_id='67c251d9-2b7a-4200-99f6-e13785b0334d'	¶
id	a2d1e756-8ae1-4f96-9aa1-e7ea16a6a68a	¶
mac_address	fa:16:3e:f4:5d:fa	¶
network_id	02d236d5-dad9-4082-bb6b-5245f9f84d13	¶
project_id		¶
status	ACTIVE	¶
+-----+-----+-----+		

Trunking

The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting it to a single port.

Operation

Network trunking consists of a service plug-in and a set of drivers that manage trunks on different layer-2 mechanism drivers. Users can create a port, associate it with a trunk, and launch an instance on that port. Users can dynamically attach and detach additional networks without disrupting operation of the instance.

Every trunk has a parent port and can have any number of subports. The parent port is the port that the trunk is associated with. Users create instances and specify the parent port of the trunk when launching instances attached to a trunk.

The network presented by the subport is the network of the associated port. When creating a subport, a `segmentation-id` may be required by the driver. `segmentation-id` defines the segmentation ID on which the subport network is presented to the instance. `segmentation-type` may be required by certain drivers like OVS, although at this time only `vlan` is supported as a `segmentation-type`.

Note: The `segmentation-type` and `segmentation-id` parameters are optional in the Networking API. However, all drivers as of the Newton release require both to be provided when adding a subport to a trunk. Future drivers may be implemented without this requirement.

The `segmentation-type` and `segmentation-id` specified by the user on the subports is intentionally decoupled from the `segmentation-type` and ID of the networks. For example, it is possible to configure the Networking service with `tenant_network_types = vxlan` and still create subports with `segmentation_type = vlan`. The Networking service performs remapping as necessary.

Example configuration

The ML2 plug-in supports trunking with the following mechanism drivers:

- Open vSwitch (OVS)
- Linux bridge
- Open Virtual Network (OVN)

When using a `segmentation-type` of `vlan`, the OVS and Linux bridge drivers present the network of the parent port as the untagged VLAN and all subports as tagged VLANs.

Controller node

- In the `neutron.conf` file, enable the trunk service plug-in:

```
[DEFAULT]
service_plugins = trunk
```

Verify service operation

1. Source the administrative project credentials and list the enabled extensions.
2. Use the command **openstack extension list --network** to verify that the Trunk Extension and Trunk port details extensions are enabled.

Workflow

At a high level, the basic steps to launching an instance on a trunk are the following:

1. Create networks and subnets for the trunk and subports
2. Create the trunk
3. Add subports to the trunk
4. Launch an instance on the trunk

Create networks and subnets for the trunk and subports

Create the appropriate networks for the trunk and subports that will be added to the trunk. Create subnets on these networks to ensure the desired layer-3 connectivity over the trunk.

Create the trunk

- Create a parent port for the trunk.

```
$ openstack port create --network project-net-A trunk
+-----+
| Field          | Value
+-----+
| admin_state_up | UP
| binding_vif_type | unbound
| binding_vnic_type | normal
| fixed_ips      | ip_address='192.0.2.7', subnet_id='8b957198-d3cf-4953-8449-
| ad4e4dd712cc' |
| id             | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
| mac_address    | fa:16:3e:dd:c4:d1
| name           | trunk
| network_id     | 1b47d3e7-cda5-48e4-b0c8-d20bd7e35f55
+-----+
```

- Create the trunk using `--parent-port` to reference the port from the previous step:

```
$ openstack network trunk create --parent-port 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
→trunk
+-----+-----+
| Field      | Value
+-----+-----+
| admin_state_up | UP
| id          | fdf02fcb-1844-45f1-9d9b-e4c2f522c164
| name        | trunk
| port_id     | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
| sub_ports   |
+-----+-----+
```

Add subports to the trunk

Subports can be added to a trunk in two ways: creating the trunk with subports or adding subports to an existing trunk.

- Create trunk with subports:

This method entails creating the trunk with subports specified at trunk creation.

```
$ openstack port create --network project-net-A trunk-parent
+-----+
| Field      | Value
+-----+
| admin_state_up | UP
| binding_vif_type | unbound
| binding_vnic_type | normal
| fixed_ips      | ip_address='192.0.2.7', subnet_id='8b957198-d3cf-4953-8449-
ad4e4dd712cc'
| id          | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
| mac_address   | fa:16:3e:dd:c4:d1
| name        | trunk-parent
| network_id    | 1b47d3e7-cda5-48e4-b0c8-d20bd7e35f55
+-----+
$ openstack port create --network trunked-net subport1
+-----+
| Field      | Value
+-----+
|
```

```
| admin_state_up | UP
+-----+
| binding_vif_type | unbound
+-----+
| binding_vnic_type | normal
+-----+
| fixed_ips | ip_address='192.0.2.8', subnet_id='2a860e2c-922b-437b-a149-
+-----+
| id | 91f9dde8-80a4-4506-b5da-c287feb8f5d8
+-----+
| mac_address | fa:16:3e:ba:f0:4d
+-----+
| name | subport1
+-----+
| network_id | aef78ec5-16e3-4445-b82d-b2b98c6a86d9
+-----+
+-----+
$ openstack network trunk create \
--parent-port 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38 \
--subport port=91f9dde8-80a4-4506-b5da-c287feb8f5d8, \
segmentation-type=vlan,segmentation-id=100
+-----+
| Field | Value
+-----+
| admin_state_up | UP
+-----+
| id | 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
+-----+
| name | trunk
+-----+
| port_id | 73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
+-----+
| sub_ports | port_id='73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38', segmentation_id=
+-----+
| segmentation-type='vlan' |
+-----+
| segmentation-id=100 | 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
+-----+
```

- Add subports to an existing trunk:

This method entails creating a trunk, then adding subports to the trunk after it has already been created.

```
$ openstack network trunk set --subport \
port=91f9dde8-80a4-4506-b5da-c287feb8f5d8, \
segmentation-type=vlan, \
segmentation-id=100 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
```

Note: The command provides no output.

```
$ openstack network trunk show 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
+-----+
```

Field	Value
admin_state_up	UP
id	61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
name	trunk
port_id	73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
sub_ports	port_id='73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38', segmentation_id='100', segmentation_type='vlan'

Launch an instance on the trunk

- Show trunk details to get the port_id of the trunk.

\$ openstack network trunk show 61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3	
Field	Value
admin_state_up	UP
id	61d8e620-fe3a-4d8f-b9e6-e1b0dea6d9e3
name	trunk
port_id	73fb9d54-43a7-4bb1-a8dc-569e0e0a0a38
sub_ports	

- Launch the instance by specifying port_id using the value of port_id from the trunk details. Launching an instance on a subport is not supported.

Using trunks and subports inside an instance

When configuring instances to use a subport, ensure that the interface on the instance is set to use the MAC address assigned to the port by the Networking service. Instances are not made aware of changes made to the trunk after they are active. For example, when a subport with a segmentation-type of `vlan` is added to a trunk, any operations specific to the instance operating system that allow the instance to send and receive traffic on the new VLAN must be handled outside of the Networking service.

When creating subports, the MAC address of the trunk parent port can be set on the subport. This will allow VLAN subinterfaces inside an instance launched on a trunk to be configured without explicitly setting a MAC address. Although unique MAC addresses can be used for subports, this can present issues with ARP spoof protections and the native OVS firewall driver. If the native OVS firewall driver is to be used, we recommend that the MAC address of the parent port be re-used on all subports.

Trunk states

- ACTIVE

The trunk is ACTIVE when both the logical and physical resources have been created. This means that all operations within the Networking and Compute services have completed and the trunk is ready for use.

- DOWN

A trunk is DOWN when it is first created without an instance launched on it, or when the instance associated with the trunk has been deleted.

- DEGRADED

A trunk can be in a DEGRADED state when a temporary failure during the provisioning process is encountered. This includes situations where a subport add or remove operation fails. When in a degraded state, the trunk is still usable and some subports may be usable as well. Operations that cause the trunk to go into a DEGRADED state can be retried to fix temporary failures and move the trunk into an ACTIVE state.

- ERROR

A trunk is in ERROR state if the request leads to a conflict or an error that cannot be fixed by retrying the request. The ERROR status can be encountered if the network is not compatible with the trunk configuration or the binding process leads to a persistent failure. When a trunk is in ERROR state, it must be brought to a sane state (ACTIVE), or else requests to add subports will be rejected.

- BUILD

A trunk is in BUILD state while the resources associated with the trunk are in the process of being provisioned. Once the trunk and all of the subports have been provisioned successfully, the trunk transitions to ACTIVE. If there was a partial failure, the trunk transitions to DEGRADED.

When `admin_state` is set to DOWN, the user is blocked from performing operations on the trunk. `admin_state` is set by the user and should not be used to monitor the health of the trunk.

Limitations and issues

- See [bugs](#) for more information.

Note: For general configuration, see the [Configuration Reference](#).

Deployment examples

The following deployment examples provide building blocks of increasing architectural complexity using the Networking service reference architecture which implements the Modular Layer 2 (ML2) plug-in and either the Open vSwitch (OVS) or Linux bridge mechanism drivers. Both mechanism drivers support the same basic features such as provider networks, self-service networks, and routers. However, more complex features often require a particular mechanism driver. Thus, you should consider the requirements (or goals) of your cloud before choosing a mechanism driver.

After choosing a [mechanism driver](#), the deployment examples generally include the following building blocks:

1. Provider (public/external) networks using IPv4 and IPv6
2. Self-service (project/private/internal) networks including routers using IPv4 and IPv6
3. High-availability features
4. Other features such as BGP dynamic routing

Prerequisites

Prerequisites, typically hardware requirements, generally increase with each building block. Each building block depends on proper deployment and operation of prior building blocks. For example, the first building block (provider networks) only requires one controller and two compute nodes, the second building block (self-service networks) adds a network node, and the high-availability building blocks typically add a second network node for a total of five nodes. Each building block could also require additional infrastructure or changes to existing infrastructure such as networks.

For basic configuration of prerequisites, see the [Ocata Install Tutorials and Guides](#).

Note: Example commands using the `openstack` client assume version 3.2.0 or higher.

Nodes

The deployment examples refer one or more of the following nodes:

- Controller: Contains control plane components of OpenStack services and their dependencies.
 - Two network interfaces: management and provider.
 - Operational SQL server with databases necessary for each OpenStack service.
 - Operational message queue service.
 - Operational OpenStack Identity (keystone) service.
 - Operational OpenStack Image Service (glance).
 - Operational management components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
 - OpenStack Networking (neutron) server service and ML2 plug-in.
- Network: Contains the OpenStack Networking service layer-3 (routing) component. High availability options may include additional components.
 - Three network interfaces: management, overlay, and provider.
 - OpenStack Networking layer-2 (switching) agent, layer-3 agent, and any dependencies.
- Compute: Contains the hypervisor component of the OpenStack Compute service and the OpenStack Networking layer-2, DHCP, and metadata components. High-availability options may include additional components.
 - Two network interfaces: management and provider.
 - Operational hypervisor components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
 - OpenStack Networking layer-2 agent, DHCP agent, metadata agent, and any dependencies.

Each building block defines the quantity and types of nodes including the components on each node.

Note: You can virtualize these nodes for demonstration, training, or proof-of-concept purposes. However, you must use physical hosts for evaluation of performance or scaling.

Networks and network interfaces

The deployment examples refer to one or more of the following networks and network interfaces:

- Management: Handles API requests from clients and control plane traffic for OpenStack services including their dependencies.
- Overlay: Handles self-service networks using an overlay protocol such as VXLAN or GRE.
- Provider: Connects virtual and physical networks at layer-2. Typically uses physical network infrastructure for switching/routing traffic to external networks such as the Internet.

Note: For best performance, 10+ Gbps physical network infrastructure should support jumbo frames.

For illustration purposes, the configuration examples typically reference the following IP address ranges:

- Management network: 10.0.0.0/24
- Overlay (tunnel) network: 10.0.1.0/24
- Provider network 1:
 - IPv4: 203.0.113.0/24
 - IPv6: fd00:203:0:113::/64
- Provider network 2:
 - IPv4: 192.0.2.0/24
 - IPv6: fd00:192:0:2::/64
- Self-service networks:
 - IPv4: 192.168.0.0/16 in /24 segments
 - IPv6: fd00:192:168::/48 in /64 segments

You may change them to work with your particular network infrastructure.

Mechanism drivers

Linux bridge mechanism driver

The Linux bridge mechanism driver uses only Linux bridges and veth pairs as interconnection devices. A layer-2 agent manages Linux bridges on each compute node and any other node that provides layer-3 (routing), DHCP, metadata, or other network services.

Linux bridge: Provider networks

The provider networks architecture example provides layer-2 connectivity between instances and the physical network infrastructure using VLAN (802.1q) tagging. It supports one untagged (flat) network and up to 4095 tagged (VLAN) networks. The actual quantity of VLAN networks depends on the physical network infrastructure. For more information on provider networks, see [Provider networks](#).

Prerequisites

One controller node with the following components:

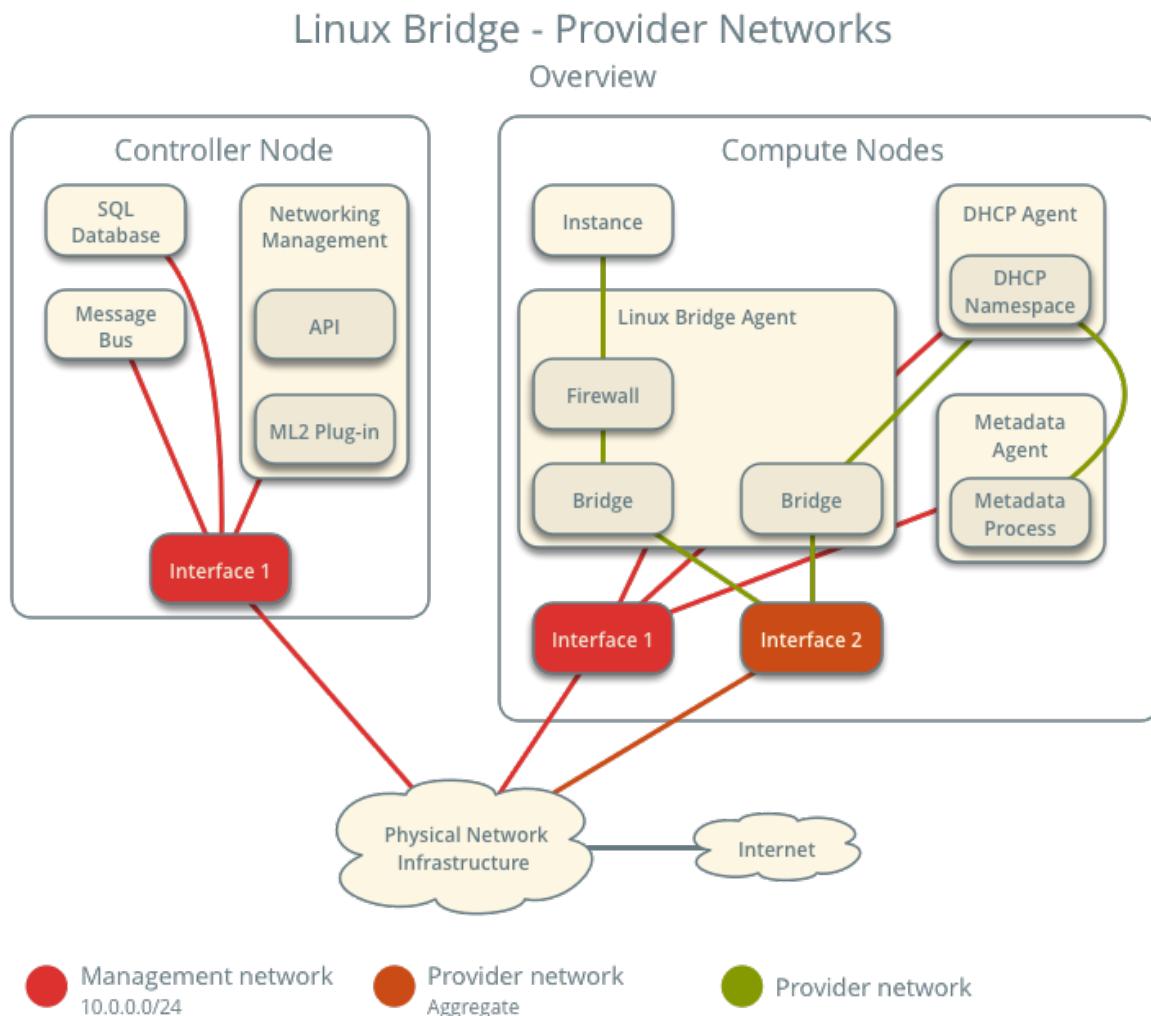
- Two network interfaces: management and provider.
- OpenStack Networking server service and ML2 plug-in.

Two compute nodes with the following components:

- Two network interfaces: management and provider.
- OpenStack Networking Linux bridge layer-2 agent, DHCP agent, metadata agent, and any dependencies.

Note: Larger deployments typically deploy the DHCP and metadata agents on a subset of compute nodes to increase performance and redundancy. However, too many agents can overwhelm the message bus. Also, to further simplify any deployment, you can omit the metadata agent and use a configuration drive to provide metadata to instances.

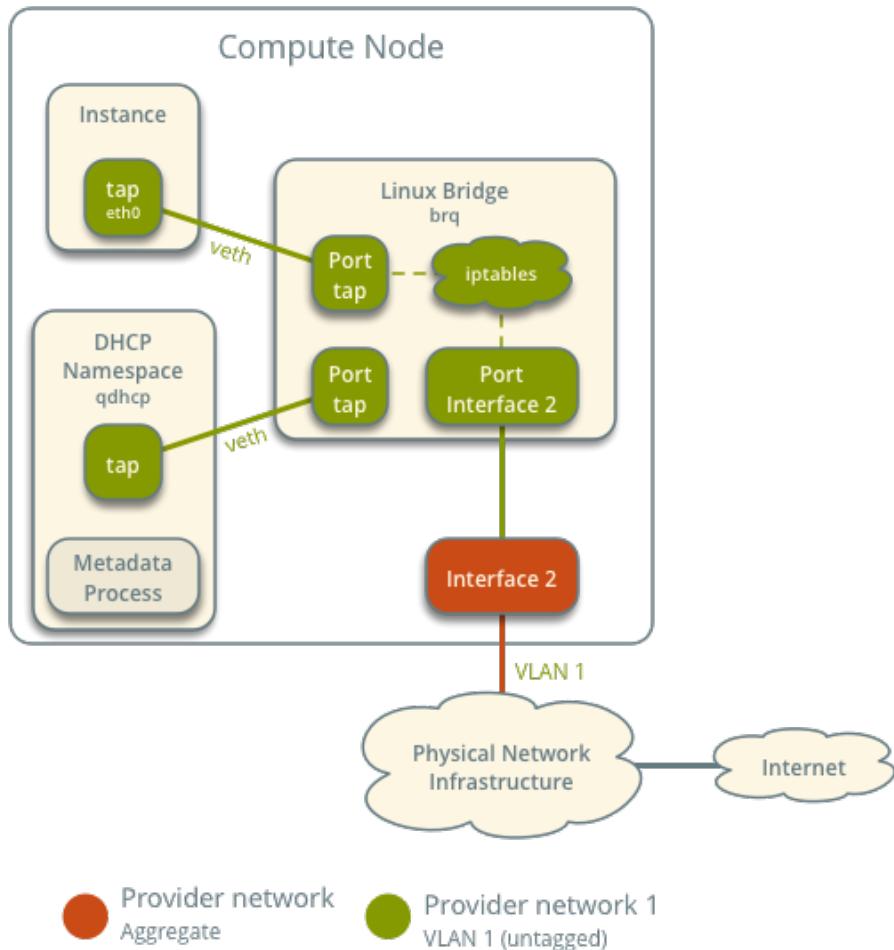
Architecture



The following figure shows components and connectivity for one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the provider physical network interface.

Linux Bridge - Provider Networks

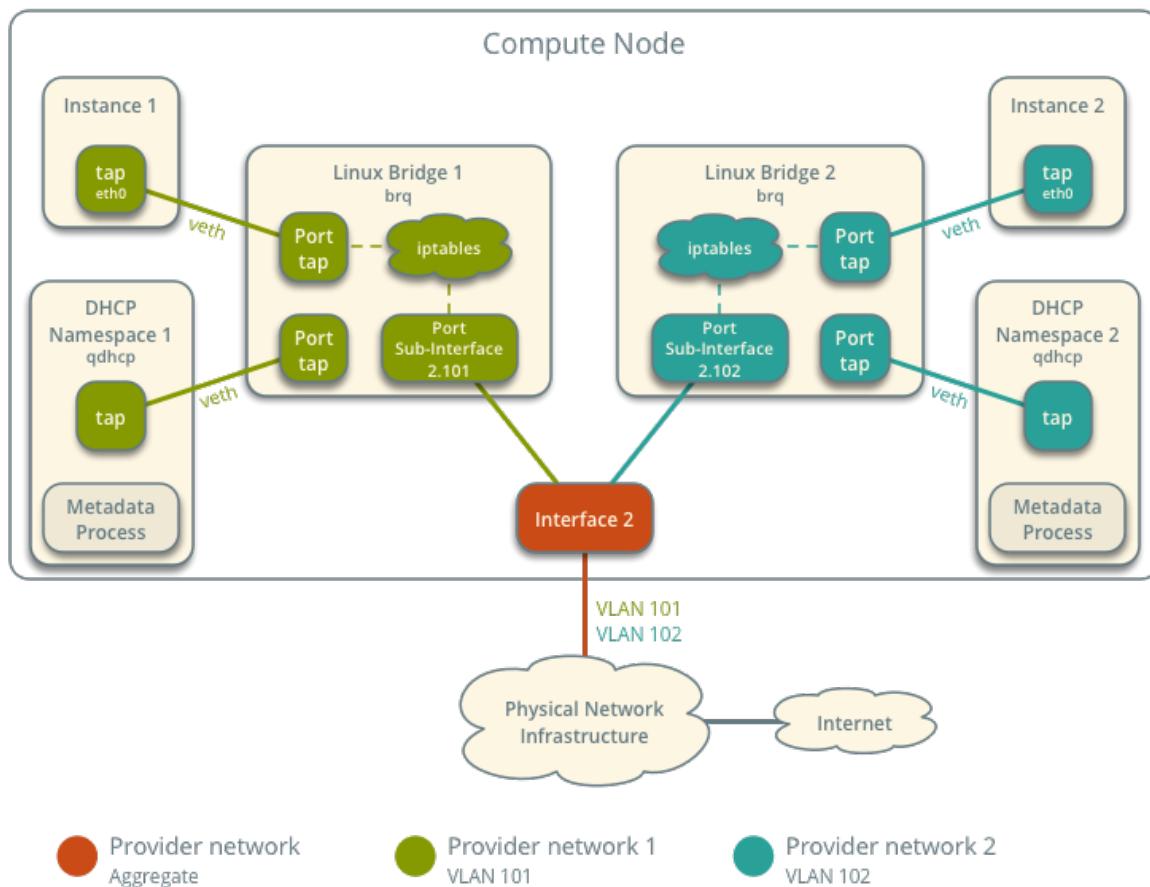
Components and Connectivity



The following figure describes virtual connectivity among components for two tagged (VLAN) networks. Essentially, each network uses a separate bridge that contains a port on the VLAN sub-interface on the provider physical network interface. Similar to the single untagged network case, the DHCP agent may reside on a different compute node.

Linux Bridge - Provider Networks

Components and Connectivity



Note: These figures omit the controller node because it does not handle instance network traffic.

Example configuration

Use the following example configuration as a template to deploy provider networks in your environment.

Controller node

1. Install the Networking service components that provides the `neutron-server` service and ML2 plug-in.
2. In the `neutron.conf` file:
 - Configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
```

```
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_auth_token], [nova], and [agent] sections.

- Disable service plug-ins because provider networks do not require any. However, this breaks portions of the dashboard that manage the Networking service. See the [Ocata Install Tutorials and Guides](#) for more information.

```
[DEFAULT]
service_plugins =
```

- Enable two DHCP agents per network so both compute nodes can provide DHCP service provider networks.

```
[DEFAULT]
dhcp_agents_per_network = 2
```

- If necessary, [configure MTU](#).

3. In the ml2_conf.ini file:

- Configure drivers and network types:

```
[ml2]
type_drivers = flat,vlan
tenant_network_types =
mechanism_drivers = linuxbridge
extension_drivers = port_security
```

- Configure network mappings:

```
[ml2_type_flat]
flat_networks = provider

[ml2_type_vlan]
network_vlan_ranges = provider
```

Note: The tenant_network_types option contains no value because the architecture does not support self-service networks.

Note: The provider value in the network_vlan_ranges option lacks VLAN ID ranges to support use of arbitrary VLAN IDs.

4. Populate the database.

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

5. Start the following services:

- Server

Compute nodes

1. Install the Networking service Linux bridge layer-2 agent.
2. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_auth_token], [nova], and [agent] sections.

3. In the `linuxbridge_agent.ini` file, configure the Linux bridge agent:

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE

[vxlan]
enable_vxlan = False

[securitygroup]
firewall_driver = iptables
```

Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

4. In the `dhcp_agent.ini` file, configure the DHCP agent:

```
[DEFAULT]
interface_driver = linuxbridge
enable_isolated_metadata = True
force_metadata = True
```

Note: The force_metadata option forces the DHCP agent to provide a host route to the metadata service on 169.254.169.254 regardless of whether the subnet contains an interface on a router, thus maintaining similar and predictable metadata behavior among subnets.

5. In the `metadata_agent.ini` file, configure the metadata agent:

```
[DEFAULT]
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

The value of `METADATA_SECRET` must match the value of the same option in the `[neutron]` section of the `nova.conf` file.

6. Start the following services:

- Linux bridge agent
- DHCP agent
- Metadata agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents:

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability |
+-----+-----+-----+-----+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent | compute2 | □
| 188945d1-9e70-4803-a276-df924e0788a4 | Linux bridge agent | compute1 | □
| e76c440d-d5f6-4316-a674-d689630b629e | DHCP agent | compute1 | nova | □
| e67367de-6657-11e6-86a4-931cd04404bb | DHCP agent | compute2 | nova | □
| e8174cae-6657-11e6-89f0-534ac6d0cb5c | Metadata agent | compute1 | □
| ece49ec6-6657-11e6-bafb-c7560f19197d | Metadata agent | compute2 | □
+-----+-----+-----+-----+
```

Create initial networks

The configuration supports one flat or multiple VLAN provider networks. For simplicity, the following procedure creates one flat provider network.

1. Source the administrative project credentials.
2. Create a flat network.

```
$ openstack network create --share --provider-physical-network provider \
    --provider-network-type flat provider1
+-----+-----+
| Field          | Value   |
+-----+-----+
| admin_state_up | UP      |
| mtu            | 1500    |
| name           | provider1 |
| port_security_enabled | True    |
| provider:network_type | flat    |
| provider:physical_network | provider |
| provider:segmentation_id | None    |
| router:external | Internal |
| shared          | True    |
| status          | ACTIVE  |
+-----+-----+
```

Note: The share option allows any project to use this network. To limit access to provider networks, see [Role-Based Access Control \(RBAC\)](#).

Note: To create a VLAN network instead of a flat network, change `--provider:network_type flat` to `--provider-network-type vlan` and add `--provider-segment` with a value referencing the VLAN ID.

3. Create a IPv4 subnet on the provider network.

```
$ openstack subnet create --subnet-range 203.0.113.0/24 --gateway 203.0.113.1 \
    --network provider1 --allocation-pool start=203.0.113.11,end=203.0.113.250 \
    --dns-nameserver 8.8.4.4 provider1-v4
+-----+-----+
| Field          | Value   |
+-----+-----+
| allocation_pools | 203.0.113.11-203.0.113.250 |
| cidr           | 203.0.113.0/24 |
| dns_nameservers | 8.8.4.4   |
| enable_dhcp    | True     |
| gateway_ip     | 203.0.113.1 |
| ip_version     | 4        |
| name           | provider1-v4 |
+-----+-----+
```

Note: Enabling DHCP causes the Networking service to provide DHCP which can interfere with existing DHCP services on the physical network infrastructure.

4. Create a IPv6 subnet on the provider network.

```
$ openstack subnet create --subnet-range fd00:203:0:113::/64 --gateway fd00:203:0:113::1 \
--ip-version 6 --ipv6-address-mode slaac --network provider1 \
--dns-nameserver 2001:4860:4860::8844 provider1-v6
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr | fd00:203:0:113::/64 |
| dns_nameservers | 2001:4860:4860::8844 |
| enable_dhcp | True |
| gateway_ip | fd00:203:0:113::1 |
| ip_version | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode | None |
| name | provider1-v6 |
+-----+-----+
```

Note: The Networking service uses the layer-3 agent to provide router advertisement. Provider networks rely on physical network infrastructure for layer-3 services rather than the layer-3 agent. Thus, the physical network infrastructure must provide router advertisement on provider networks for proper operation of IPv6.

Verify network operation

1. On each compute node, verify creation of the qdhcp namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
```

2. Source a regular (non-administrative) project credentials.
3. Create the appropriate security group rules to allow ping and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+-----+-----+
| Field | Value |
+-----+-----+
| direction | ingress |
| ethertype | IPv4 |
| protocol | icmp |
| remote_ip_prefix | 0.0.0.0/0 |
+-----+-----+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-icmp default
+-----+-----+
| Field | Value |
+-----+-----+
| direction | ingress |
| ethertype | IPv6 |
| protocol | ipv6-icmp |
+-----+-----+
```

```
$ openstack security group rule create --proto tcp --dst-port 22 default
+-----+-----+
| Field | Value |
+-----+-----+
| direction | ingress |
| ethertype | IPv4 |
| port_range_max | 22 |
| port_range_min | 22 |
| protocol | tcp |
| remote_ip_prefix | 0.0.0.0/0 |
+-----+-----+

$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22
↪default
+-----+-----+
| Field | Value |
+-----+-----+
| direction | ingress |
| ethertype | IPv6 |
| port_range_max | 22 |
| port_range_min | 22 |
| protocol | tcp |
+-----+-----+
```

4. Launch an instance with an interface on the provider network. For example, a Cirros image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros \
--nic net-id=NETWORK_ID provider-instance1
```

Replace NETWORK_ID with the ID of the provider network.

5. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID | Name | Status | Networks | Image Name |
+-----+-----+-----+
| 018e0ae2-b43c-4271-a78d-62653dd03285 | provider-instance1 | ACTIVE | provider1=203.0.113.13, fd00:203:0:113:f816:3eff:fe58:be4e | cirros |
+-----+-----+-----+
```

6. On the controller node or any host with access to the provider network, ping the IPv4 and IPv6 addresses of the instance.

```
$ ping -c 4 203.0.113.13
PING 203.0.113.13 (203.0.113.13) 56(84) bytes of data.
64 bytes from 203.0.113.13: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.13: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.13: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.13: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.13 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms

$ ping6 -c 4 fd00:203:0:113:f816:3eff:fe58:be4e
PING fd00:203:0:113:f816:3eff:fe58:be4e(fd00:203:0:113:f816:3eff:fe58:be4e) 56 data bytes
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=1 ttl=64 time=1.25 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=2 ttl=64 time=0.683 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=3 ttl=64 time=0.762 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=4 ttl=64 time=0.486 ms

--- fd00:203:0:113:f816:3eff:fe58:be4e ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.486/0.796/1.253/0.282 ms
```

7. Obtain access to the instance.
8. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network 1 (VLAN)
 - VLAN ID 101 (tagged)
 - IP address ranges 203.0.113.0/24 and fd00:203:0:113::/64
 - Gateway (via physical network infrastructure)
 - * IP addresses 203.0.113.1 and fd00:203:0:113:0::1
- Provider network 2 (VLAN)
 - VLAN ID 102 (tagged)
 - IP address range 192.0.2.0/24 and fd00:192:0:2::/64
 - Gateway
 - * IP addresses 192.0.2.1 and fd00:192:0:2::1
- Instance 1
 - IP addresses 203.0.113.101 and fd00:203:0:113:0::101
- Instance 2
 - IP addresses 192.0.2.101 and fd00:192:0:2:0::101

North-south scenario: Instance with a fixed IP address

- The instance resides on compute node 1 and uses provider network 1.

- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1.

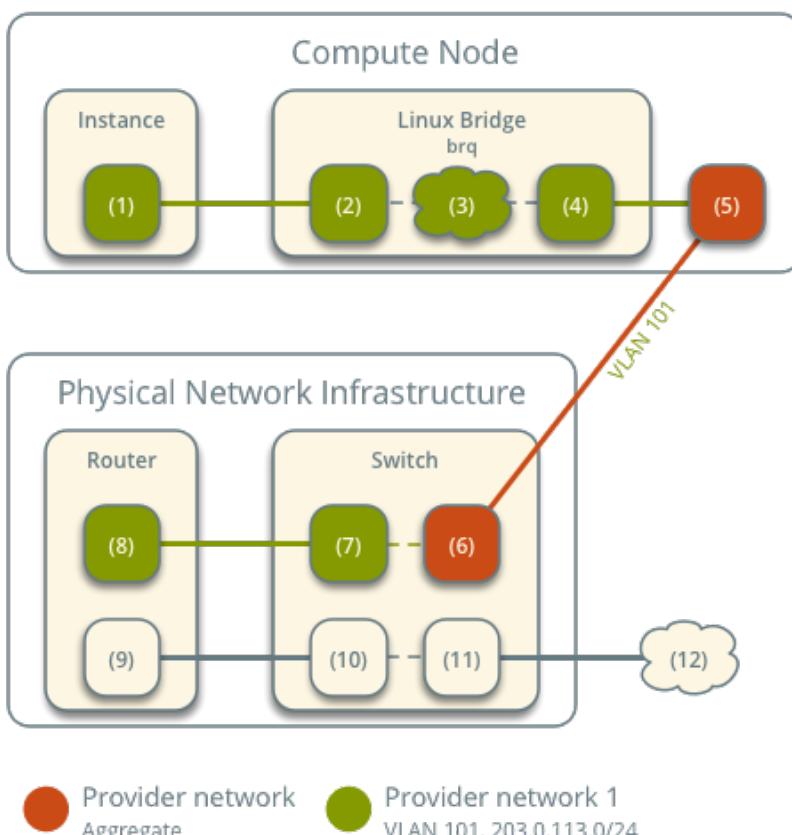
1. The instance interface (1) forwards the packet to the provider bridge instance port (2) via veth pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (7).
2. The router routes the packet from the provider network (8) to the external network (9) and forwards the packet to the switch (10).
3. The switch forwards the packet to the external network (11).
4. The external network (12) receives the packet.

Linux Bridge - Provider Networks

Network Traffic Flow - North/South Scenario



Note: Return traffic follows similar steps in reverse.

East-west scenario 1: Instances on the same network

Instances on the same network communicate directly between compute nodes containing those instances.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 2 and uses provider network 1.
- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the provider bridge instance port (2) via veth pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

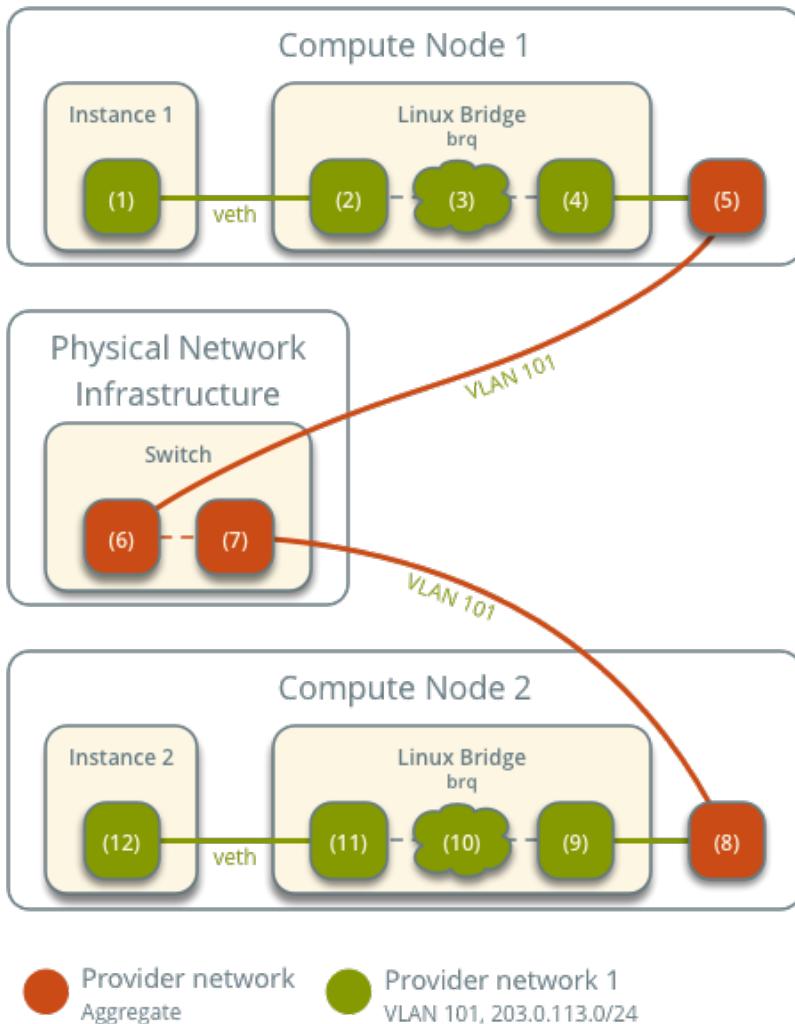
1. The switch forwards the packet from compute node 1 to compute node 2 (7).

The following steps involve compute node 2:

1. The physical network interface (8) removes VLAN tag 101 from the packet and forwards it to the VLAN sub-interface port (9) on the provider bridge.
2. Security group rules (10) on the provider bridge handle firewalling and connection tracking for the packet.
3. The provider bridge instance port (11) forwards the packet to the instance 2 interface (12) via veth pair.

Linux Bridge - Provider Networks

Network Traffic Flow - East/West Scenario 1



Note: Return traffic follows similar steps in reverse.

East-west scenario 2: Instances on different networks

Instances communicate via router on the physical network infrastructure.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 1 and uses provider network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VLAN tagging enables multiple logical layer-2 networks to use the same physical layer-2 network.

The following steps involve the compute node:

1. The instance 1 interface (1) forwards the packet to the provider bridge instance port (2) via veth pair.
2. Security group rules (3) on the provider bridge handle firewalling and connection tracking for the packet.
3. The VLAN sub-interface port (4) on the provider bridge forwards the packet to the physical network interface (5).
4. The physical network interface (5) adds VLAN tag 101 to the packet and forwards it to the physical network infrastructure switch (6).

The following steps involve the physical network infrastructure:

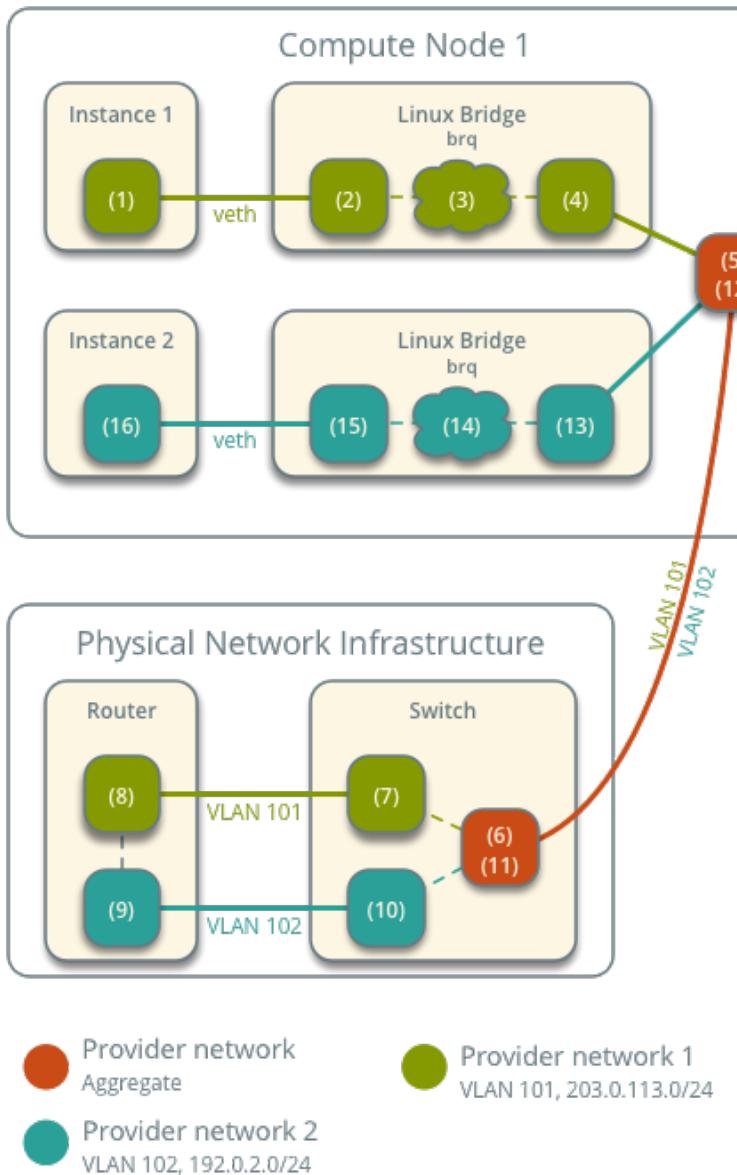
1. The switch removes VLAN tag 101 from the packet and forwards it to the router (7).
2. The router routes the packet from provider network 1 (8) to provider network 2 (9).
3. The router forwards the packet to the switch (10).
4. The switch adds VLAN tag 102 to the packet and forwards it to compute node 1 (11).

The following steps involve the compute node:

1. The physical network interface (12) removes VLAN tag 102 from the packet and forwards it to the VLAN sub-interface port (13) on the provider bridge.
2. Security group rules (14) on the provider bridge handle firewalling and connection tracking for the packet.
3. The provider bridge instance port (15) forwards the packet to the instance 2 interface (16) via veth pair.

Linux Bridge - Provider Networks

Network Traffic Flow - East/West Scenario 2



Note: Return traffic follows similar steps in reverse.

Linux bridge: Self-service networks

This architecture example augments [Linux bridge: Provider networks](#) to support a nearly limitless quantity of entirely virtual networks. Although the Networking service supports VLAN self-service networks, this example focuses on VXLAN self-service networks. For more information on self-service networks, see [Self-service networks](#).

Note: The Linux bridge agent lacks support for other overlay protocols such as GRE and Geneve.

Prerequisites

Add one network node with the following components:

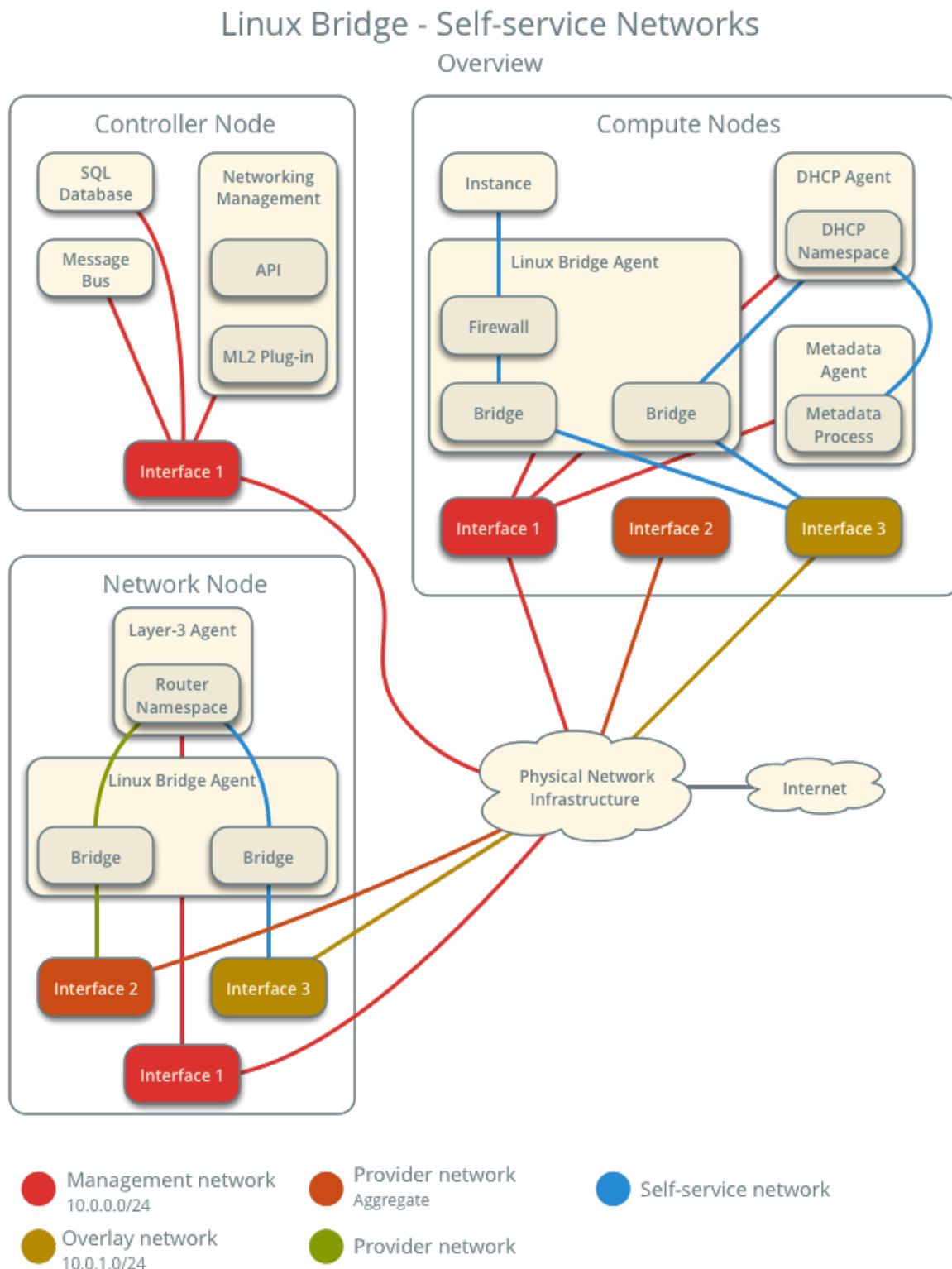
- Three network interfaces: management, provider, and overlay.
- **OpenStack Networking Linux bridge layer-2 agent, layer-3 agent, and any dependencies.**

Modify the compute nodes with the following components:

- Add one network interface: overlay.

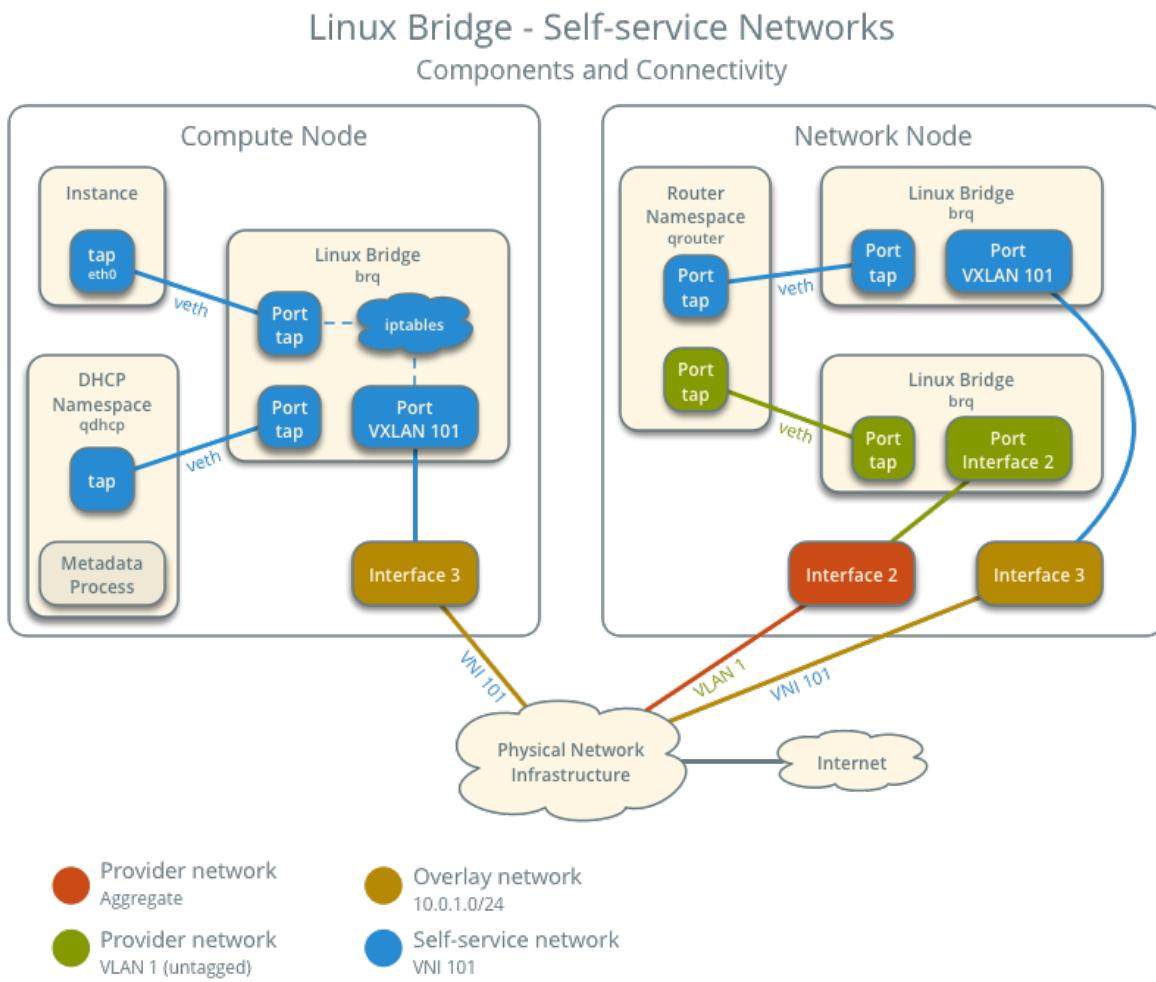
Note: You can keep the DHCP and metadata agents on each compute node or move them to the network node.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) provider network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace

and Linux bridge with a port on the overlay physical network interface.



Example configuration

Use the following example configuration as a template to add support for self-service networks to an existing operational environment that supports provider networks.

Controller node

1. In the `neutron.conf` file:

- Enable routing and allow overlapping IP address ranges.

```
[DEFAULT]
service_plugins = router
allow_overlapping_ips = True
```

2. In the `ml2_conf.ini` file:

- Add vxlan to type drivers and project network types.

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
```

- Enable the layer-2 population mechanism driver.

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

- Configure the VXLAN network ID (VNI) range.

```
[ml2_type_vxlan]
vni_ranges = VNI_START:VNI_END
```

Replace VNI_START and VNI_END with appropriate numerical values.

3. Restart the following services:

- Server

Network node

1. Install the Networking service layer-3 agent.
2. In the neutron.conf file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authhtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_authhtoken], [nova], and [agent] sections.

3. In the linuxbridge_agent.ini file, configure the layer-2 agent.

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE

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

```
[securitygroup]
firewall_driver = iptables
```

Replace PROVIDER_INTERFACE with the name of the underlying interface that handles provider networks. For example, eth1.

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the interface that handles VXLAN overlays for self-service networks.

4. In the l3_agent.ini file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = linuxbridge
external_network_bridge =
```

Note: The external_network_bridge option intentionally contains no value.

5. Start the following services:

- Linux bridge agent
- Layer-3 agent

Compute nodes

1. In the linuxbridge_agent.ini file, enable VXLAN support including layer-2 population.

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

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the interface that handles VXLAN overlays for self-service networks.

2. Restart the following services:

- Linux bridge agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID          | Agent Type      | Host       | Availability |
| Zone | Alive | State | Binary   |           |
+-----+-----+-----+-----+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent | compute2 |          |
|           | True  | UP    | neutron-linuxbridge-agent |
```

188945d1-9e70-4803-a276-df924e0788a4 Linux bridge agent compute1	
↳ True UP neutron-linuxbridge-agent	
e76c440d-d5f6-4316-a674-d689630b629e DHCP agent compute1 nova	
↳ True UP neutron-dhcp-agent	
e67367de-6657-11e6-86a4-931cd04404bb DHCP agent compute2 nova	
↳ True UP neutron-dhcp-agent	
e8174cae-6657-11e6-89f0-534ac6d0cb5c Metadata agent compute1	
↳ True UP neutron-metadata-agent	
ece49ec6-6657-11e6-bafb-c7560f19197d Metadata agent compute2	
↳ True UP neutron-metadata-agent	
598f6357-4331-4da5-a420-0f5be000bec9 L3 agent network1 nova	
↳ True UP neutron-l3-agent	
f4734e0f-bcd5-4922-a19d-e31d56b0a7ae Linux bridge agent network1	
↳ True UP neutron-linuxbridge-agent	
+-----+-----+-----+-----+	+-----+-----+-----+
↳ +-----+-----+-----+-----+	+-----+-----+-----+

Create initial networks

The configuration supports multiple VXLAN self-service networks. For simplicity, the following procedure creates one self-service network and a router with a gateway on the flat provider network. The router uses NAT for IPv4 network traffic and directly routes IPv6 network traffic.

Note: IPv6 connectivity with self-service networks often requires addition of static routes to nodes and physical network infrastructure.

1. Source the administrative project credentials.
2. Update the provider network to support external connectivity for self-service networks.

```
$ openstack network set --external provider1
```

Note: This command provides no output.

3. Source a regular (non-administrative) project credentials.
4. Create a self-service network.

```
$ openstack network create selfservice1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| mtu | 1450 |
| name | selfservice1 |
| port_security_enabled | True |
| router:external | Internal |
| shared | False |
| status | ACTIVE |
+-----+-----+
```

5. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
--network selfservice1 --dns-nameserver 8.8.4.4 selfservice1-v4
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 192.0.2.2-192.0.2.254 |
| cidr | 192.0.2.0/24 |
| dns_nameservers | 8.8.4.4 |
| enable_dhcp | True |
| gateway_ip | 192.0.2.1 |
| ip_version | 4 |
| name | selfservice1-v4 |
+-----+-----+
```

6. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-version 6 \
--ipv6-ra-mode slaac --ipv6-address-mode slaac --network selfservice1 \
--dns-nameserver 2001:4860:4860::8844 selfservice1-v6
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | fd00:192:0:2::2-fd00:192:0:2:ffff:ffff:ffff:ffff |
| cidr | fd00:192:0:2::/64 |
| dns_nameservers | 2001:4860:4860::8844 |
| enable_dhcp | True |
| gateway_ip | fd00:192:0:2::1 |
| ip_version | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode | slaac |
| name | selfservice1-v6 |
+-----+-----+
```

7. Create a router.

```
$ openstack router create router1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| name | router1 |
| status | ACTIVE |
+-----+-----+
```

8. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router1 selfservice1-v4
$ openstack router add subnet router1 selfservice1-v6
```

Note: These commands provide no output.

9. Add the provider network as the gateway on the router.

```
$ neutron router-gateway-set router1 provider1
Set gateway for router router1
```

Verify network operation

1. On each compute node, verify creation of a second qdhcp namespace.

```
# ip netns  
qdhcp-8b868082-e312-4110-8627-298109d4401c  
qdhcp-8fbc13ca-cfe0-4b8a-993b-e33f37ba66d1
```

2. On the network node, verify creation of the qrouter namespace.

```
# ip netns  
qrouter-17db2a15-e024-46d0-9250-4cd4d336a2cc
```

3. Source a regular (non-administrative) project credentials.
4. Create the appropriate security group rules to allow ping and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| direction | ingress |  
| ethertype | IPv4 |  
| protocol | icmp |  
| remote_ip_prefix | 0.0.0.0/0 |  
+-----+-----+  
  
$ openstack security group rule create --ethertype IPv6 --proto ipv6-icmp default  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| direction | ingress |  
| ethertype | IPv6 |  
| protocol | ipv6-icmp |  
+-----+-----+  
  
$ openstack security group rule create --proto tcp --dst-port 22 default  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| direction | ingress |  
| ethertype | IPv4 |  
| port_range_max | 22 |  
| port_range_min | 22 |  
| protocol | tcp |  
| remote_ip_prefix | 0.0.0.0/0 |  
+-----+-----+  
  
$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22  
default  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| direction | ingress |  
| ethertype | IPv6 |  
| port_range_max | 22 |  
| port_range_min | 22 |
```

protocol	tcp	
+-----+-----+		

5. Launch an instance with an interface on the self-service network. For example, a Cirros image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-id=NETWORK_ID
→selfservice-instance1
```

Replace NETWORK_ID with the ID of the self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID | Name | Status | Networks |
```

ID Name Status Networks			
+-----+-----+-----+			
c055cdb0-ebb4-4d65-957c-35cbdbd59306 selfservice-instance1 ACTIVE			
selfservice1=192.0.2.4, fd00:192:0:2:f816:3eff:fe30:9cb0			
+-----+-----+-----+			

Warning: The IPv4 address resides in a private IP address range (RFC1918). Thus, the Networking service performs source network address translation (SNAT) for the instance to access external networks such as the Internet. Access from external networks such as the Internet to the instance requires a floating IPv4 address. The Networking service performs destination network address translation (DNAT) from the floating IPv4 address to the instance IPv4 address on the self-service network. On the other hand, the Networking service architecture for IPv6 lacks support for NAT due to the significantly larger address space and complexity of NAT. Thus, floating IP addresses do not exist for IPv6 and the Networking service only performs routing for IPv6 subnets on self-service networks. In other words, you cannot rely on NAT to “hide” instances with IPv4 and IPv6 addresses or only IPv6 addresses and must properly implement security groups to restrict access.

7. On the controller node or any host with access to the provider network, ping the IPv6 address of the instance.

```
$ ping6 -c 4 fd00:192:0:2:f816:3eff:fe30:9cb0
PING fd00:192:0:2:f816:3eff:fe30:9cb0(fd00:192:0:2:f816:3eff:fe30:9cb0) 56 data bytes
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=1 ttl=63 time=2.08 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=2 ttl=63 time=1.88 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=3 ttl=63 time=1.55 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=4 ttl=63 time=1.62 ms

--- fd00:192:0:2:f816:3eff:fe30:9cb0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.557/1.788/2.085/0.217 ms
```

8. Optionally, enable IPv4 access from external networks such as the Internet to the instance.

- (a) Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-----+-----+
| Field      | Value
+-----+-----+
| fixed_ip   | None
| id         | 22a1b088-5c9b-43b4-97f3-970ce5df77f2
| instance_id| None
| ip          | 203.0.113.16
| pool        | provider1
+-----+-----+
```

- (b) Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance1 203.0.113.16
```

Note: This command provides no output.

- (c) On the controller node or any host with access to the provider network, ping the floating IPv4 address of the instance.

```
$ ping -c 4 203.0.113.16
PING 203.0.113.16 (203.0.113.16) 56(84) bytes of data.
64 bytes from 203.0.113.16: icmp_seq=1 ttl=63 time=3.41 ms
64 bytes from 203.0.113.16: icmp_seq=2 ttl=63 time=1.67 ms
64 bytes from 203.0.113.16: icmp_seq=3 ttl=63 time=1.47 ms
64 bytes from 203.0.113.16: icmp_seq=4 ttl=63 time=1.59 ms

--- 203.0.113.16 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.473/2.040/3.414/0.798 ms
```

9. Obtain access to the instance.
10. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
 - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
 - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
 - VXLAN ID (VNI) 102

- Self-service router
 - Gateway on the provider network
 - Interface on self-service network 1
 - Interface on self-service network 2
- Instance 1
- Instance 2

North-south scenario 1: Instance with a fixed IP address

For instances with a fixed IPv4 address, the network node performs SNAT on north-south traffic passing from self-service to external networks such as the Internet. For instances with a fixed IPv6 address, the network node performs conventional routing of traffic between self-service and external networks.

- The instance resides on compute node 1 and uses self-service network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1:

1. The instance interface (1) forwards the packet to the self-service bridge instance port (2) via veth pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to the network node via the overlay network (6).

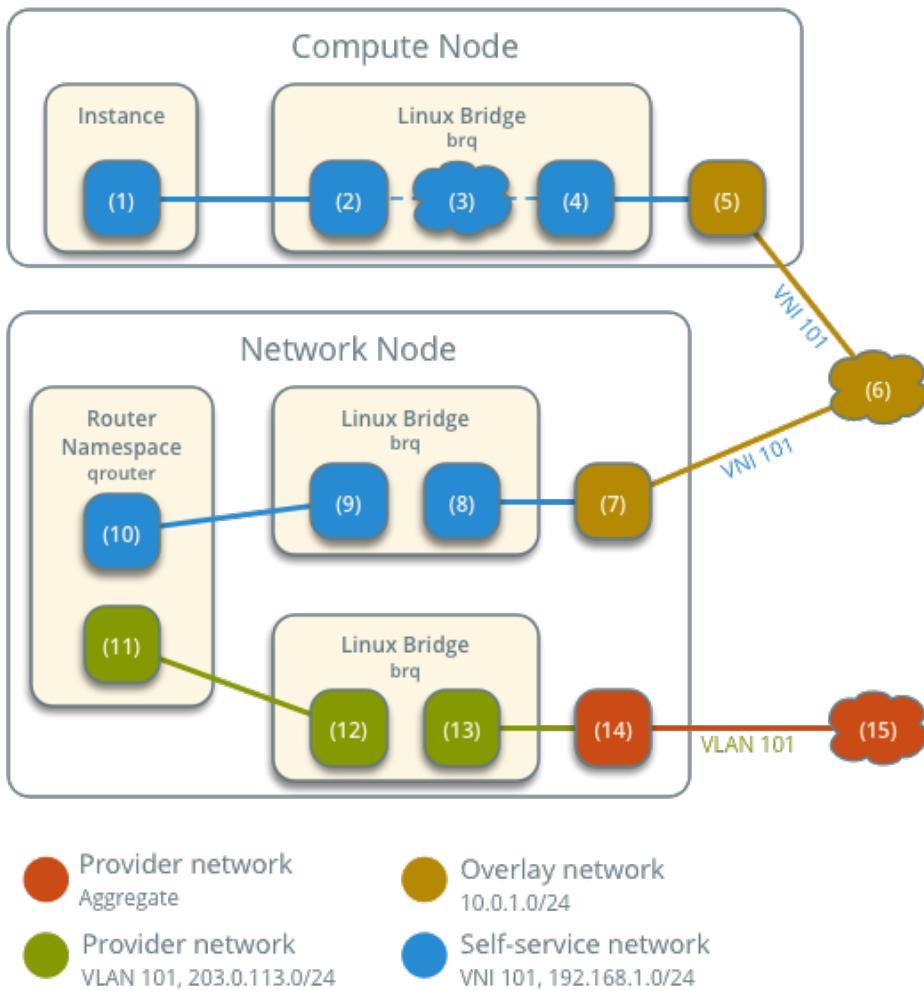
The following steps involve the network node:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. The self-service bridge router port (9) forwards the packet to the self-service network interface (10) in the router namespace.
 - For IPv4, the router performs SNAT on the packet which changes the source IP address to the router IP address on the provider network and sends it to the gateway IP address on the provider network via the gateway interface on the provider network (11).
 - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the provider network, via the provider gateway interface (11).
3. The router forwards the packet to the provider bridge router port (12).
4. The VLAN sub-interface port (13) on the provider bridge forwards the packet to the provider physical network interface (14).
5. The provider physical network interface (14) adds VLAN tag 101 to the packet and forwards it to the Internet via physical network infrastructure (15).

Note: Return traffic follows similar steps in reverse. However, without a floating IPv4 address, hosts on the provider or external networks cannot originate connections to instances on the self-service network.

Linux Bridge - Self-service Networks

Network Traffic Flow - North/South Scenario 1



North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address, the network node performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario.

- The instance resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

The following steps involve the network node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface

- (2).
2. The provider physical network interface removes VLAN tag 101 and forwards the packet to the VLAN sub-interface on the provider bridge.
 3. The provider bridge forwards the packet to the self-service router gateway port on the provider network (5).
 - For IPv4, the router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network and sends it to the gateway IP address on the self-service network via the self-service interface (6).
 - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the self-service network, via the self-service interface (6).
 4. The router forwards the packet to the self-service bridge router port (7).
 5. The self-service bridge forwards the packet to the VXLAN interface (8) which wraps the packet using VNI 101.
 6. The underlying physical interface (9) for the VXLAN interface forwards the packet to the network node via the overlay network (10).

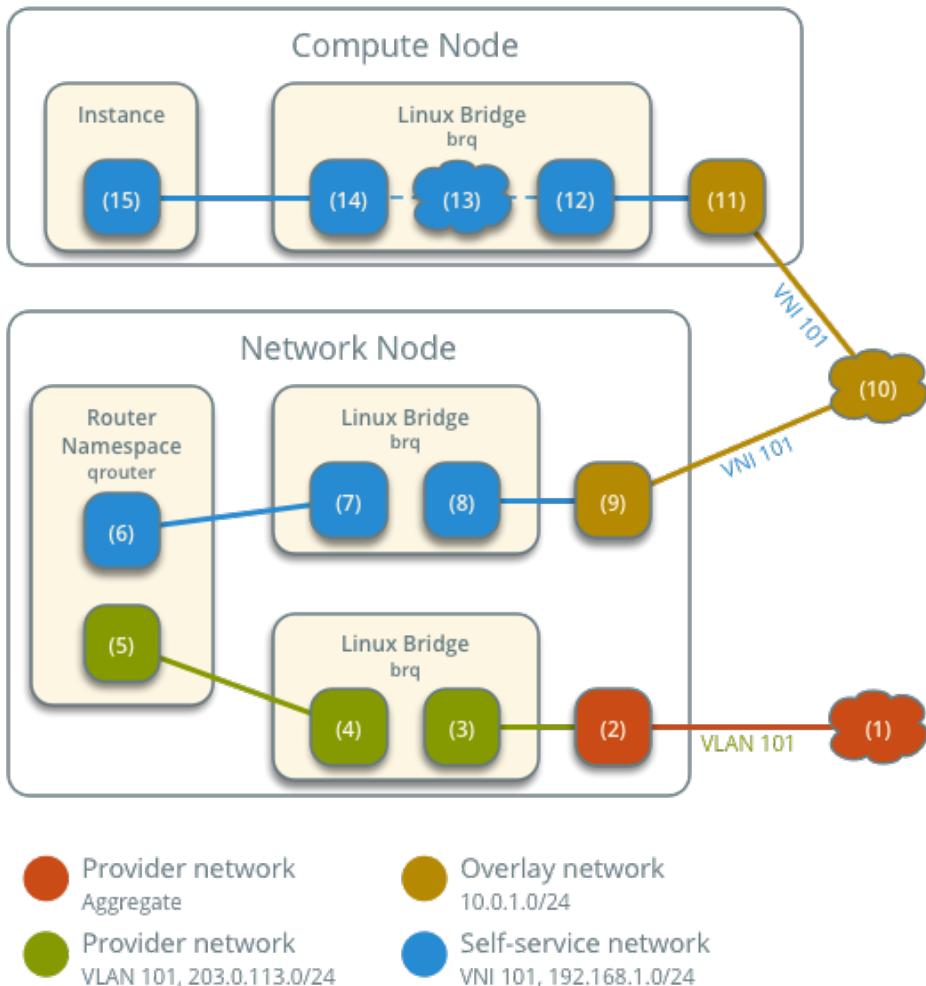
The following steps involve the compute node:

1. The underlying physical interface (11) for the VXLAN interface forwards the packet to the VXLAN interface (12) which unwraps the packet.
2. Security group rules (13) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (14) forwards the packet to the instance interface (15) via veth pair.

Note: Egress instance traffic flows similar to north-south scenario 1, except SNAT changes the source IP address of the packet to the floating IPv4 address rather than the router IP address on the provider network.

Linux Bridge - Self-service Networks

Network Traffic Flow - North/South Scenario 2



East-west scenario 1: Instances on the same network

Instances with a fixed IPv4/IPv6 or floating IPv4 address on the same network communicate directly between compute nodes containing those instances.

By default, the VXLAN protocol lacks knowledge of target location and uses multicast to discover it. After discovery, it stores the location in the local forwarding database. In large deployments, the discovery process can generate a significant amount of network traffic that all nodes must process. To eliminate the latter and generally increase efficiency, the Networking service includes the layer-2 population mechanism driver that automatically populates the forwarding database for VXLAN interfaces. The example configuration enables this driver. For more information, see [ML2 plug-in](#).

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 2 and uses self-service network 1.
- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the self-service bridge instance port (2) via veth pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to compute node 2 via the overlay network (6).

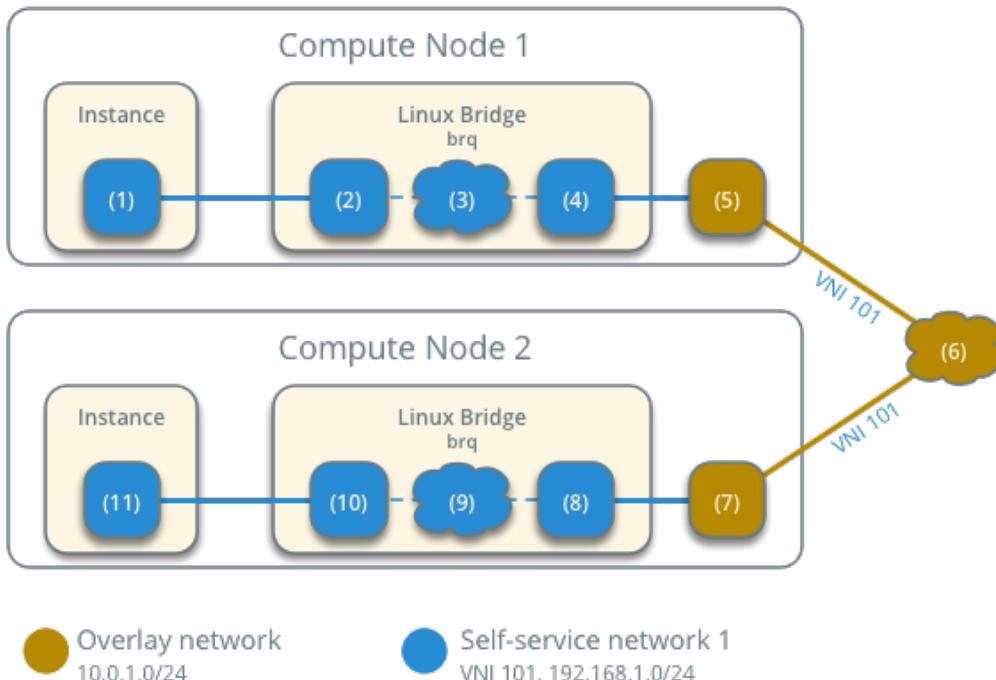
The following steps involve compute node 2:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. Security group rules (9) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (10) forwards the packet to the instance 1 interface (11) via veth pair.

Note: Return traffic follows similar steps in reverse.

Linux Bridge - Self-service Networks

Network Traffic Flow - East/West Scenario 1



East-west scenario 2: Instances on different networks

Instances using a fixed IPv4/IPv6 address or floating IPv4 address communicate via router on the network node. The self-service networks must reside on the same router.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 1 and uses self-service network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VXLAN enables multiple overlays to use the same layer-3 network.

The following steps involve the compute node:

1. The instance 1 interface (1) forwards the packet to the self-service bridge instance port (2) via veth pair.
2. Security group rules (3) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge forwards the packet to the VXLAN interface (4) which wraps the packet using VNI 101.
4. The underlying physical interface (5) for the VXLAN interface forwards the packet to the network node via the overlay network (6).

The following steps involve the network node:

1. The underlying physical interface (7) for the VXLAN interface forwards the packet to the VXLAN interface (8) which unwraps the packet.
2. The self-service bridge router port (9) forwards the packet to the self-service network 1 interface (10) in the router namespace.
3. The router sends the packet to the next-hop IP address, typically the gateway IP address on self-service network 2, via the self-service network 2 interface (11).
4. The router forwards the packet to the self-service network 2 bridge router port (12).
5. The self-service network 2 bridge forwards the packet to the VXLAN interface (13) which wraps the packet using VNI 102.
6. The physical network interface (14) for the VXLAN interface sends the packet to the compute node via the overlay network (15).

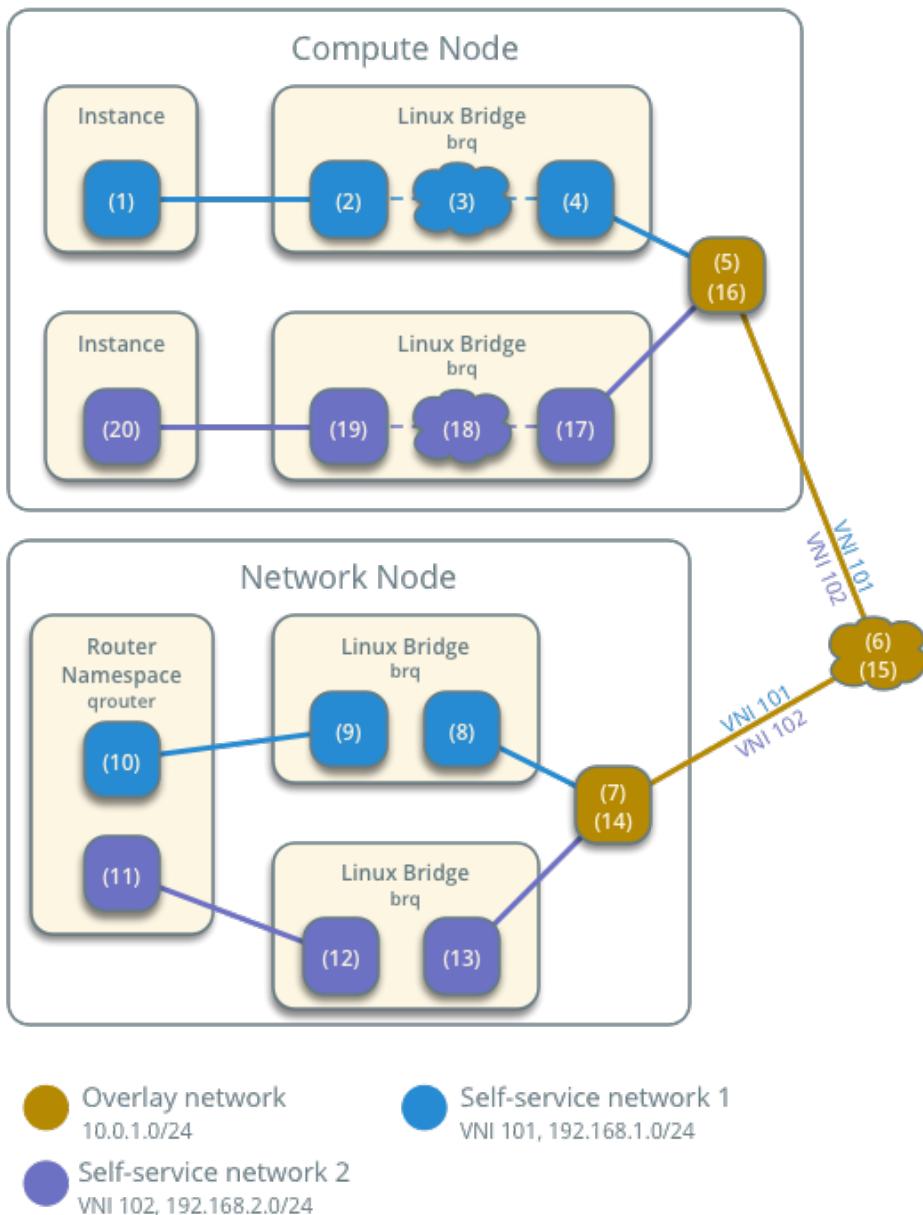
The following steps involve the compute node:

1. The underlying physical interface (16) for the VXLAN interface sends the packet to the VXLAN interface (17) which unwraps the packet.
2. Security group rules (18) on the self-service bridge handle firewalling and connection tracking for the packet.
3. The self-service bridge instance port (19) forwards the packet to the instance 2 interface (20) via veth pair.

Note: Return traffic follows similar steps in reverse.

Linux Bridge - Self-service Networks

Network Traffic Flow - East/West Scenario 2



Linux bridge: High availability using VRRP

This architecture example augments the self-service deployment example with a high-availability mechanism using the Virtual Router Redundancy Protocol (VRRP) via keepalived and provides failover of routing for self-service networks. It requires a minimum of two network nodes because VRRP creates one master (active) instance and at least one backup instance of each router.

During normal operation, `keepalived` on the master router periodically transmits *heartbeat* packets over a hidden network that connects all VRRP routers for a particular project. Each project with VRRP routers uses a separate hidden network. By default this network uses the first value in the `tenant_network_types` option in the `m12_conf.ini` file. For additional control, you can specify the self-service network type and physical network name for the hidden network using the `13_ha_network_type` and `13_ha_network_name` options in the `neutron.conf` file.

If `keepalived` on the backup router stops receiving *heartbeat* packets, it assumes failure of the master router and promotes the backup router to master router by configuring IP addresses on the interfaces in the `qrouting` namespace. In environments with more than one backup router, `keepalived` on the backup router with the next highest priority promotes that backup router to master router.

Note: This high-availability mechanism configures VRRP using the same priority for all routers. Therefore, VRRP promotes the backup router with the highest IP address to the master router.

Warning: There is a known bug with `keepalived` v1.2.15 and earlier which can cause packet loss when `max_13_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to `keepalived` v1.2.16 or greater when using this feature.

Interruption of VRRP *heartbeat* traffic between network nodes, typically due to a network interface or physical network infrastructure failure, triggers a failover. Restarting the layer-3 agent, or failure of it, does not trigger a failover providing `keepalived` continues to operate.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Instance network traffic on self-service networks using a particular router only traverses the master instance of that router. Thus, resource limitations of a particular network node can impact all master instances of routers on that network node without triggering failover to another network node. However, you can configure the scheduler to distribute the master instance of each router uniformly across a pool of network nodes to reduce the chance of resource contention on any particular network node.
- Only supports self-service networks using a router. Provider networks operate at layer-2 and rely on physical network infrastructure for redundancy.
- For instances with a floating IPv4 address, maintains state of network connections during failover as a side effect of 1:1 static NAT. The mechanism does not actually implement connection tracking.

For production deployments, we recommend at least three network nodes with sufficient resources to handle network traffic for the entire environment if one network node fails. Also, the remaining two nodes can continue to provide redundancy.

Warning: This high-availability mechanism is not compatible with the layer-2 population mechanism. You must disable layer-2 population in the `linuxbridge_agent.ini` file and restart the Linux bridge agent on all existing network and compute nodes prior to deploying the example configuration.

Prerequisites

Add one network node with the following components:

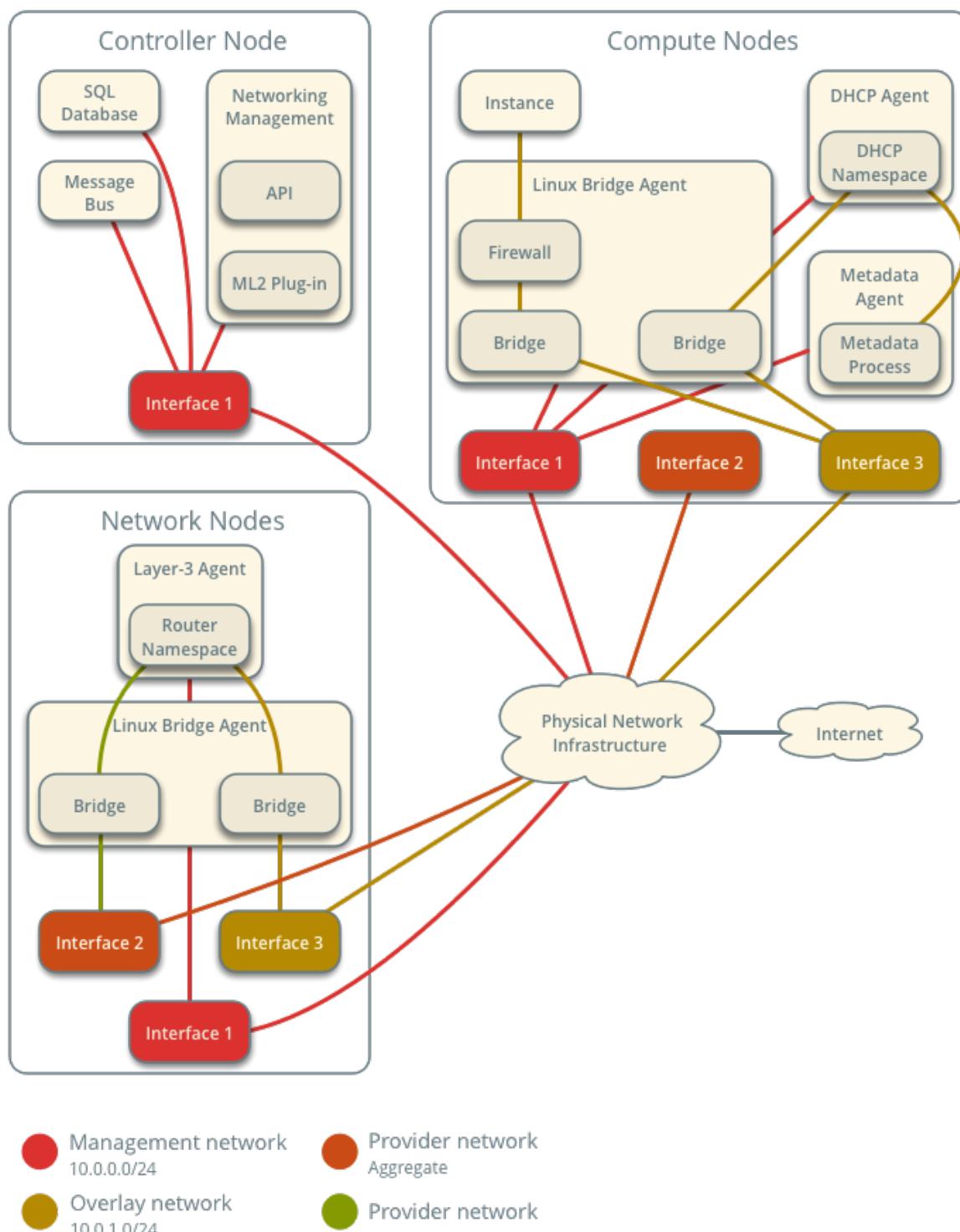
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking layer-2 agent, layer-3 agent, and any dependencies.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network nodes.

Architecture

Linux Bridge - High-availability with VRRP

Overview

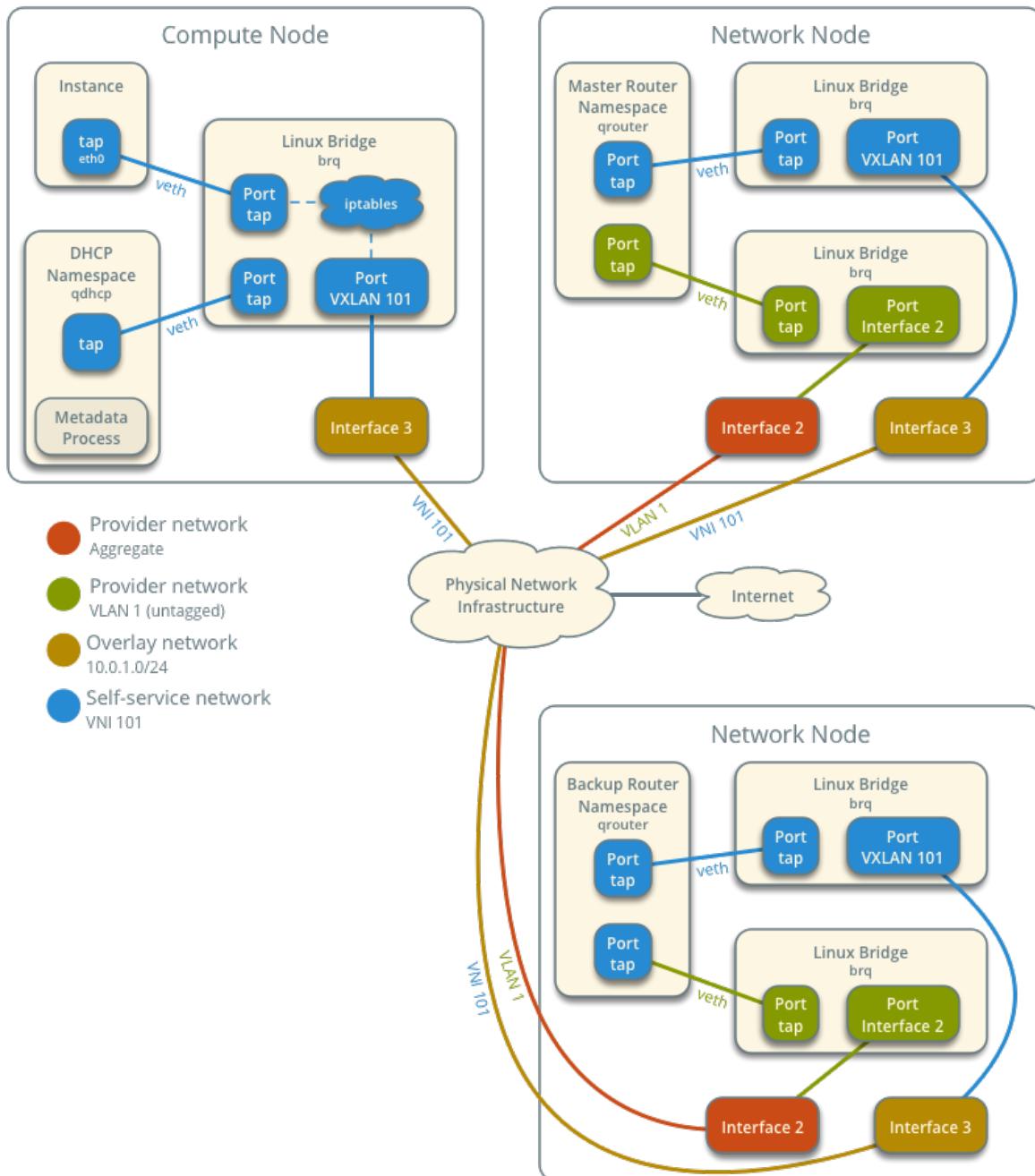


The following figure shows components and connectivity for one self-service network and one untagged (flat) network. The master router resides on network node 1. In this particular case, the instance resides on the same

compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the overlay physical network interface.

Linux Bridge - High-availability with VRRP

Components and Connectivity



Example configuration

Use the following example configuration as a template to add support for high-availability using VRRP to an existing operational environment that supports self-service networks.

Controller node

1. In the neutron.conf file:

- Enable VRRP.

```
[DEFAULT]
l3_ha = True
```

2. Restart the following services:

- Server

Network node 1

No changes.

Network node 2

1. Install the Networking service Linux bridge layer-2 agent and layer-3 agent.

2. In the neutron.conf file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_auth_token], [nova], and [agent] sections.

3. In the linuxbridge_agent.ini file, configure the layer-2 agent.

```
[linux_bridge]
physical_interface_mappings = provider:PROVIDER_INTERFACE

[vxlan]
enable_vxlan = True
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[securitygroup]
firewall_driver = iptables
```

Replace PROVIDER_INTERFACE with the name of the underlying interface that handles provider networks. For example, eth1.

Replace OVERLAY_INTERFACE_IP_ADDRESS with the IP address of the interface that handles VXLAN overlays for self-service networks.

4. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = linuxbridge
external_network_bridge =
```

Note: The `external_network_bridge` option intentionally contains no value.

5. Start the following services:

- Linux bridge agent
- Layer-3 agent

Compute nodes

No changes.

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID           | Agent Type      | Host     | Availability |
| Zone | Alive | State | Binary
+-----+-----+-----+-----+
| 09de6af6-c5f1-4548-8b09-18801f068c57 | Linux bridge agent | compute2 |   |
|   | True  | UP    | neutron-linuxbridge-agent |
| 188945d1-9e70-4803-a276-df924e0788a4 | Linux bridge agent | compute1 |   |
|   | True  | UP    | neutron-linuxbridge-agent |
| e76c440d-d5f6-4316-a674-d689630b629e | DHCP agent        | compute1 | nova |
|   | True  | UP    | neutron-dhcp-agent   |
| e67367de-6657-11e6-86a4-931cd04404bb | DHCP agent        | compute2 | nova |
|   | True  | UP    | neutron-dhcp-agent   |
| e8174cae-6657-11e6-89f0-534ac6d0cb5c | Metadata agent    | compute1 |   |
|   | True  | UP    | neutron-metadata-agent |
| ece49ec6-6657-11e6-bafb-c7560f19197d | Metadata agent    | compute2 |   |
|   | True  | UP    | neutron-metadata-agent |
| 598f6357-4331-4da5-a420-0f5be000bec9 | L3 agent          | network1 | nova |
|   | True  | UP    | neutron-l3-agent    |
| f4734e0f-bcd5-4922-a19d-e31d56b0a7ae | Linux bridge agent | network1 |   |
|   | True  | UP    | neutron-linuxbridge-agent |
| 670e5805-340b-4182-9825-fa8319c99f23 | Linux bridge agent | network2 |   |
|   | True  | UP    | neutron-linuxbridge-agent |
```

96224e89-7c15-42e9-89c4-8caac7abdd54 L3 agent	network2 nova	
↳ True UP neutron-l3-agent		
+-----+-----+-----+-----+	+-----+-----+-----+	

Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use VRRP. The following procedure creates an additional self-service network and router. The Networking service also supports adding high-availability to existing routers. However, the procedure requires administratively disabling and enabling each router which temporarily interrupts network connectivity for self-service networks with interfaces on that router.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

\$ openstack network create selfservice2		
Field	Value	
+-----+-----+-----+		
admin_state_up	UP	
mtu	1450	
name	selfservice2	
port_security_enabled	True	
router:external	Internal	
shared	False	
status	ACTIVE	
+-----+-----+-----+		

3. Create a IPv4 subnet on the self-service network.

\$ openstack subnet create --subnet-range 198.51.100.0/24 \\\n--network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4		
Field	Value	
+-----+-----+-----+		
allocation_pools	198.51.100.2-198.51.100.254	
cidr	198.51.100.0/24	
dns_nameservers	8.8.4.4	
enable_dhcp	True	
gateway_ip	198.51.100.1	
ip_version	4	
name	selfservice2-v4	
+-----+-----+-----+		

4. Create a IPv6 subnet on the self-service network.

\$ openstack subnet create --subnet-range fd00:198:51:100::/64 --ip-version 6 \\\n--ipv6-ra-mode slaac --ipv6-address-mode slaac --network selfservice2 \\\n--dns-nameserver 2001:4860:4860::8844 selfservice2-v6		
Field	Value	
+-----+-----+-----+		
allocation_pools	fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff	

cidr	fd00:198:51:100::/64
dns_nameservers	2001:4860:4860::8844
enable_dhcp	True
gateway_ip	fd00:198:51:100::1
ip_version	6
ipv6_address_mode	slaac
ipv6_ra_mode	slaac
name	selfservice2-v6
+	-----+-----+

5. Create a router.

\$ openstack router create router2
-----+-----+
Field Value
-----+-----+
admin_state_up UP
name router2
status ACTIVE
-----+-----+

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4  
$ openstack router add subnet router2 selfservice2-v6
```

Note: These commands provide no output.

7. Add the provider network as a gateway on the router.

```
$ neutron router-gateway-set router2 provider1  
Set gateway for router router2
```

Verify network operation

1. Source the administrative project credentials.
2. Verify creation of the internal high-availability network that handles VRRP *heartbeat* traffic.

\$ openstack network list
-----+-----+
ID Name
Subnets
-----+-----+
1b8519c1-59c4-415c-9da2-a67d53c68455 HA network tenant
f986edf55ae945e2bef3cb4bfd589928 6843314a-1e76-4cc9-94f5-c64b7a39364a
-----+-----+

3. On each network node, verify creation of a qrouter namespace with the same ID.

Network node 1:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Network node 2:

```
# ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Note: The namespace for router 1 from *Linux bridge: Self-service networks* should only appear on network node 1 because of creation prior to enabling VRRP.

4. On each network node, show the IP address of interfaces in the qrouting namespace. With the exception of the VRRP interface, only one namespace belonging to the master router instance contains IP addresses on the interfaces.

Network node 1:

```
# ip netns exec qrouting-b6206312-878e-497c-8ef7-eb384f8add96 ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    ↳qlen 1
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ha-eb820380-40@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
    ↳state UP group default qlen 1000
        link/ether fa:16:3e:78:ba:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 169.254.192.1/18 brd 169.254.255.255 scope global ha-eb820380-40
            valid_lft forever preferred_lft forever
        inet 169.254.0.1/24 scope global ha-eb820380-40
            valid_lft forever preferred_lft forever
        inet6 fe80::f816:3eff:fe78:ba99/64 scope link
            valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
    ↳state UP group default qlen 1000
        link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 198.51.100.1/24 scope global qr-da3504ad-ba
            valid_lft forever preferred_lft forever
        inet6 fe80::f816:3eff:fedc:8ea8/64 scope link
            valid_lft forever preferred_lft forever
4: qr-442e36eb-fc@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
    ↳state UP group default qlen 1000
        link/ether fa:16:3e:ee:c8:41 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet6 fd00:198:51:100::1/64 scope global nodad
            valid_lft forever preferred_lft forever
        inet6 fe80::f816:3eff:feee:c841/64 scope link
            valid_lft forever preferred_lft forever
5: qg-33fedbc5-43@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    ↳state UP group default qlen 1000
        link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 203.0.113.21/24 scope global qg-33fedbc5-43
            valid_lft forever preferred_lft forever
        inet6 fd00:203:0:113::21/64 scope global nodad
            valid_lft forever preferred_lft forever
```

```
inet6 fe80::f816:3eff:fe03:1af6/64 scope link
  valid_lft forever preferred_lft forever
```

Network node 2:

```
# ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
  ↳qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: ha-7a7ce184-36@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP
  ↳UP group default qlen 1000
    link/ether fa:16:3e:16:59:84 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 169.254.192.2/18 brd 169.254.255.255 scope global ha-7a7ce184-36
      valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe16:5984/64 scope link
      valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
  ↳state UP group default qlen 1000
    link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
4: qr-442e36eb-fc@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
  ↳state UP group default qlen 1000
5: qg-33fedbc5-43@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
  ↳state UP group default qlen 1000
    link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Note: The master router may reside on network node 2.

5. Launch an instance with an interface on the additional self-service network. For example, a Cirros image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-id=NETWORK_ID
  ↳selfservice-instance2
```

Replace NETWORK_ID with the ID of the additional self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID          | Name           | Status | Networks   |
|-----+-----+-----+
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 | ACTIVE |           |
|   ↳selfservice2=fd00:198:51:100:f816:3eff:fe71:e93e, 198.51.100.4 |           |
|-----+-----+-----+
```

7. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-----+
| Field      | Value
+-----+
| fixed_ip   | None
| id         | 0174056a-fa56-4403-b1ea-b5151a31191f
| instance_id| None
| ip          | 203.0.113.17
| pool        | provider1
+-----+
```

- Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

Note: This command provides no output.

Verify failover operation

- Begin a continuous ping of both the floating IPv4 address and IPv6 address of the instance. While performing the next three steps, you should see a minimal, if any, interruption of connectivity to the instance.
- On the network node with the master router, administratively disable the overlay network interface.
- On the other network node, verify promotion of the backup router to master router by noting addition of IP addresses to the interfaces in the qrouter namespace.
- On the original network node in step 2, administratively enable the overlay network interface. Note that the master router remains on the network node in step 3.

Keepalived VRRP health check

The health of your keepalived instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the `l3_agent.ini` file:

```
ha_vrrp_health_check_interval = 30
```

Where `ha_vrrp_health_check_interval` indicates how often in seconds the health check should run. The default value is 0, which indicates that the check should not run at all.

Network traffic flow

This high-availability mechanism simply augments [Linux bridge: Self-service networks](#) with failover of layer-3 services to another router if the master router fails. Thus, you can reference [Self-service network traffic flow](#)

for normal operation.

Open vSwitch mechanism driver

The Open vSwitch (OVS) mechanism driver uses a combination of OVS and Linux bridges as interconnection devices. However, optionally enabling the OVS native implementation of security groups removes the dependency on Linux bridges.

We recommend using Open vSwitch version 2.4 or higher. Optional features may require a higher minimum version.

Open vSwitch: Provider networks

This architecture example provides layer-2 connectivity between instances and the physical network infrastructure using VLAN (802.1q) tagging. It supports one untagged (flat) network and up to 4095 tagged (VLAN) networks. The actual quantity of VLAN networks depends on the physical network infrastructure. For more information on provider networks, see [Provider networks](#).

Warning: Linux distributions often package older releases of Open vSwitch that can introduce issues during operation with the Networking service. We recommend using at least the latest long-term stable (LTS) release of Open vSwitch for the best experience and support from Open vSwitch. See <http://www.openvswitch.org> for available releases and the [installation instructions](#) for

Prerequisites

One controller node with the following components:

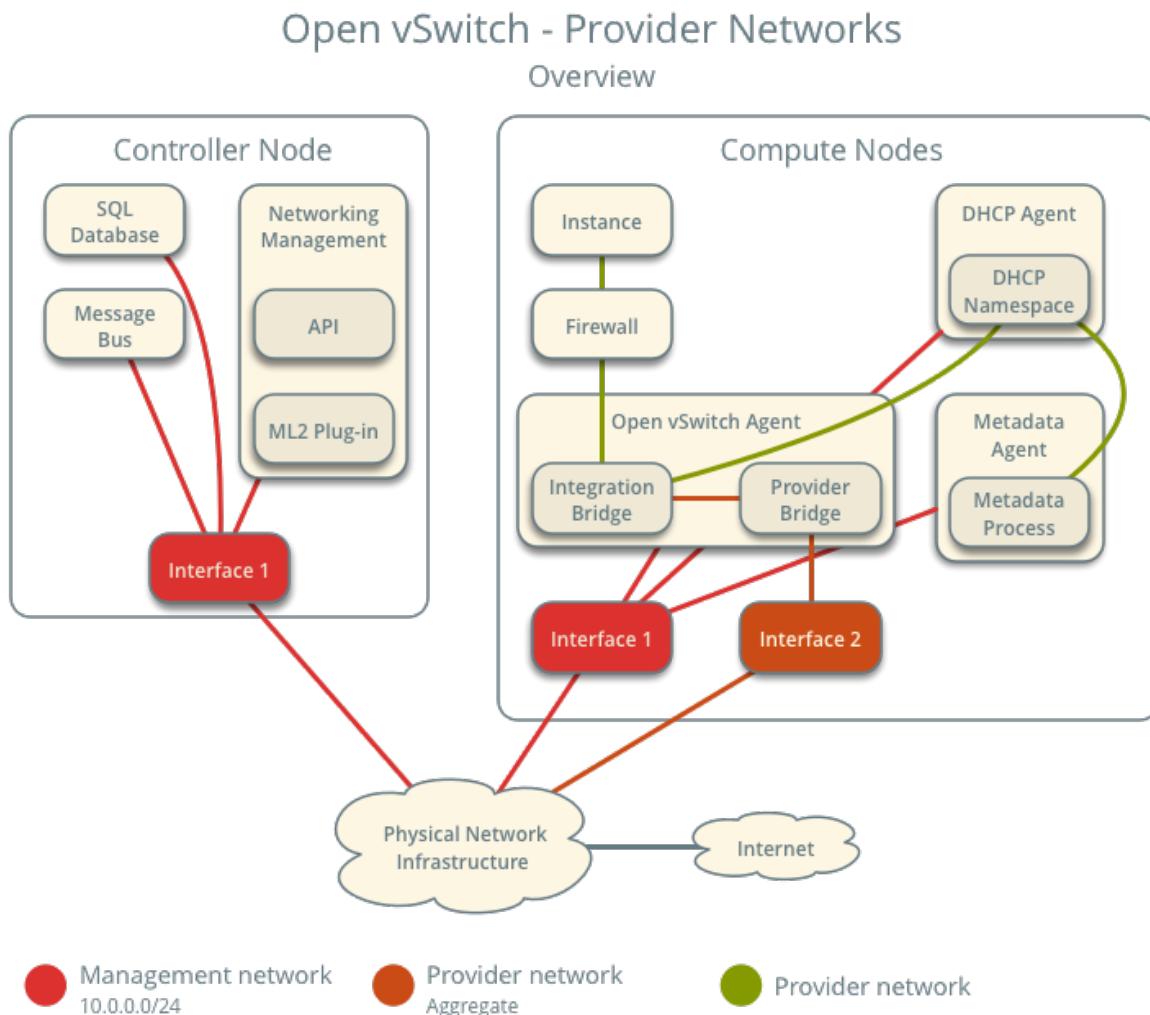
- Two network interfaces: management and provider.
- OpenStack Networking server service and ML2 plug-in.

Two compute nodes with the following components:

- Two network interfaces: management and provider.
- OpenStack Networking Open vSwitch (OVS) layer-2 agent, DHCP agent, metadata agent, and any dependencies including OVS.

Note: Larger deployments typically deploy the DHCP and metadata agents on a subset of compute nodes to increase performance and redundancy. However, too many agents can overwhelm the message bus. Also, to further simplify any deployment, you can omit the metadata agent and use a configuration drive to provide metadata to instances.

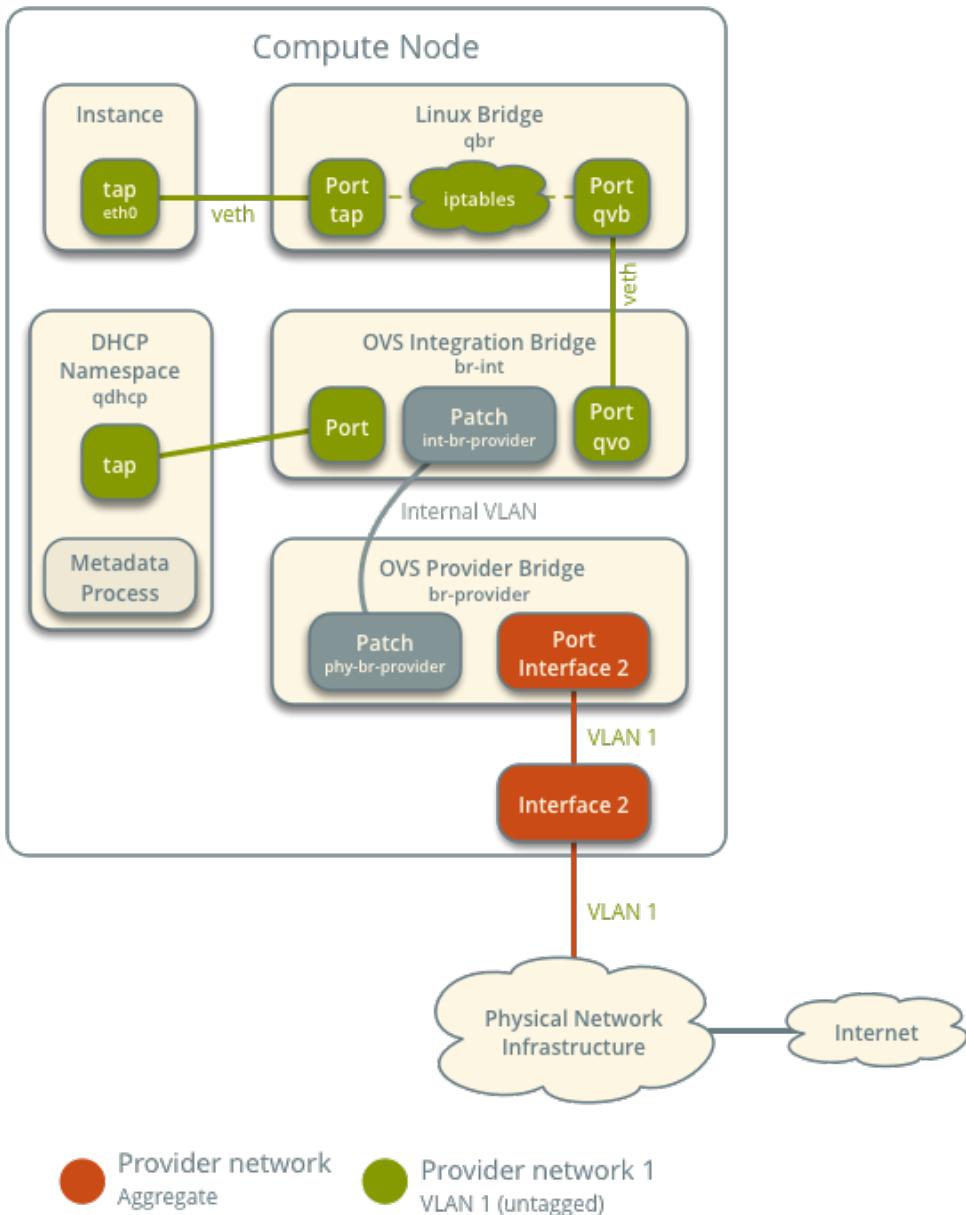
Architecture



The following figure shows components and connectivity for one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace with a port on the OVS integration bridge.

Open vSwitch - Provider Networks

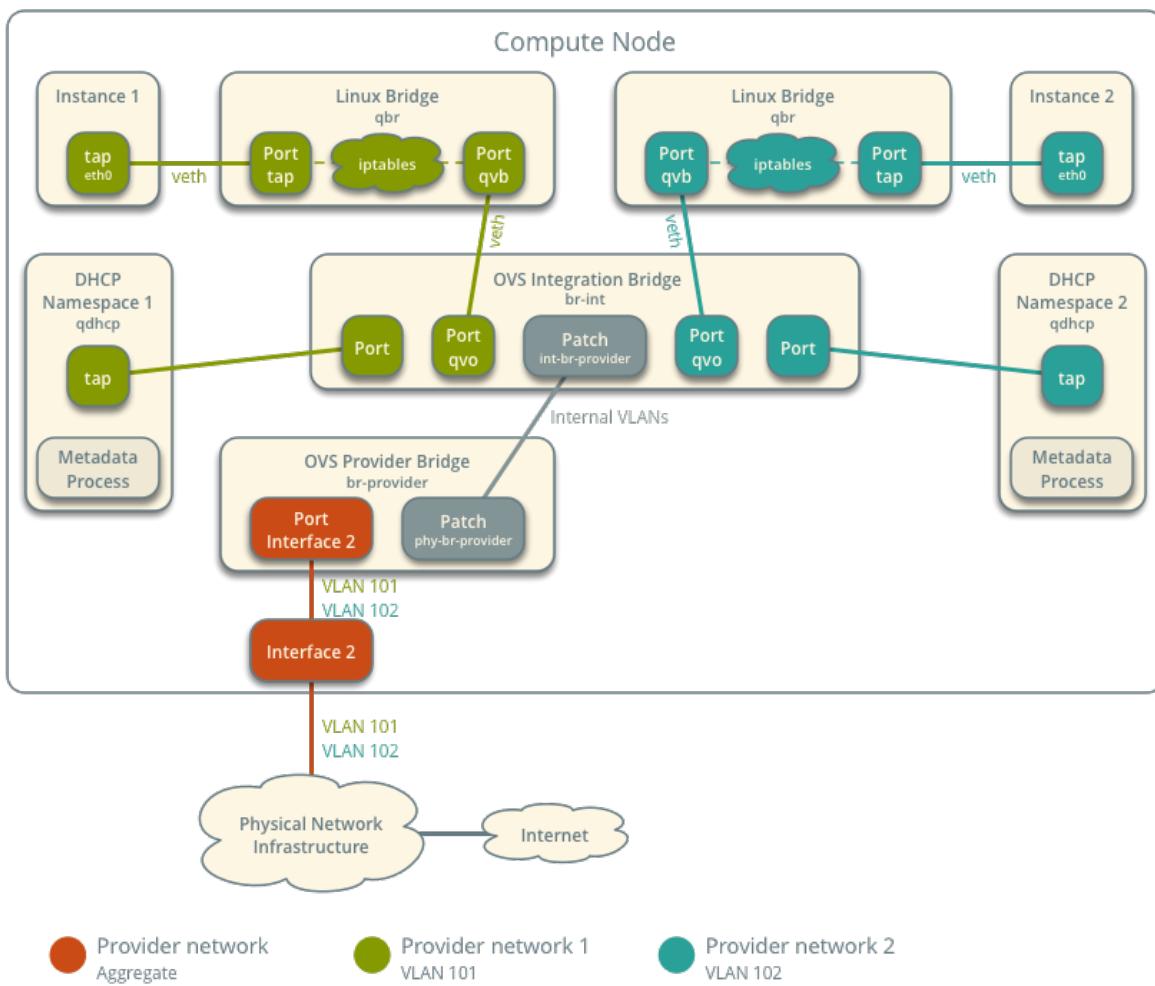
Components and Connectivity



The following figure describes virtual connectivity among components for two tagged (VLAN) networks. Essentially, all networks use a single OVS integration bridge with different internal VLAN tags. The internal VLAN tags almost always differ from the network VLAN assignment in the Networking service. Similar to the untagged network case, the DHCP agent may reside on a different compute node.

Open vSwitch - Provider Networks

Components and Connectivity



Note: These figures omit the controller node because it does not handle instance network traffic.

Example configuration

Use the following example configuration as a template to deploy provider networks in your environment.

Controller node

1. Install the Networking service components that provide the `neutron-server` service and ML2 plug-in.
2. In the `neutron.conf` file:
 - Configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone
```

```
[database]
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_auth_token], [nova], and [agent] sections.

- Disable service plug-ins because provider networks do not require any. However, this breaks portions of the dashboard that manage the Networking service. See the [Ocata Install Tutorials and Guides](#) for more information.

```
[DEFAULT]
service_plugins =
```

- Enable two DHCP agents per network so both compute nodes can provide DHCP service provider networks.

```
[DEFAULT]
dhcp_agents_per_network = 2
```

- If necessary, *configure MTU*.

3. In the `ml2_conf.ini` file:

- Configure drivers and network types:

```
[ml2]
type_drivers = flat,vlan
tenant_network_types =
mechanism_drivers = openvswitch
extension_drivers = port_security
```

- Configure network mappings:

```
[ml2_type_flat]
flat_networks = provider

[ml2_type_vlan]
network_vlan_ranges = provider
```

Note: The `tenant_network_types` option contains no value because the architecture does not support self-service networks.

Note: The `provider` value in the `network_vlan_ranges` option lacks VLAN ID ranges to

support use of arbitrary VLAN IDs.

4. Populate the database.

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

5. Start the following services:

- Server

Compute nodes

1. Install the Networking service OVS layer-2 agent, DHCP agent, and metadata agent.
2. Install OVS.
3. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_authhtoken]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_authhtoken], [nova], and [agent] sections.

4. In the `openvswitch_agent.ini` file, configure the OVS agent:

```
[ovs]
bridge_mappings = provider:br-provider

[securitygroup]
firewall_driver = iptables_hybrid
```

5. In the `dhcp_agent.ini` file, configure the DHCP agent:

```
[DEFAULT]
interface_driver = openvswitch
enable_isolated_metadata = True
force_metadata = True
```

Note: The `force_metadata` option forces the DHCP agent to provide a host route to the metadata

service on 169.254.169.254 regardless of whether the subnet contains an interface on a router, thus maintaining similar and predictable metadata behavior among subnets.

6. In the `metadata_agent.ini` file, configure the metadata agent:

```
[DEFAULT]
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

The value of `METADATA_SECRET` must match the value of the same option in the `[neutron]` section of the `nova.conf` file.

7. Start the following services:

- OVS

8. Create the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-br br-provider
```

9. Add the provider network interface as a port on the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-port br-provider PROVIDER_INTERFACE
```

Replace `PROVIDER_INTERFACE` with the name of the underlying interface that handles provider networks. For example, `eth1`.

10. Start the following services:

- OVS agent
- DHCP agent
- Metadata agent

Verify service operation

1. Source the administrative project credentials.

2. Verify presence and operation of the agents:

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID               | Agent Type      | Host     | Availability |
| Zone  | Alive | State | Binary          |
+-----+-----+-----+-----+
| 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent    | compute2  |             |
|       | True   | UP    | neutron-metadata-agent |
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent | compute2  |             |
|       | True   | UP    | neutron-openvswitch-agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent       | compute2  | nova        |
|       | True   | UP    | neutron-dhcp-agent      |
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent    | compute1  |             |
|       | True   | UP    | neutron-metadata-agent  |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent       | compute1  | nova        |
|       | True   | UP    | neutron-dhcp-agent      |
```

bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1 Open vSwitch agent compute1	✖
↳ True UP neutron-openvswitch-agent	
+-----+-----+-----+-----+	

Create initial networks

The configuration supports one flat or multiple VLAN provider networks. For simplicity, the following procedure creates one flat provider network.

1. Source the administrative project credentials.
2. Create a flat network.

\$ openstack network create --share --provider-physical-network provider \	
--provider-network-type flat provider1	
+-----+-----+	
Field Value	
+-----+-----+	
admin_state_up UP	
mtu 1500	
name provider1	
port_security_enabled True	
provider:network_type flat	
provider:physical_network provider	
provider:segmentation_id None	
router:external Internal	
shared True	
status ACTIVE	
+-----+-----+	

Note: The share option allows any project to use this network. To limit access to provider networks, see [Role-Based Access Control \(RBAC\)](#).

Note: To create a VLAN network instead of a flat network, change `--provider:network_type flat` to `--provider-network-type vlan` and add `--provider-segment` with a value referencing the VLAN ID.

3. Create a IPv4 subnet on the provider network.

\$ openstack subnet create --subnet-range 203.0.113.0/24 --gateway 203.0.113.1 \	
--network provider1 --allocation-pool start=203.0.113.11,end=203.0.113.250 \	
--dns-nameserver 8.8.4.4 provider1-v4	
+-----+-----+	
Field Value	
+-----+-----+	
allocation_pools 203.0.113.11-203.0.113.250	
cidr 203.0.113.0/24	
dns_nameservers 8.8.4.4	
enable_dhcp True	
gateway_ip 203.0.113.1	

ip_version	4	
name	provider1-v4	

Note: Enabling DHCP causes the Networking service to provide DHCP which can interfere with existing DHCP services on the physical network infrastructure.

4. Create a IPv6 subnet on the provider network.

```
$ openstack subnet create --subnet-range fd00:203:0:113::/64 --gateway fd00:203:0:113::1 \
--ip-version 6 --ipv6-address-mode slaac --network provider1 \
--dns-nameserver 2001:4860:4860::8844 provider1-v6
+-----+
| Field      | Value
+-----+
| allocation_pools | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr       | fd00:203:0:113::/64
| dns_nameservers | 2001:4860:4860::8844
| enable_dhcp | True
| gateway_ip  | fd00:203:0:113::1
| ip_version   | 6
| ipv6_address_mode | slaac
| ipv6_ra_mode | None
| name        | provider1-v6
+-----+
```

Note: The Networking service uses the layer-3 agent to provide router advertisement. Provider networks rely on physical network infrastructure for layer-3 services rather than the layer-3 agent. Thus, the physical network infrastructure must provide router advertisement on provider networks for proper operation of IPv6.

Verify network operation

1. On each compute node, verify creation of the qdhcp namespace.

```
# ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
```

2. Source a regular (non-administrative) project credentials.
3. Create the appropriate security group rules to allow ping and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+-----+
| Field      | Value
+-----+
| direction   | ingress
| ethertype   | IPv4
| protocol    | icmp
| remote_ip_prefix | 0.0.0.0/0
```

```
+-----+-----+
$ openstack security group rule create --ethertype IPv6 --proto ipv6-icmp default
+-----+-----+
| Field      | Value      |
+-----+-----+
| direction  | ingress   |
| ethertype  | IPv6      |
| protocol   | ipv6-icmp |
+-----+-----+

$ openstack security group rule create --proto tcp --dst-port 22 default
+-----+-----+
| Field          | Value      |
+-----+-----+
| direction      | ingress   |
| ethertype      | IPv4      |
| port_range_max | 22        |
| port_range_min | 22        |
| protocol       | tcp        |
| remote_ip_prefix | 0.0.0.0/0 |
+-----+-----+

$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22
↪default
+-----+-----+
| Field      | Value      |
+-----+-----+
| direction  | ingress   |
| ethertype  | IPv6      |
| port_range_max | 22        |
| port_range_min | 22        |
| protocol   | tcp        |
+-----+-----+
```

4. Launch an instance with an interface on the provider network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros \
--nic net-id=NETWORK_ID provider-instance1
```

Replace NETWORK_ID with the ID of the provider network.

5. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID           | Name         | Status | Networks  |
|             | Image Name |        |           |
+-----+-----+-----+
| 018e0ae2-b43c-4271-a78d-62653dd03285 | provider-instance1 | ACTIVE | provider1=203.
| 0.113.13, fd00:203:0:113:f816:3eff:fe58:be4e | cirros     |
+-----+-----+-----+
```

6. On the controller node or any host with access to the provider network, ping the IPv4 and IPv6 addresses of the instance.

```
$ ping -c 4 203.0.113.13
PING 203.0.113.13 (203.0.113.13) 56(84) bytes of data.
64 bytes from 203.0.113.13: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.13: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.13: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.13: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.13 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms

$ ping6 -c 4 fd00:203:0:113:f816:3eff:fe58:be4e
PING fd00:203:0:113:f816:3eff:fe58:be4e(fd00:203:0:113:f816:3eff:fe58:be4e) 56 data
→bytes
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=1 ttl=64 time=1.25 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=2 ttl=64 time=0.683 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=3 ttl=64 time=0.762 ms
64 bytes from fd00:203:0:113:f816:3eff:fe58:be4e icmp_seq=4 ttl=64 time=0.486 ms

--- fd00:203:0:113:f816:3eff:fe58:be4e ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.486/0.796/1.253/0.282 ms
```

7. Obtain access to the instance.
8. Test IPv4 and IPv6 connectivity to the Internet or other external network.

Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network 1 (VLAN)
 - VLAN ID 101 (tagged)
 - IP address ranges 203.0.113.0/24 and fd00:203:0:113::/64
 - Gateway (via physical network infrastructure)
 - * IP addresses 203.0.113.1 and fd00:203:0:113:0::1
- Provider network 2 (VLAN)
 - VLAN ID 102 (tagged)
 - IP address range 192.0.2.0/24 and fd00:192:0:2::/64
 - Gateway
 - * IP addresses 192.0.2.1 and fd00:192:0:2::1
- Instance 1

- IP addresses 203.0.113.101 and fd00:203:0:113:0::101
- Instance 2
 - IP addresses 192.0.2.101 and fd00:192:0:2:0::101

North-south

- The instance resides on compute node 1 and uses provider network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1.

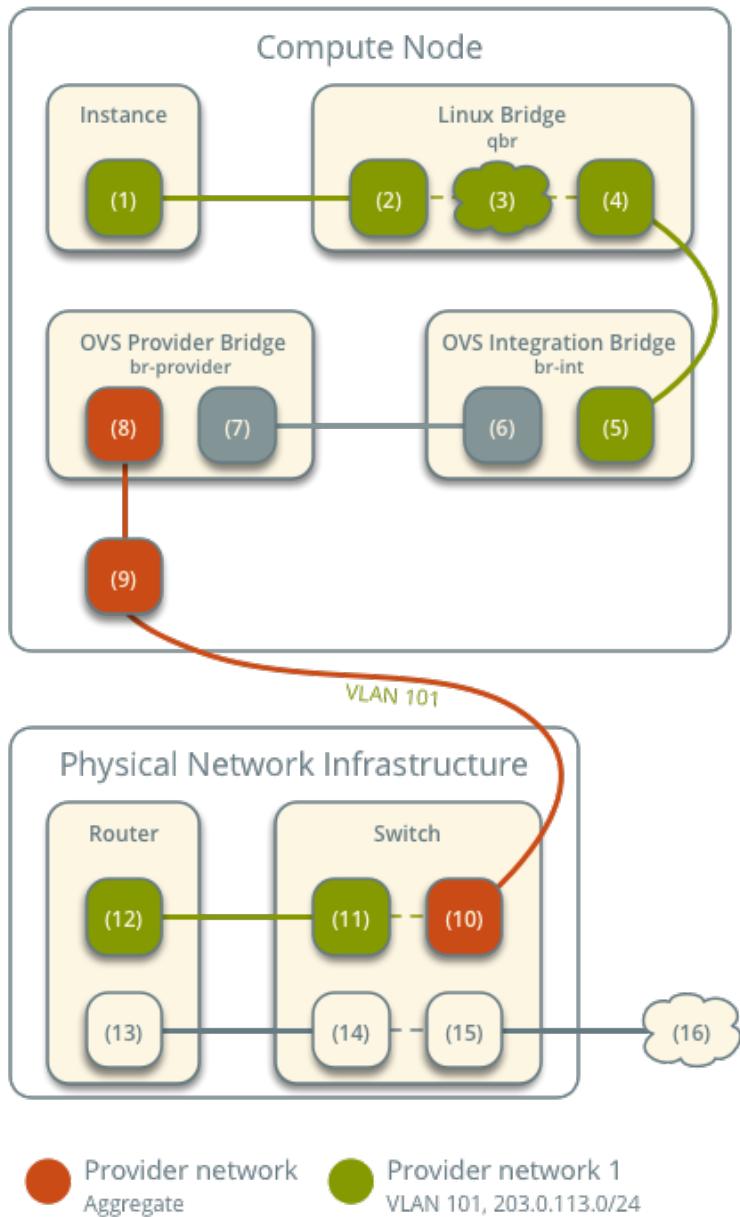
1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge int-br-provider patch port (6) forwards the packet to the OVS provider bridge phy-br-provider patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (11).
2. The router routes the packet from the provider network (12) to the external network (13) and forwards the packet to the switch (14).
3. The switch forwards the packet to the external network (15).
4. The external network (16) receives the packet.

Open vSwitch - Provider Networks

Network Traffic Flow - North/South Scenario



Note: Return traffic follows similar steps in reverse.

East-west scenario 1: Instances on the same network

Instances on the same network communicate directly between compute nodes containing those instances.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 2 and uses provider network 1.

- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge `int-br-provider` patch port (6) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

The following steps involve the physical network infrastructure:

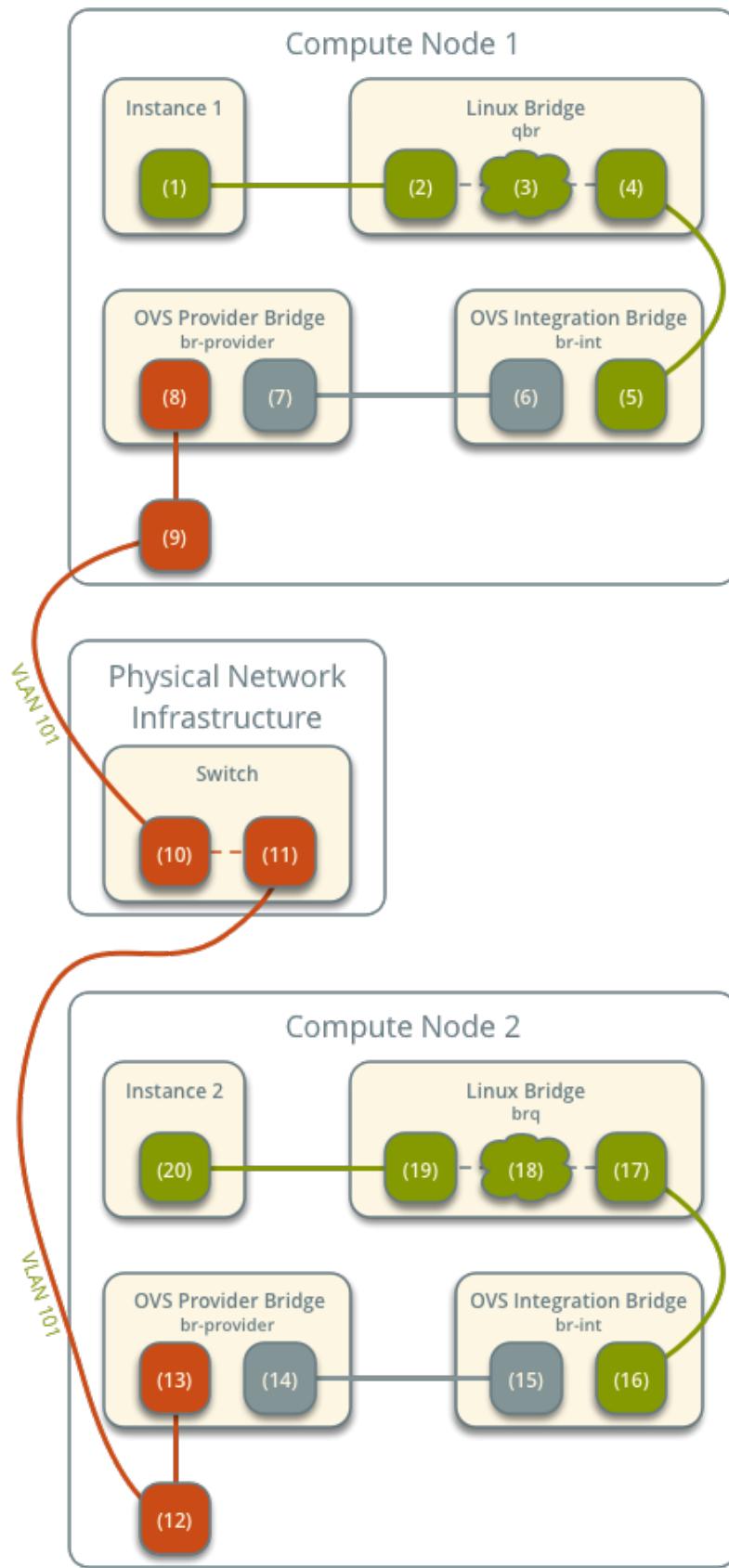
1. The switch forwards the packet from compute node 1 to compute node 2 (11).

The following steps involve compute node 2:

1. The physical network interface (12) forwards the packet to the OVS provider bridge provider network port (13).
2. The OVS provider bridge `phy-br-provider` patch port (14) forwards the packet to the OVS integration bridge `int-br-provider` patch port (15).
3. The OVS integration bridge swaps the actual VLAN tag 101 with the internal VLAN tag.
4. The OVS integration bridge security group port (16) forwards the packet to the security group bridge OVS port (17).
5. Security group rules (18) on the security group bridge handle firewalling and connection tracking for the packet.
6. The security group bridge instance port (19) forwards the packet to the instance 2 interface (20) via veth pair.

Open vSwitch - Provider Networks

Network Traffic Flow - East/West Scenario 1



Note: Return traffic follows similar steps in reverse.

East-west scenario 2: Instances on different networks

Instances communicate via router on the physical network infrastructure.

- Instance 1 resides on compute node 1 and uses provider network 1.
- Instance 2 resides on compute node 1 and uses provider network 2.
- Instance 1 sends a packet to instance 2.

Note: Both instances reside on the same compute node to illustrate how VLAN tagging enables multiple logical layer-2 networks to use the same physical layer-2 network.

The following steps involve the compute node:

1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge `int-br-provider` patch port (6) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (7).
6. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
7. The OVS provider bridge provider network port (8) forwards the packet to the physical network interface (9).
8. The physical network interface forwards the packet to the physical network infrastructure switch (10).

The following steps involve the physical network infrastructure:

1. The switch removes VLAN tag 101 from the packet and forwards it to the router (11).
2. The router routes the packet from provider network 1 (12) to provider network 2 (13).
3. The router forwards the packet to the switch (14).
4. The switch adds VLAN tag 102 to the packet and forwards it to compute node 1 (15).

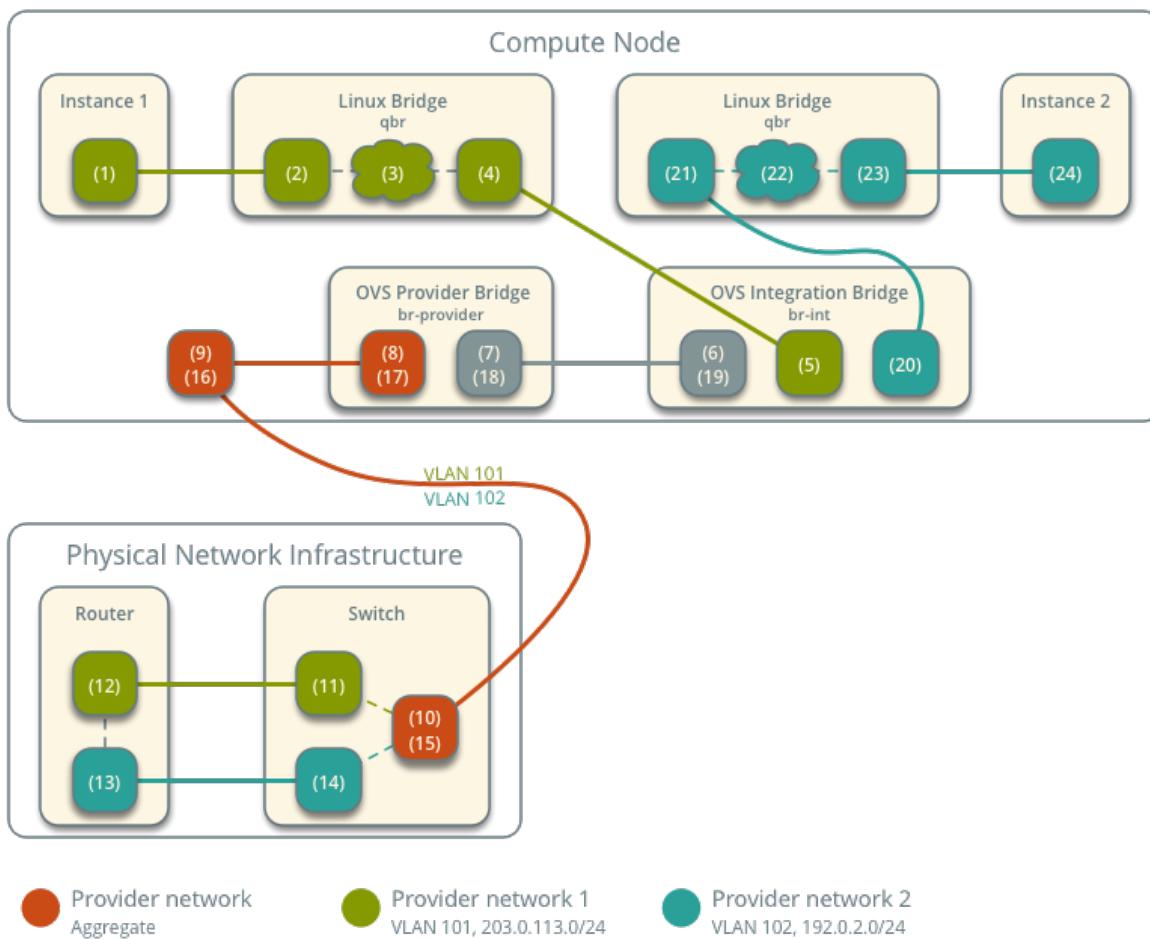
The following steps involve the compute node:

1. The physical network interface (16) forwards the packet to the OVS provider bridge provider network port (17).
2. The OVS provider bridge `phy-br-provider` patch port (18) forwards the packet to the OVS integration bridge `int-br-provider` patch port (19).
3. The OVS integration bridge swaps the actual VLAN tag 102 with the internal VLAN tag.

4. The OVS integration bridge security group port (20) removes the internal VLAN tag and forwards the packet to the security group bridge OVS port (21).
5. Security group rules (22) on the security group bridge handle firewalling and connection tracking for the packet.
6. The security group bridge instance port (23) forwards the packet to the instance 2 interface (24) via veth pair.

Open vSwitch - Provider Networks

Network Traffic Flow - East/West Scenario 2



Note: Return traffic follows similar steps in reverse.

Open vSwitch: Self-service networks

This architecture example augments [Open vSwitch: Provider networks](#) to support a nearly limitless quantity of entirely virtual networks. Although the Networking service supports VLAN self-service networks, this example focuses on VXLAN self-service networks. For more information on self-service networks, see [Self-service networks](#).

Prerequisites

Add one network node with the following components:

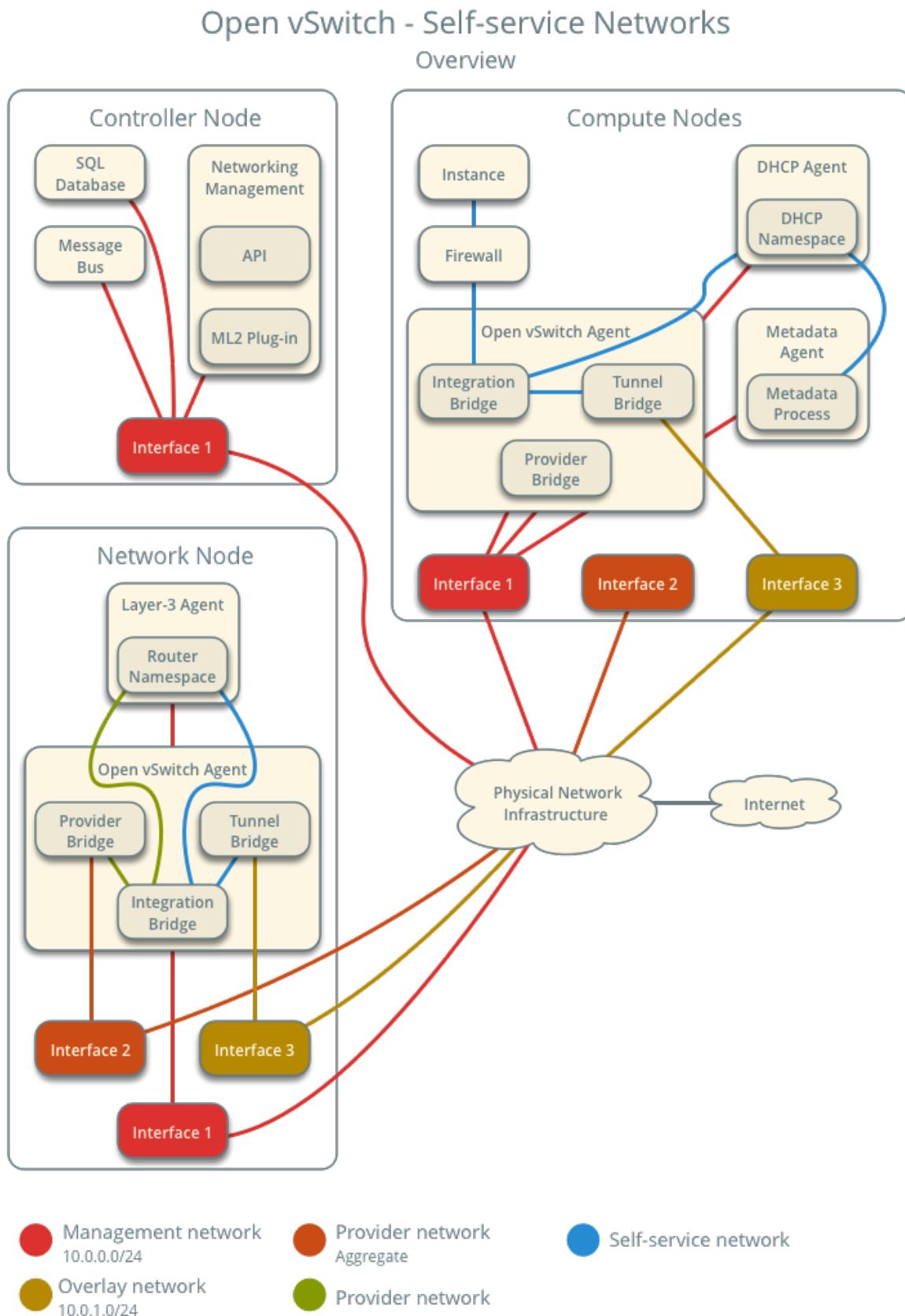
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking Open vSwitch (OVS) layer-2 agent, layer-3 agent, and any including OVS.

Modify the compute nodes with the following components:

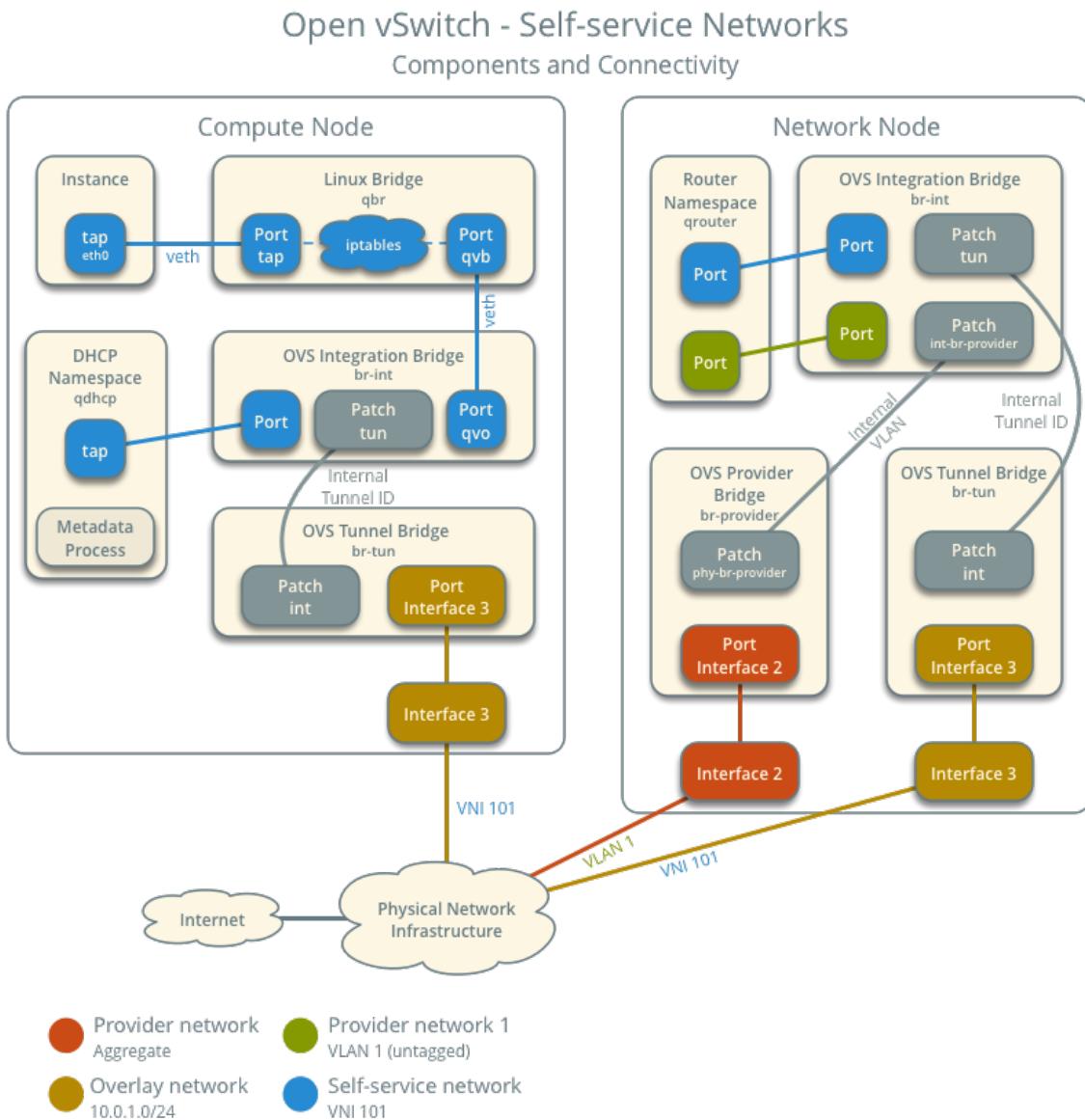
- Add one network interface: overlay.

Note: You can keep the DHCP and metadata agents on each compute node or move them to the network node.

Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) provider network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and with a port on the OVS integration bridge.



Example configuration

Use the following example configuration as a template to add support for self-service networks to an existing operational environment that supports provider networks.

Controller node

1. In the `neutron.conf` file:
 - Enable routing and allow overlapping IP address ranges.

```
[DEFAULT]
service_plugins = router
allow_overlapping_ips = True
```

2. In the `ml2_conf.ini` file:

- Add vxlan to type drivers and project network types.

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
```

- Enable the layer-2 population mechanism driver.

```
[ml2]
mechanism_drivers = openvswitch,l2population
```

- Configure the VXLAN network ID (VNI) range.

```
[ml2_type_vxlan]
vni_ranges = VNI_START:VNI_END
```

Replace `VNI_START` and `VNI_END` with appropriate numerical values.

3. Restart the following services:

- Neutron Server
- Open vSwitch agent

Network node

1. Install the Networking service OVS layer-2 agent and layer-3 agent.
2. Install OVS.
3. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
# ...

[keystone_auth_token]
# ...

[nova]
# ...

[agent]
# ...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone_auth_token], [nova], and [agent] sections.

4. Start the following services:
 - OVS
5. Create the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-br br-provider
```

6. In the `openvswitch_agent.ini` file, configure the layer-2 agent.

```
[ovs]
bridge_mappings = provider:br-provider
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = True

[securitygroup]
firewall_driver = iptables_hybrid
```

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

7. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
external_network_bridge =
```

Note: The `external_network_bridge` option intentionally contains no value.

8. Start the following services:
 - Open vSwitch agent
 - Layer-3 agent

Compute nodes

1. In the `openvswitch_agent.ini` file, enable VXLAN support including layer-2 population.

```
[ovs]
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = True
```

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

2. Restart the following services:
 - Open vSwitch agent

Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID          | Agent Type      | Host     | Availability |
| Zone        | Alive | State | Binary           |
+-----+-----+-----+-----+
| 1236bbcb-e0ba-48a9-80fc-81202ca4fa51 | Metadata agent    | compute2  |   |
|       | True  | UP    | neutron-metadata-agent |
| 457d6898-b373-4bb3-b41f-59345dcfb5c5 | Open vSwitch agent | compute2  |   |
|       | True  | UP    | neutron-openvswitch-agent |
| 71f15e84-bc47-4c2a-b9fb-317840b2d753 | DHCP agent        | compute2  | nova |
|       | True  | UP    | neutron-dhcp-agent    |
| 8805b962-de95-4e40-bdc2-7a0add7521e8 | L3 agent         | network1 | nova |
|       | True  | UP    | neutron-l3-agent     |
| a33cac5a-0266-48f6-9cac-4cef4f8b0358 | Open vSwitch agent | network1 |   |
|       | True  | UP    | neutron-openvswitch-agent |
| a6c69690-e7f7-4e56-9831-1282753e5007 | Metadata agent    | compute1  |   |
|       | True  | UP    | neutron-metadata-agent |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68 | DHCP agent        | compute1  | nova |
|       | True  | UP    | neutron-dhcp-agent    |
| bcf977b-ec0e-4ba9-be62-9489b4b0e6f1 | Open vSwitch agent | compute1  |   |
|       | True  | UP    | neutron-openvswitch-agent |
+-----+-----+-----+-----+
```

Create initial networks

The configuration supports multiple VXLAN self-service networks. For simplicity, the following procedure creates one self-service network and a router with a gateway on the flat provider network. The router uses NAT for IPv4 network traffic and directly routes IPv6 network traffic.

Note: IPv6 connectivity with self-service networks often requires addition of static routes to nodes and physical network infrastructure.

1. Source the administrative project credentials.
2. Update the provider network to support external connectivity for self-service networks.

```
$ openstack network set --external provider1
```

Note: This command provides no output.

3. Source a regular (non-administrative) project credentials.
4. Create a self-service network.

```
$ openstack network create selfservice1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| mtu | 1450 |
| name | selfservice1 |
| port_security_enabled | True |
| router:external | Internal |
| shared | False |
| status | ACTIVE |
+-----+-----+
```

5. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
--network selfservice1 --dns-nameserver 8.8.4.4 selfservice1-v4
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 192.0.2.2-192.0.2.254 |
| cidr | 192.0.2.0/24 |
| dns_nameservers | 8.8.4.4 |
| enable_dhcp | True |
| gateway_ip | 192.0.2.1 |
| ip_version | 4 |
| name | selfservice1-v4 |
+-----+-----+
```

6. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-version 6 \
--ipv6-ra-mode slaac --ipv6-address-mode slaac --network selfservice1 \
--dns-nameserver 2001:4860:4860::8844 selfservice1-v6
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | fd00:192:0:2::2-fd00:192:0:2:ffff:ffff:ffff:ffff |
| cidr | fd00:192:0:2::/64 |
| dns_nameservers | 2001:4860:4860::8844 |
| enable_dhcp | True |
| gateway_ip | fd00:192:0:2::1 |
| ip_version | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode | slaac |
| name | selfservice1-v6 |
+-----+-----+
```

7. Create a router.

```
$ openstack router create router1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| name | router1 |
| status | ACTIVE |
+-----+-----+
```

- ```
+-----+-----+
```
8. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router1 selfservice1-v4
$ openstack router add subnet router1 selfservice1-v6
```

---

**Note:** These commands provide no output.

---

9. Add the provider network as the gateway on the router.

```
$ neutron router-gateway-set router1 provider1
Set gateway for router router1
```

## Verify network operation

1. On each compute node, verify creation of a second qdhcp namespace.

```
ip netns
qdhcp-8b868082-e312-4110-8627-298109d4401c
qdhcp-8fbc13ca-cfe0-4b8a-993b-e33f37ba66d1
```

2. On the network node, verify creation of the qrouter namespace.

```
ip netns
qrouter-17db2a15-e024-46d0-9250-4cd4d336a2cc
```

3. Source a regular (non-administrative) project credentials.

4. Create the appropriate security group rules to allow ping and SSH access instances using the network.

```
$ openstack security group rule create --proto icmp default
+-----+-----+
| Field | Value |
+-----+-----+
direction	ingress
ethertype	IPv4
protocol	icmp
remote_ip_prefix	0.0.0.0/0
+-----+-----+

$ openstack security group rule create --ethertype IPv6 --proto ipv6-icmp default
+-----+-----+
| Field | Value |
+-----+-----+
direction	ingress
ethertype	IPv6
protocol	ipv6-icmp
+-----+-----+

$ openstack security group rule create --proto tcp --dst-port 22 default
+-----+-----+
| Field | Value |
```

```
+-----+-----+
direction	ingress
ethertype	IPv4
port_range_max	22
port_range_min	22
protocol	tcp
remote_ip_prefix	0.0.0.0/0
+-----+-----+	
$ openstack security group rule create --ethertype IPv6 --proto tcp --dst-port 22	
→default	
+-----+-----+	
Field	Value
+-----+-----+	
direction	ingress
ethertype	IPv6
port_range_max	22
port_range_min	22
protocol	tcp
+-----+-----+
```

5. Launch an instance with an interface on the self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-id=NETWORK_ID
→selfservice-instance1
```

Replace NETWORK\_ID with the ID of the self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID | Name | Status | Networks |
+-----+-----+-----+
| c055cdb0-ebb4-4d65-957c-35cbdbd59306 | selfservice-instance1 | ACTIVE | |
| selfservice1=192.0.2.4, fd00:192:0:2:f816:3eff:fe30:9cb0 |
+-----+-----+-----+
```

**Warning:** The IPv4 address resides in a private IP address range (RFC1918). Thus, the Networking service performs source network address translation (SNAT) for the instance to access external networks such as the Internet. Access from external networks such as the Internet to the instance requires a floating IPv4 address. The Networking service performs destination network address translation (DNAT) from the floating IPv4 address to the instance IPv4 address on the self-service network. On the other hand, the Networking service architecture for IPv6 lacks support for NAT due to the significantly larger address space and complexity of NAT. Thus, floating IP addresses do not exist for IPv6 and the Networking service only performs routing for IPv6 subnets on self-service networks. In other words, you cannot rely on NAT to “hide” instances with IPv4 and IPv6 addresses or only IPv6 addresses and must properly implement security groups to restrict access.

7. On the controller node or any host with access to the provider network, ping the IPv6 address of the instance.

```
$ ping6 -c 4 fd00:192:0:2:f816:3eff:fe30:9cb0
PING fd00:192:0:2:f816:3eff:fe30:9cb0(fd00:192:0:2:f816:3eff:fe30:9cb0) 56 data bytes
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=1 ttl=63 time=2.08 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=2 ttl=63 time=1.88 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=3 ttl=63 time=1.55 ms
64 bytes from fd00:192:0:2:f816:3eff:fe30:9cb0: icmp_seq=4 ttl=63 time=1.62 ms

--- fd00:192:0:2:f816:3eff:fe30:9cb0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.557/1.788/2.085/0.217 ms
```

8. Optionally, enable IPv4 access from external networks such as the Internet to the instance.

- (a) Create a floating IPv4 address on the provider network.

| Field       | Value                                |
|-------------|--------------------------------------|
| fixed_ip    | None                                 |
| id          | 22a1b088-5c9b-43b4-97f3-970ce5df77f2 |
| instance_id | None                                 |
| ip          | 203.0.113.16                         |
| pool        | provider1                            |

- (b) Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance1 203.0.113.16
```

---

**Note:** This command provides no output.

---

- (c) On the controller node or any host with access to the provider network, ping the floating IPv4 address of the instance.

```
$ ping -c 4 203.0.113.16
PING 203.0.113.16 (203.0.113.16) 56(84) bytes of data.
64 bytes from 203.0.113.16: icmp_seq=1 ttl=63 time=3.41 ms
64 bytes from 203.0.113.16: icmp_seq=2 ttl=63 time=1.67 ms
64 bytes from 203.0.113.16: icmp_seq=3 ttl=63 time=1.47 ms
64 bytes from 203.0.113.16: icmp_seq=4 ttl=63 time=1.59 ms

--- 203.0.113.16 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.473/2.040/3.414/0.798 ms
```

9. Obtain access to the instance.

10. Test IPv4 and IPv6 connectivity to the Internet or other external network.

## Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
  - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
  - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
  - VXLAN ID (VNI) 102
- Self-service router
  - Gateway on the provider network
  - Interface on self-service network 1
  - Interface on self-service network 2
- Instance 1
- Instance 2

### North-south scenario 1: Instance with a fixed IP address

For instances with a fixed IPv4 address, the network node performs SNAT on north-south traffic passing from self-service to external networks such as the Internet. For instances with a fixed IPv6 address, the network node performs conventional routing of traffic between self-service and external networks.

- The instance resides on compute node 1 and uses self-service network 1.
- The instance sends a packet to a host on the Internet.

The following steps involve compute node 1:

1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
6. The OVS integration bridge patch port (6) forwards the packet to the OVS tunnel bridge patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.

8. The underlying physical interface (9) for overlay networks forwards the packet to the network node via the overlay network (10).

The following steps involve the network node:

1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch port (13) forwards the packet to the OVS integration bridge patch port (14).
5. The OVS integration bridge port for the self-service network (15) removes the internal VLAN tag and forwards the packet to the self-service network interface (16) in the router namespace.
  - For IPv4, the router performs SNAT on the packet which changes the source IP address to the router IP address on the provider network and sends it to the gateway IP address on the provider network via the gateway interface on the provider network (17).
  - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the provider network, via the provider gateway interface (17).
6. The router forwards the packet to the OVS integration bridge port for the provider network (18).
7. The OVS integration bridge adds the internal VLAN tag to the packet.
8. The OVS integration bridge `int-br-provider` patch port (19) forwards the packet to the OVS provider bridge `phy-br-provider` patch port (20).
9. The OVS provider bridge swaps the internal VLAN tag with actual VLAN tag 101.
10. The OVS provider bridge provider network port (21) forwards the packet to the physical network interface (22).
11. The physical network interface forwards the packet to the Internet via physical network infrastructure (23).

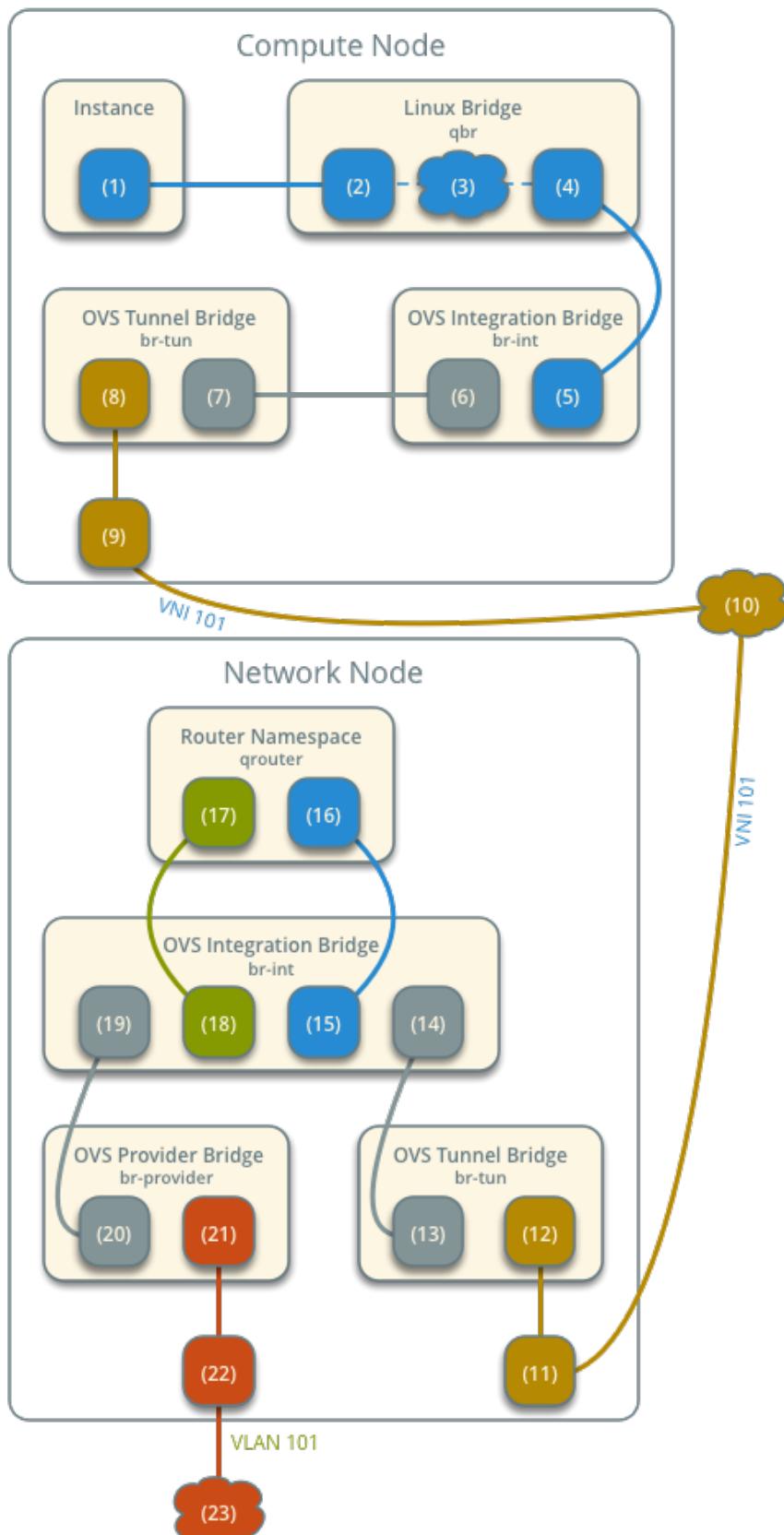
---

**Note:** Return traffic follows similar steps in reverse. However, without a floating IPv4 address, hosts on the provider or external networks cannot originate connections to instances on the self-service network.

---

## Open vSwitch - Self-service Networks

### Network Traffic Flow - North/South Scenario 1



### North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address, the network node performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario.

- The instance resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

The following steps involve the network node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface (2).
2. The provider physical network interface forwards the packet to the OVS provider bridge provider network port (3).
3. The OVS provider bridge swaps actual VLAN tag 101 with the internal VLAN tag.
4. The OVS provider bridge phy-br-provider port (4) forwards the packet to the OVS integration bridge int-br-provider port (5).
5. The OVS integration bridge port for the provider network (6) removes the internal VLAN tag and forwards the packet to the provider network interface (6) in the router namespace.
  - For IPv4, the router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network and sends it to the gateway IP address on the self-service network via the self-service interface (7).
  - For IPv6, the router sends the packet to the next-hop IP address, typically the gateway IP address on the self-service network, via the self-service interface (8).
6. The router forwards the packet to the OVS integration bridge port for the self-service network (9).
7. The OVS integration bridge adds an internal VLAN tag to the packet.
8. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
9. The OVS integration bridge patch-tun patch port (10) forwards the packet to the OVS tunnel bridge patch-int patch port (11).
10. The OVS tunnel bridge (12) wraps the packet using VNI 101.
11. The underlying physical interface (13) for overlay networks forwards the packet to the network node via the overlay network (14).

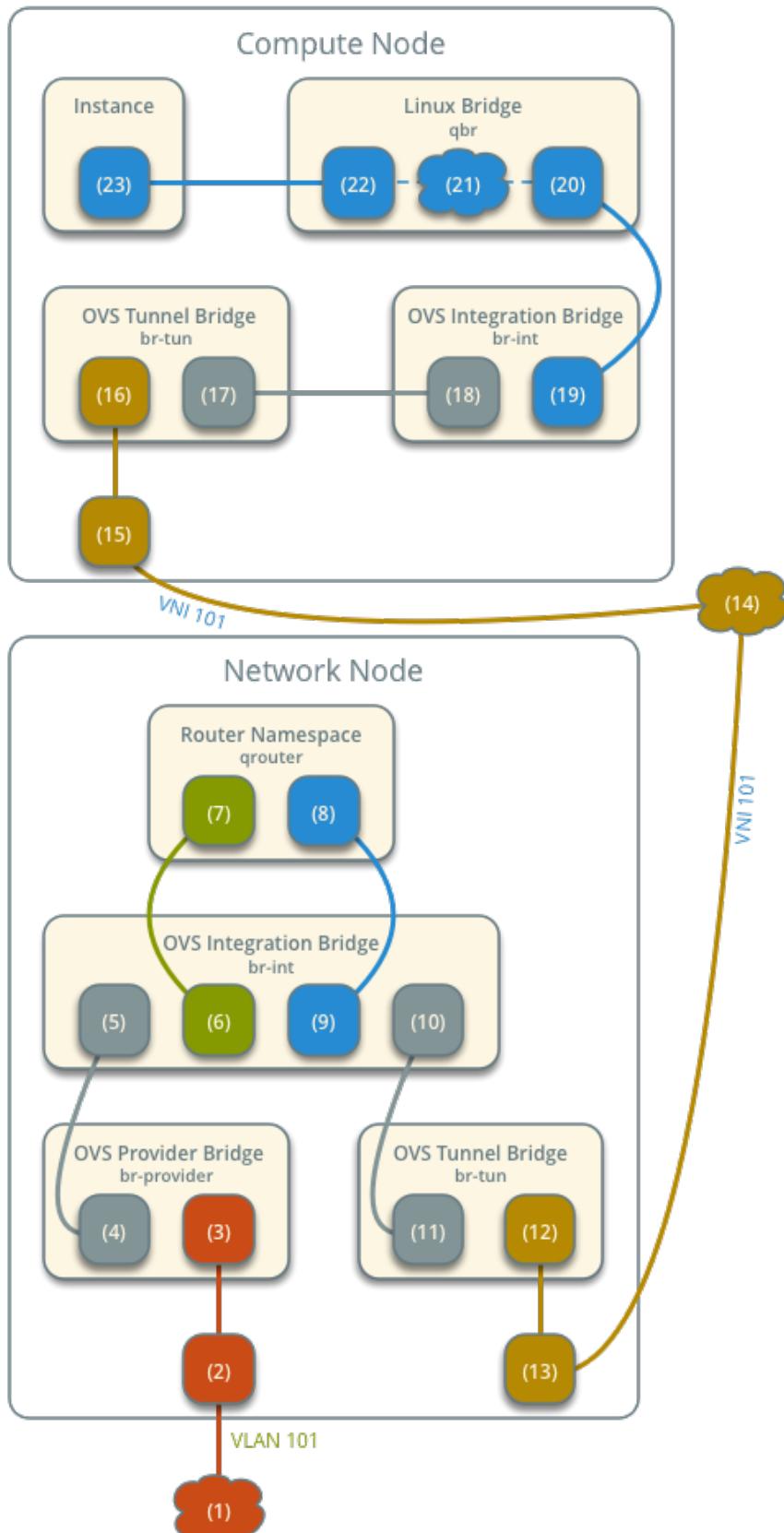
The following steps involve the compute node:

1. The underlying physical interface (15) for overlay networks forwards the packet to the OVS tunnel bridge (16).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch-int patch port (17) forwards the packet to the OVS integration bridge patch-tun patch port (18).
5. The OVS integration bridge removes the internal VLAN tag from the packet.

6. The OVS integration bridge security group port (19) forwards the packet to the security group bridge OVS port (20) via veth pair.
7. Security group rules (21) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (22) forwards the packet to the instance interface (23) via veth pair.

## Open vSwitch - Self-service Networks

### Network Traffic Flow - North/South Scenario 2



---

**Note:** Egress instance traffic flows similar to north-south scenario 1, except SNAT changes the source IP address of the packet to the floating IPv4 address rather than the router IP address on the provider network.

---

### East-west scenario 1: Instances on the same network

Instances with a fixed IPv4/IPv6 address or floating IPv4 address on the same network communicate directly between compute nodes containing those instances.

By default, the VXLAN protocol lacks knowledge of target location and uses multicast to discover it. After discovery, it stores the location in the local forwarding database. In large deployments, the discovery process can generate a significant amount of network that all nodes must process. To eliminate the latter and generally increase efficiency, the Networking service includes the layer-2 population mechanism driver that automatically populates the forwarding database for VXLAN interfaces. The example configuration enables this driver. For more information, see [ML2 plug-in](#).

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 2 and uses self-service network 1.
- Instance 1 sends a packet to instance 2.

The following steps involve compute node 1:

1. The instance 1 interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
6. The OVS integration bridge patch port (6) forwards the packet to the OVS tunnel bridge patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.
8. The underlying physical interface (9) for overlay networks forwards the packet to compute node 2 via the overlay network (10).

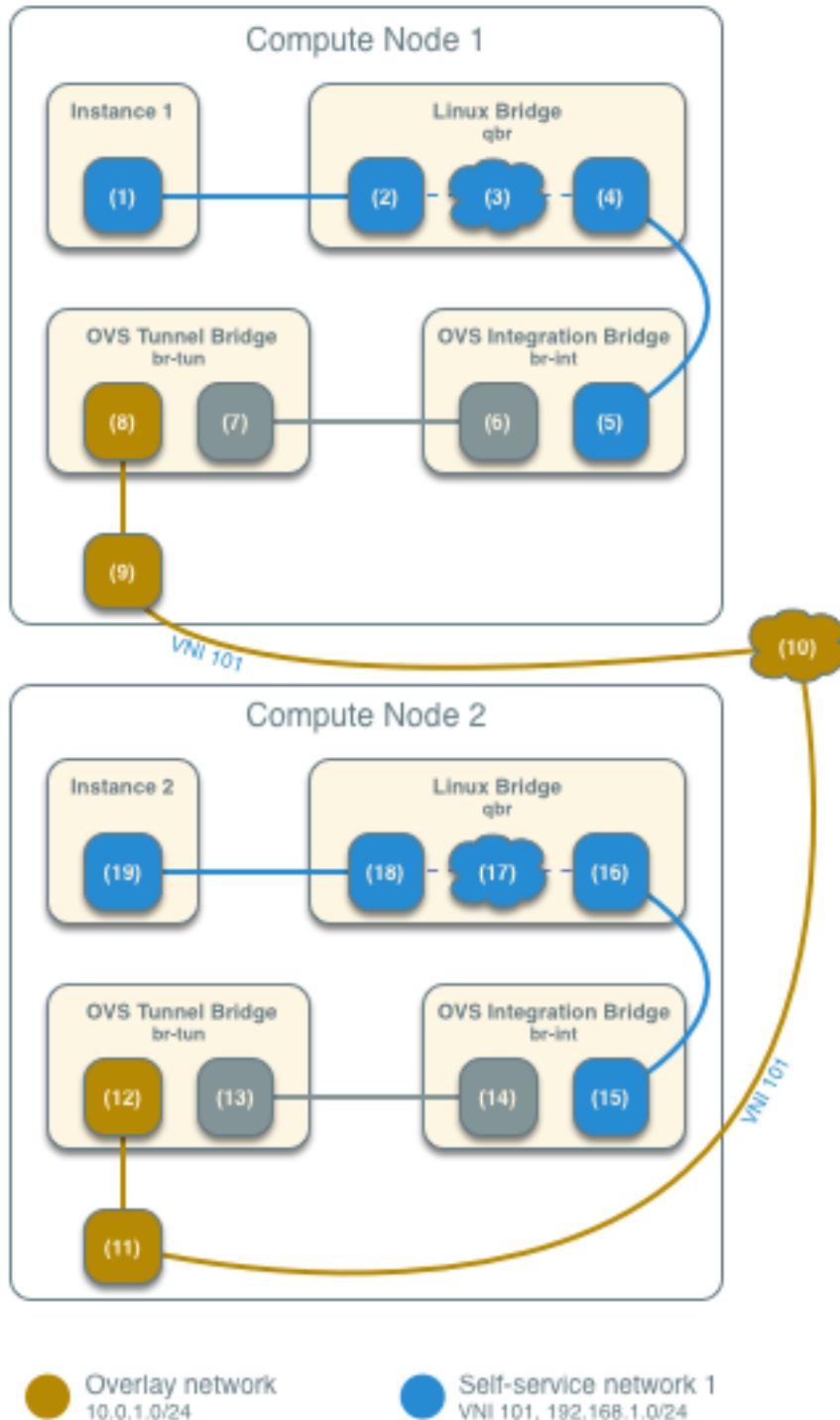
The following steps involve compute node 2:

1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch-int patch port (13) forwards the packet to the OVS integration bridge patch-tun patch port (14).
5. The OVS integration bridge removes the internal VLAN tag from the packet.

6. The OVS integration bridge security group port (15) forwards the packet to the security group bridge OVS port (16) via veth pair.
7. Security group rules (17) on the security group bridge handle firewalling and connection tracking for the packet.
8. The security group bridge instance port (18) forwards the packet to the instance 2 interface (19) via veth pair.

## Open vSwitch - Self-service Networks

Network Traffic Flow - East/West Scenario 1



**Note:** Return traffic follows similar steps in reverse.

### East-west scenario 2: Instances on different networks

Instances using a fixed IPv4/IPv6 address or floating IPv4 address communicate via router on the network node. The self-service networks must reside on the same router.

- Instance 1 resides on compute node 1 and uses self-service network 1.
- Instance 2 resides on compute node 1 and uses self-service network 2.
- Instance 1 sends a packet to instance 2.

---

**Note:** Both instances reside on the same compute node to illustrate how VXLAN enables multiple overlays to use the same layer-3 network.

---

The following steps involve the compute node:

1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group port (5) via veth pair.
4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
6. The OVS integration bridge patch-tun patch port (6) forwards the packet to the OVS tunnel bridge patch-int patch port (7).
7. The OVS tunnel bridge (8) wraps the packet using VNI 101.
8. The underlying physical interface (9) for overlay networks forwards the packet to the network node via the overlay network (10).

The following steps involve the network node:

1. The underlying physical interface (11) for overlay networks forwards the packet to the OVS tunnel bridge (12).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch-int patch port (13) forwards the packet to the OVS integration bridge patch-tun patch port (14).
5. The OVS integration bridge port for self-service network 1 (15) removes the internal VLAN tag and forwards the packet to the self-service network 1 interface (16) in the router namespace.
6. The router sends the packet to the next-hop IP address, typically the gateway IP address on self-service network 2, via the self-service network 2 interface (17).
7. The router forwards the packet to the OVS integration bridge port for self-service network 2 (18).
8. The OVS integration bridge adds the internal VLAN tag to the packet.
9. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.

10. The OVS integration bridge patch-tun patch port (19) forwards the packet to the OVS tunnel bridge patch-int patch port (20).
11. The OVS tunnel bridge (21) wraps the packet using VNI 102.
12. The underlying physical interface (22) for overlay networks forwards the packet to the compute node via the overlay network (23).

The following steps involve the compute node:

1. The underlying physical interface (24) for overlay networks forwards the packet to the OVS tunnel bridge (25).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch-int patch port (26) forwards the packet to the OVS integration bridge patch-tun patch port (27).
5. The OVS integration bridge removes the internal VLAN tag from the packet.
6. The OVS integration bridge security group port (28) forwards the packet to the security group bridge OVS port (29) via veth pair.
7. Security group rules (30) on the security group bridge handle firewalls and connection tracking for the packet.
8. The security group bridge instance port (31) forwards the packet to the instance interface (32) via veth pair.

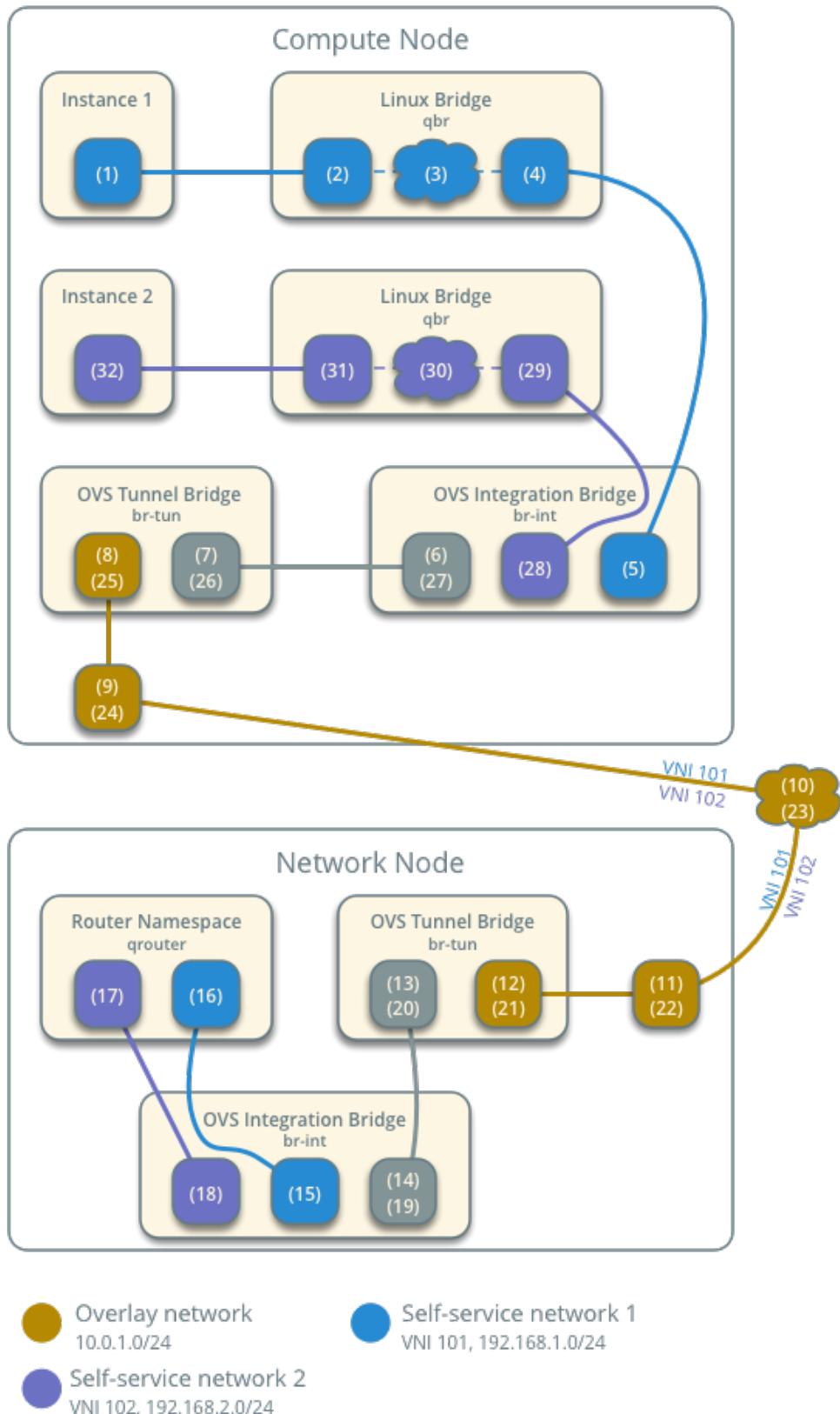
---

**Note:** Return traffic follows similar steps in reverse.

---

## Open vSwitch - Self-service Networks

### Network Traffic Flow - East/West Scenario 2



## Open vSwitch: High availability using VRRP

This architecture example augments the self-service deployment example with a high-availability mechanism using the Virtual Router Redundancy Protocol (VRRP) via keepalived and provides failover of routing for self-service networks. It requires a minimum of two network nodes because VRRP creates one master (active) instance and at least one backup instance of each router.

During normal operation, keepalived on the master router periodically transmits *heartbeat* packets over a hidden network that connects all VRRP routers for a particular project. Each project with VRRP routers uses a separate hidden network. By default this network uses the first value in the `tenant_network_types` option in the `m12.conf.ini` file. For additional control, you can specify the self-service network type and physical network name for the hidden network using the `l3_ha_network_type` and `l3_ha_network_name` options in the `neutron.conf` file.

If keepalived on the backup router stops receiving *heartbeat* packets, it assumes failure of the master router and promotes the backup router to master router by configuring IP addresses on the interfaces in the `qrouting` namespace. In environments with more than one backup router, keepalived on the backup router with the next highest priority promotes that backup router to master router.

---

**Note:** This high-availability mechanism configures VRRP using the same priority for all routers. Therefore, VRRP promotes the backup router with the highest IP address to the master router.

---

**Warning:** There is a known bug with keepalived v1.2.15 and earlier which can cause packet loss when `max_l3_agents_per_router` is set to 3 or more. Therefore, we recommend that you upgrade to keepalived v1.2.16 or greater when using this feature.

Interruption of VRRP *heartbeat* traffic between network nodes, typically due to a network interface or physical network infrastructure failure, triggers a failover. Restarting the layer-3 agent, or failure of it, does not trigger a failover providing keepalived continues to operate.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Instance network traffic on self-service networks using a particular router only traverses the master instance of that router. Thus, resource limitations of a particular network node can impact all master instances of routers on that network node without triggering failover to another network node. However, you can configure the scheduler to distribute the master instance of each router uniformly across a pool of network nodes to reduce the chance of resource contention on any particular network node.
- Only supports self-service networks using a router. Provider networks operate at layer-2 and rely on physical network infrastructure for redundancy.
- For instances with a floating IPv4 address, maintains state of network connections during failover as a side effect of 1:1 static NAT. The mechanism does not actually implement connection tracking.

For production deployments, we recommend at least three network nodes with sufficient resources to handle network traffic for the entire environment if one network node fails. Also, the remaining two nodes can continue to provide redundancy.

## Prerequisites

Add one network node with the following components:

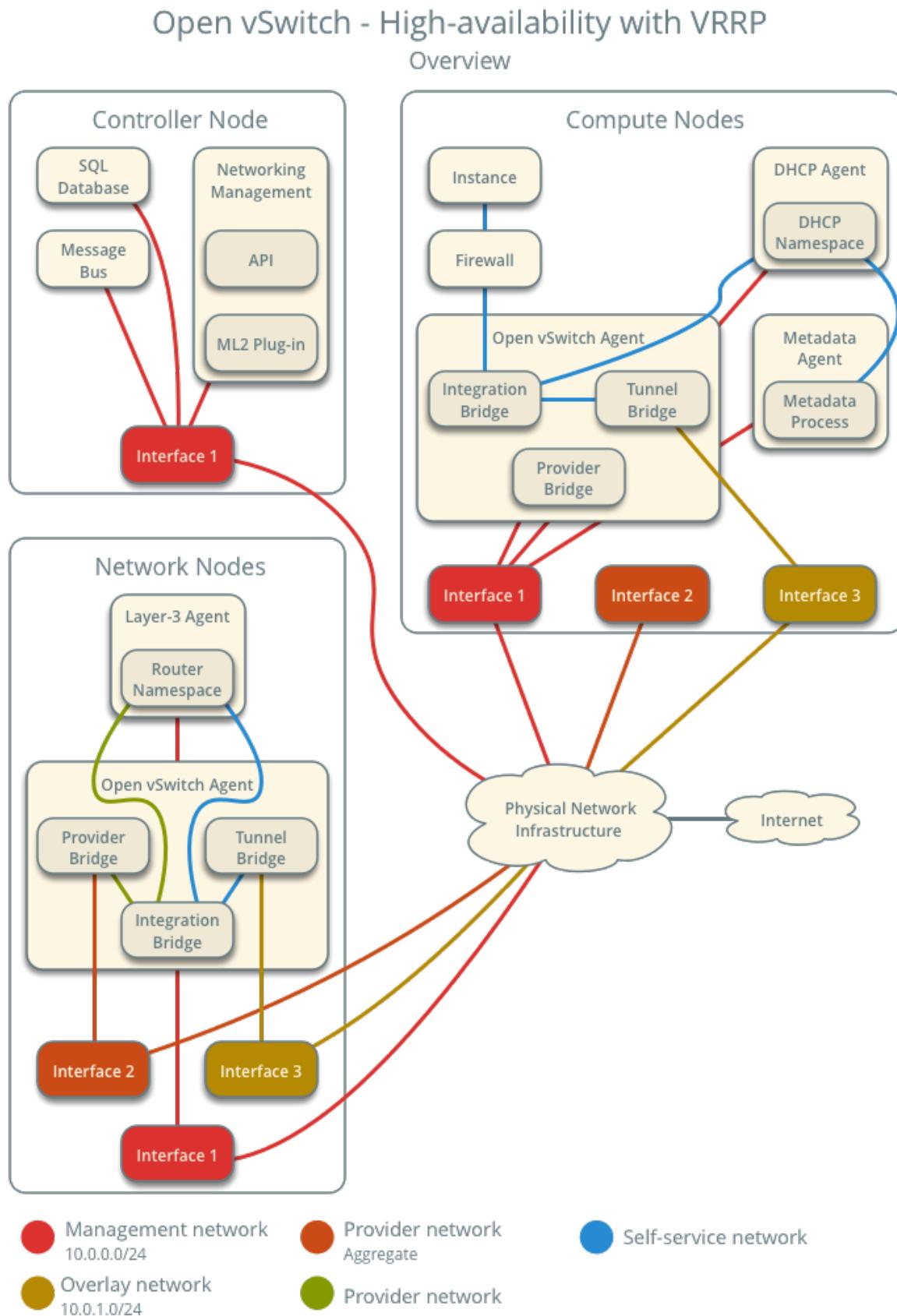
- Three network interfaces: management, provider, and overlay.
- OpenStack Networking layer-2 agent, layer-3 agent, and any dependencies.

---

**Note:** You can keep the DHCP and metadata agents on each compute node or move them to the network nodes.

---

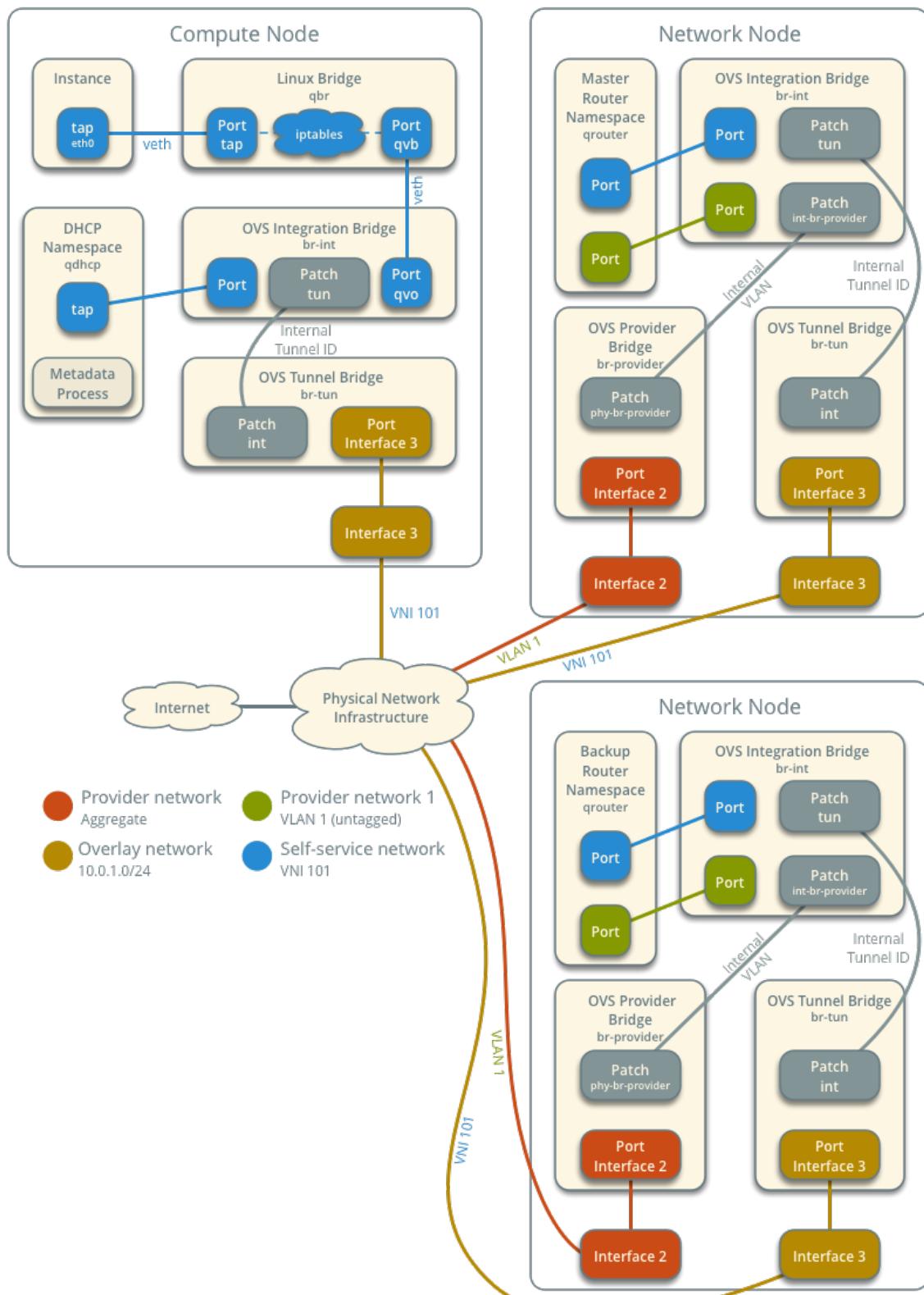
## Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) network. The master router resides on network node 1. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace and Linux bridge with a port on the overlay physical network interface.

## Open vSwitch - High-availability with VRRP

### Components and Connectivity



## Example configuration

Use the following example configuration as a template to add support for high-availability using VRRP to an existing operational environment that supports self-service networks.

### Controller node

1. In the `neutron.conf` file:

- Enable VRRP.

```
[DEFAULT]
l3_ha = True
```

2. Restart the following services:

- Server

### Network node 1

No changes.

### Network node 2

1. Install the Networking service OVS layer-2 agent and layer-3 agent.
2. Install OVS.
3. In the `neutron.conf` file, configure common options:

```
[DEFAULT]
core_plugin = ml2
auth_strategy = keystone

[database]
...

[keystone_auth_token]
...

[nova]
...

[agent]
...
```

See the [Installation Tutorials and Guides](#) and [Configuration Reference](#) for your OpenStack release to obtain the appropriate additional configuration for the [DEFAULT], [database], [keystone\_auth\_token], [nova], and [agent] sections.

4. Start the following services:

- OVS

5. Create the OVS provider bridge `br-provider`:

```
$ ovs-vsctl add-br br-provider
```

6. In the `openvswitch_agent.ini` file, configure the layer-2 agent.

```
[ovs]
bridge_mappings = provider:br-provider
local_ip = OVERLAY_INTERFACE_IP_ADDRESS

[agent]
tunnel_types = vxlan
l2_population = true

[securitygroup]
firewall_driver = iptables_hybrid
```

Replace `OVERLAY_INTERFACE_IP_ADDRESS` with the IP address of the interface that handles VXLAN overlays for self-service networks.

7. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
external_network_bridge =
```

---

**Note:** The `external_network_bridge` option intentionally contains no value.

---

8. Start the following services:

- Open vSwitch agent
- Layer-3 agent

## Compute nodes

No changes.

## Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability |
| Zone | Alive | State | Binary | |
+-----+-----+-----+-----+
1236bbcb-e0ba-48a9-80fc-81202ca4fa51	Metadata agent	compute2		
	True	UP	neutron-metadata-agent	
457d6898-b373-4bb3-b41f-59345dcfb5c5	Open vSwitch agent	compute2		
	True	UP	neutron-openvswitch-agent	
```

|                                                                      |
|----------------------------------------------------------------------|
| 71f15e84-bc47-4c2a-b9fb-317840b2d753   DHCP agent   compute2   nova  |
| True   UP   neutron-dhcp-agent                                       |
| 8805b962-de95-4e40-bdc2-7a0add7521e8   L3 agent   network1   nova    |
| True   UP   neutron-l3-agent                                         |
| a33cac5a-0266-48f6-9cac-4cef4f8b0358   Open vSwitch agent   network1 |
| True   UP   neutron-openvswitch-agent                                |
| a6c69690-e7f7-4e56-9831-1282753e5007   Metadata agent   compute1     |
| True   UP   neutron-metadata-agent                                   |
| af11f22f-a9f4-404f-9fd8-cd7ad55c0f68   DHCP agent   compute1   nova  |
| True   UP   neutron-dhcp-agent                                       |
| bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1   Open vSwitch agent   compute1 |
| True   UP   neutron-openvswitch-agent                                |
| 7f00d759-f2c9-494a-9fbf-fd9118104d03   Open vSwitch agent   network2 |
| True   UP   neutron-openvswitch-agent                                |
| b28d8818-9e32-4888-930b-29addbdd2ef9   L3 agent   network2   nova    |
| True   UP   neutron-l3-agent                                         |
| +-----+-----+-----+-----+                                            |
| -----+-----+-----+-----+                                             |

### Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use VRRP. The following procedure creates an additional self-service network and router. The Networking service also supports adding high-availability to existing routers. However, the procedure requires administratively disabling and enabling each router which temporarily interrupts network connectivity for self-service networks with interfaces on that router.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

| \$ openstack network create selfservice2 |              |  |
|------------------------------------------|--------------|--|
| Field                                    | Value        |  |
| admin_state_up                           | UP           |  |
| mtu                                      | 1450         |  |
| name                                     | selfservice2 |  |
| port_security_enabled                    | True         |  |
| router:external                          | Internal     |  |
| shared                                   | False        |  |
| status                                   | ACTIVE       |  |

3. Create a IPv4 subnet on the self-service network.

| \$ openstack subnet create --subnet-range 198.51.100.0/24 \\\n--network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4 |                             |  |
|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------|--|
| Field                                                                                                                         | Value                       |  |
| allocation_pools                                                                                                              | 198.51.100.2-198.51.100.254 |  |
| cidr                                                                                                                          | 198.51.100.0/24             |  |
| dns_nameservers                                                                                                               | 8.8.4.4                     |  |
| enable_dhcp                                                                                                                   | True                        |  |

|               |                 |  |
|---------------|-----------------|--|
| gateway_ip    | 198.51.100.1    |  |
| ip_version    | 4               |  |
| name          | selfservice2-v4 |  |
| +-----+-----+ |                 |  |

4. Create a IPv6 subnet on the self-service network.

|                                                                                 |
|---------------------------------------------------------------------------------|
| \$ openstack subnet create --subnet-range fd00:198:51:100::/64 --ip-version 6 \ |
| --ipv6-ra-mode slaac --ipv6-address-mode slaac --network selfservice2 \         |
| --dns-nameserver 2001:4860:4860::8844 selfservice2-v6                           |
| +-----+-----+                                                                   |
| Field   Value                                                                   |
| +-----+-----+                                                                   |
| allocation_pools   fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff       |
| cidr   fd00:198:51:100::/64                                                     |
| dns_nameservers   2001:4860:4860::8844                                          |
| enable_dhcp   True                                                              |
| gateway_ip   fd00:198:51:100::1                                                 |
| ip_version   6                                                                  |
| ipv6_address_mode   slaac                                                       |
| ipv6_ra_mode   slaac                                                            |
| name   selfservice2-v6                                                          |
| +-----+-----+                                                                   |

5. Create a router.

|                                    |
|------------------------------------|
| \$ openstack router create router2 |
| +-----+-----+                      |
| Field   Value                      |
| +-----+-----+                      |
| admin_state_up   UP                |
| name   router2                     |
| status   ACTIVE                    |
| +-----+-----+                      |

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4
$ openstack router add subnet router2 selfservice2-v6
```

---

**Note:** These commands provide no output.

---

7. Add the provider network as a gateway on the router.

```
$ neutron router-gateway-set router2 provider1
Set gateway for router router2
```

## Verify network operation

1. Source the administrative project credentials.
2. Verify creation of the internal high-availability network that handles VRRP *heartbeat* traffic.

```
$ openstack network list
+-----+-----+
| ID | Name |
+-----+-----+
| Subnets |
+-----+-----+
| 1b8519c1-59c4-415c-9da2-a67d53c68455 | HA network tenant |
| f986edf55ae945e2bef3cb4bfd589928 | 6843314a-1e76-4cc9-94f5-c64b7a39364a |
+-----+-----+
```

3. On each network node, verify creation of a qrouter namespace with the same ID.

Network node 1:

```
ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

Network node 2:

```
ip netns
qrouter-b6206312-878e-497c-8ef7-eb384f8add96
```

---

**Note:** The namespace for router 1 from *Linux bridge: Self-service networks* should only appear on network node 1 because of creation prior to enabling VRRP.

---

4. On each network node, show the IP address of interfaces in the qrouter namespace. With the exception of the VRRP interface, only one namespace belonging to the master router instance contains IP addresses on the interfaces.

Network node 1:

```
ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: ha-eb820380-40@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
 state UP group default qlen 1000
 link/ether fa:16:3e:78:ba:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet 169.254.192.1/18 brd 169.254.255.255 scope global ha-eb820380-40
 valid_lft forever preferred_lft forever
 inet 169.254.0.1/24 scope global ha-eb820380-40
 valid_lft forever preferred_lft forever
 inet6 fe80::f816:3eff:fe78:ba99/64 scope link
 valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
 state UP group default qlen 1000
 link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet 198.51.100.1/24 scope global qr-da3504ad-ba
 valid_lft forever preferred_lft forever
```

```

inet6 fe80::f816:3eff:fedc:8ea8/64 scope link
 valid_lft forever preferred_lft forever
4: qr-442e36eb-fc@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
 ↳state UP group default qlen 1000
 link/ether fa:16:3e:ee:c8:41 brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet6 fd00:198:51:100::1/64 scope global nodad
 valid_lft forever preferred_lft forever
 inet6 fe80::f816:3eff:feee:c841/64 scope link
 valid_lft forever preferred_lft forever
5: qg-33fedbc5-43@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
 ↳state UP group default qlen 1000
 link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet 203.0.113.21/24 scope global qg-33fedbc5-43
 valid_lft forever preferred_lft forever
 inet6 fd00:203:0:113::21/64 scope global nodad
 valid_lft forever preferred_lft forever
 inet6 fe80::f816:3eff:fe03:1af6/64 scope link
 valid_lft forever preferred_lft forever

```

Network node 2:

```

ip netns exec qrouter-b6206312-878e-497c-8ef7-eb384f8add96 ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 ↳qlen 1
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: ha-7a7ce184-36@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state
 ↳UP group default qlen 1000
 link/ether fa:16:3e:16:59:84 brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet 169.254.192.2/18 brd 169.254.255.255 scope global ha-7a7ce184-36
 valid_lft forever preferred_lft forever
 inet6 fe80::f816:3eff:fe16:5984/64 scope link
 valid_lft forever preferred_lft forever
3: qr-da3504ad-ba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
 ↳state UP group default qlen 1000
 link/ether fa:16:3e:dc:8e:a8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
4: qr-442e36eb-fc@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
 ↳state UP group default qlen 1000
5: qg-33fedbc5-43@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
 ↳state UP group default qlen 1000
 link/ether fa:16:3e:03:1a:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0

```

---

**Note:** The master router may reside on network node 2.

---

5. Launch an instance with an interface on the additional self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-id=NETWORK_ID
 ↳selfservice-instance2
```

Replace NETWORK\_ID with the ID of the additional self-service network.

6. Determine the IPv4 and IPv6 addresses of the instance.

```
$ openstack server list
+-----+-----+-----+
| ID | Name | Status | Networks |
+-----+-----+-----+
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 | ACTIVE | |
| selfservice2=fd00:198:51:100:f816:3eff:fe71:e93e, 198.51.100.4 |
+-----+-----+-----+
```

7. Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-----+-----+
| Field | Value |
+-----+-----+
fixed_ip	None
id	0174056a-fa56-4403-b1ea-b5151a31191f
instance_id	None
ip	203.0.113.17
pool	provider1
+-----+-----+
```

8. Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

---

**Note:** This command provides no output.

---

## Verify failover operation

1. Begin a continuous ping of both the floating IPv4 address and IPv6 address of the instance. While performing the next three steps, you should see a minimal, if any, interruption of connectivity to the instance.
2. On the network node with the master router, administratively disable the overlay network interface.
3. On the other network node, verify promotion of the backup router to master router by noting addition of IP addresses to the interfaces in the qrouter namespace.
4. On the original network node in step 2, administratively enable the overlay network interface. Note that the master router remains on the network node in step 3.

## Keepalived VRRP health check

The health of your keepalived instances can be automatically monitored via a bash script that verifies connectivity to all available and configured gateway addresses. In the event that connectivity is lost, the master router is rescheduled to another node.

If all routers lose connectivity simultaneously, the process of selecting a new master router will be repeated in a round-robin fashion until one or more routers have their connectivity restored.

To enable this feature, edit the `l3_agent.ini` file:

```
ha_vrrp_health_check_interval = 30
```

Where `ha_vrrp_health_check_interval` indicates how often in seconds the health check should run. The default value is 0, which indicates that the check should not run at all.

## Network traffic flow

This high-availability mechanism simply augments [Open vSwitch: Self-service networks](#) with failover of layer-3 services to another router if the master router fails. Thus, you can reference [Self-service network traffic flow](#) for normal operation.

### Open vSwitch: High availability using DVR

This architecture example augments the self-service deployment example with the Distributed Virtual Router (DVR) high-availability mechanism that provides connectivity between self-service and provider networks on compute nodes rather than network nodes for specific scenarios. For instances with a floating IPv4 address, routing between self-service and provider networks resides completely on the compute nodes to eliminate single point of failure and performance issues with network nodes. Routing also resides completely on the compute nodes for instances with a fixed or floating IPv4 address using self-service networks on the same distributed virtual router. However, instances with a fixed IP address still rely on the network node for routing and SNAT services between self-service and provider networks.

Consider the following attributes of this high-availability mechanism to determine practicality in your environment:

- Only provides connectivity to an instance via the compute node on which the instance resides if the instance resides on a self-service network with a floating IPv4 address. Instances on self-service networks with only an IPv6 address or both IPv4 and IPv6 addresses rely on the network node for IPv6 connectivity.
- The instance of a router on each compute node consumes an IPv4 address on the provider network on which it contains a gateway.

## Prerequisites

Modify the compute nodes with the following components:

- Install the OpenStack Networking layer-3 agent.

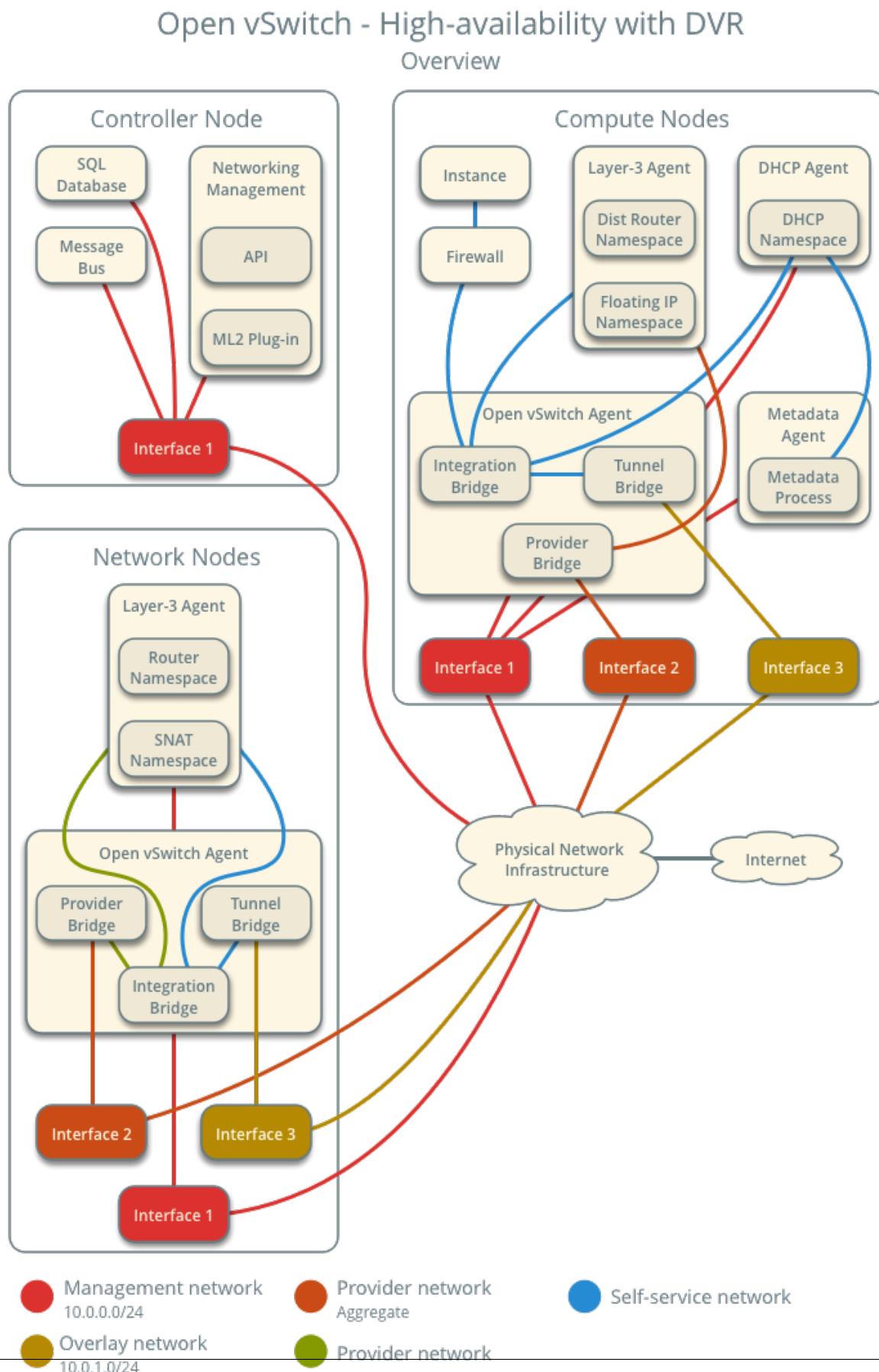
---

**Note:** Consider adding at least one additional network node to provide high-availability for instances with a fixed IP address. See [Distributed Virtual Routing with VRRP](#) for more information.

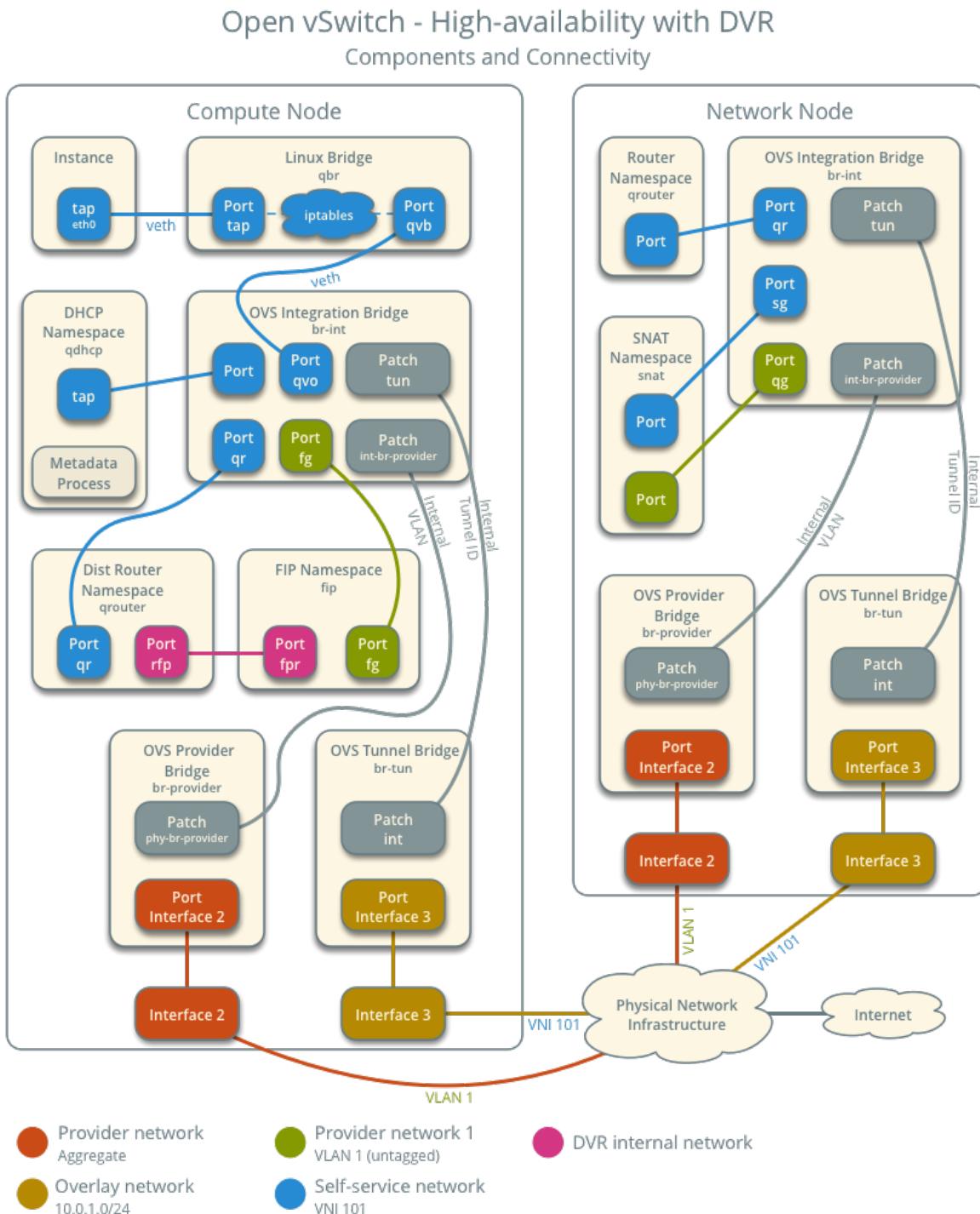
---



## Architecture



The following figure shows components and connectivity for one self-service network and one untagged (flat) network. In this particular case, the instance resides on the same compute node as the DHCP agent for the network. If the DHCP agent resides on another compute node, the latter only contains a DHCP namespace with a port on the OVS integration bridge.



## Example configuration

Use the following example configuration as a template to add support for high-availability using DVR to an existing operational environment that supports self-service networks.

### Controller node

1. In the `neutron.conf` file:

- Enable distributed routing by default for all routers.

```
[DEFAULT]
router_distributed = True
```

2. Restart the following services:

- Server

### Network node

1. In the `openswitch_agent.ini` file, enable distributed routing.

```
[DEFAULT]
enable_distributed_routing = True
```

2. In the `l3_agent.ini` file, configure the layer-3 agent to provide SNAT services.

```
[DEFAULT]
agent_mode = dvr_snat
```

---

**Note:** The `external_network_bridge` option intentionally contains no value.

---

3. Restart the following services:

- Open vSwitch agent
- Layer-3 agent

### Compute nodes

1. Install the Networking service layer-3 agent.

2. In the `openswitch_agent.ini` file, enable distributed routing.

```
[DEFAULT]
enable_distributed_routing = True
```

3. In the `l3_agent.ini` file, configure the layer-3 agent.

```
[DEFAULT]
interface_driver = openvswitch
external_network_bridge =
agent_mode = dvr
```

---

**Note:** The external\_network\_bridge option intentionally contains no value.

---

4. Restart the following services:

- Open vSwitch agent
- Layer-3 agent

#### Verify service operation

1. Source the administrative project credentials.
2. Verify presence and operation of the agents.

```
$ openstack network agent list
+-----+-----+-----+
| ID | Agent Type | Host | Availability |
+-----+-----+-----+
05d980f2-a4fc-4815-91e7-a7f7e118c0db	L3 agent	compute1	nova
1236bbcb-e0ba-48a9-80fc-81202ca4fa51	Metadata agent	compute2	nova
2a2e9a90-51b8-4163-a7d6-3e199ba2374b	L3 agent	compute2	nova
457d6898-b373-4bb3-b41f-59345dcfb5c5	Open vSwitch agent	compute2	nova
513caa68-0391-4e53-a530-082e2c23e819	Linux bridge agent	compute1	nova
71f15e84-bc47-4c2a-b9fb-317840b2d753	DHCP agent	compute2	nova
8805b962-de95-4e40-bdc2-7a0add7521e8	L3 agent	network1	nova
a33cac5a-0266-48f6-9cac-4cef4f8b0358	Open vSwitch agent	network1	nova
a6c69690-e7f7-4e56-9831-1282753e5007	Metadata agent	compute1	nova
af11f22f-a9f4-404f-9fd8-cd7ad55c0f68	DHCP agent	compute1	nova
bcfc977b-ec0e-4ba9-be62-9489b4b0e6f1	Open vSwitch agent	compute1	nova
+-----+-----+-----+
```

## Create initial networks

Similar to the self-service deployment example, this configuration supports multiple VXLAN self-service networks. After enabling high-availability, all additional routers use distributed routing. The following procedure creates an additional self-service network and router. The Networking service also supports adding distributed routing to existing routers.

1. Source a regular (non-administrative) project credentials.
2. Create a self-service network.

```
$ openstack network create selfservice2
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	UP
mtu	1450
name	selfservice2
port_security_enabled	True
router:external	Internal
shared	False
status	ACTIVE
+-----+-----+
```

3. Create a IPv4 subnet on the self-service network.

```
$ openstack subnet create --subnet-range 192.0.2.0/24 \
 --network selfservice2 --dns-nameserver 8.8.4.4 selfservice2-v4
+-----+-----+
| Field | Value |
+-----+-----+
allocation_pools	192.0.2.2-192.0.2.254
cidr	192.0.2.0/24
dns_nameservers	8.8.4.4
enable_dhcp	True
gateway_ip	192.0.2.1
ip_version	4
name	selfservice2-v4
+-----+-----+
```

4. Create a IPv6 subnet on the self-service network.

```
$ openstack subnet create --subnet-range fd00:192:0:2::/64 --ip-version 6 \
 --ipv6-ra-mode slaac --ipv6-address-mode slaac --network selfservice2 \
 --dns-nameserver 2001:4860:4860::8844 selfservice2-v6
+-----+-----+
| Field | Value |
+-----+-----+
allocation_pools	fd00:192:0:2::2-fd00:192:0:2:ffff:ffff:ffff:ffff
cidr	fd00:192:0:2::/64
dns_nameservers	2001:4860:4860::8844
enable_dhcp	True
gateway_ip	fd00:192:0:2::1
ip_version	6
ipv6_address_mode	slaac
ipv6_ra_mode	slaac
name	selfservice2-v6
+-----+-----+
```

---

5. Create a router.

```
$ openstack router create router2
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	UP
name	router2
status	ACTIVE
+-----+-----+
```

6. Add the IPv4 and IPv6 subnets as interfaces on the router.

```
$ openstack router add subnet router2 selfservice2-v4
$ openstack router add subnet router2 selfservice2-v6
```

---

**Note:** These commands provide no output.

---

7. Add the provider network as a gateway on the router.

```
$ openstack router set router2 --external-gateway provider1
```

## Verify network operation

1. Source the administrative project credentials.
2. Verify distributed routing on the router.

```
$ openstack router show router2
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	UP
distributed	True
ha	False
name	router2
status	ACTIVE
+-----+-----+
```

3. On each compute node, verify creation of a qrouter namespace with the same ID.

Compute node 1:

```
ip netns
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

Compute node 2:

```
ip netns
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

- On the network node, verify creation of the snat and qrouter namespaces with the same ID.

```
ip netns
snat-78d2f628-137c-4f26-a257-25fc20f203c1
qrouter-78d2f628-137c-4f26-a257-25fc20f203c1
```

---

**Note:** The namespace for router 1 from [Open vSwitch: Self-service networks](#) should also appear on network node 1 because of creation prior to enabling distributed routing.

---

- Launch an instance with an interface on the additional self-service network. For example, a CirrOS image using flavor ID 1.

```
$ openstack server create --flavor 1 --image cirros --nic net-id=NETWORK_ID
→selfservice-instance2
```

Replace NETWORK\_ID with the ID of the additional self-service network.

- Determine the IPv4 and IPv6 addresses of the instance.

| \$ openstack server list             |                       |        |                                                          |
|--------------------------------------|-----------------------|--------|----------------------------------------------------------|
| ID                                   | Name                  | Status | Networks                                                 |
| bde64b00-77ae-41b9-b19a-cd8e378d9f8b | selfservice-instance2 | ACTIVE | selfservice2=fd00:192:0:2:f816:3eff:fe71:e93e, 192.0.2.4 |

- Create a floating IPv4 address on the provider network.

```
$ openstack floating ip create provider1
+-----+
| Field | Value |
+-----+
fixed_ip	None
id	0174056a-fa56-4403-b1ea-b5151a31191f
instance_id	None
ip	203.0.113.17
pool	provider1
+-----+
```

- Associate the floating IPv4 address with the instance.

```
$ openstack server add floating ip selfservice-instance2 203.0.113.17
```

---

**Note:** This command provides no output.

---

- On the compute node containing the instance, verify creation of the fip namespace with the same ID as the provider network.

```
ip netns
fip-4bfa3075-b4b2-4f7d-b88e-df1113942d43
```

## Network traffic flow

The following sections describe the flow of network traffic in several common scenarios. *North-south* network traffic travels between an instance and external network such as the Internet. *East-west* network traffic travels between instances on the same or different networks. In all scenarios, the physical network infrastructure handles switching and routing among provider networks and external networks such as the Internet. Each case references one or more of the following components:

- Provider network (VLAN)
  - VLAN ID 101 (tagged)
- Self-service network 1 (VXLAN)
  - VXLAN ID (VNI) 101
- Self-service network 2 (VXLAN)
  - VXLAN ID (VNI) 102
- Self-service router
  - Gateway on the provider network
  - Interface on self-service network 1
  - Interface on self-service network 2
- Instance 1
- Instance 2

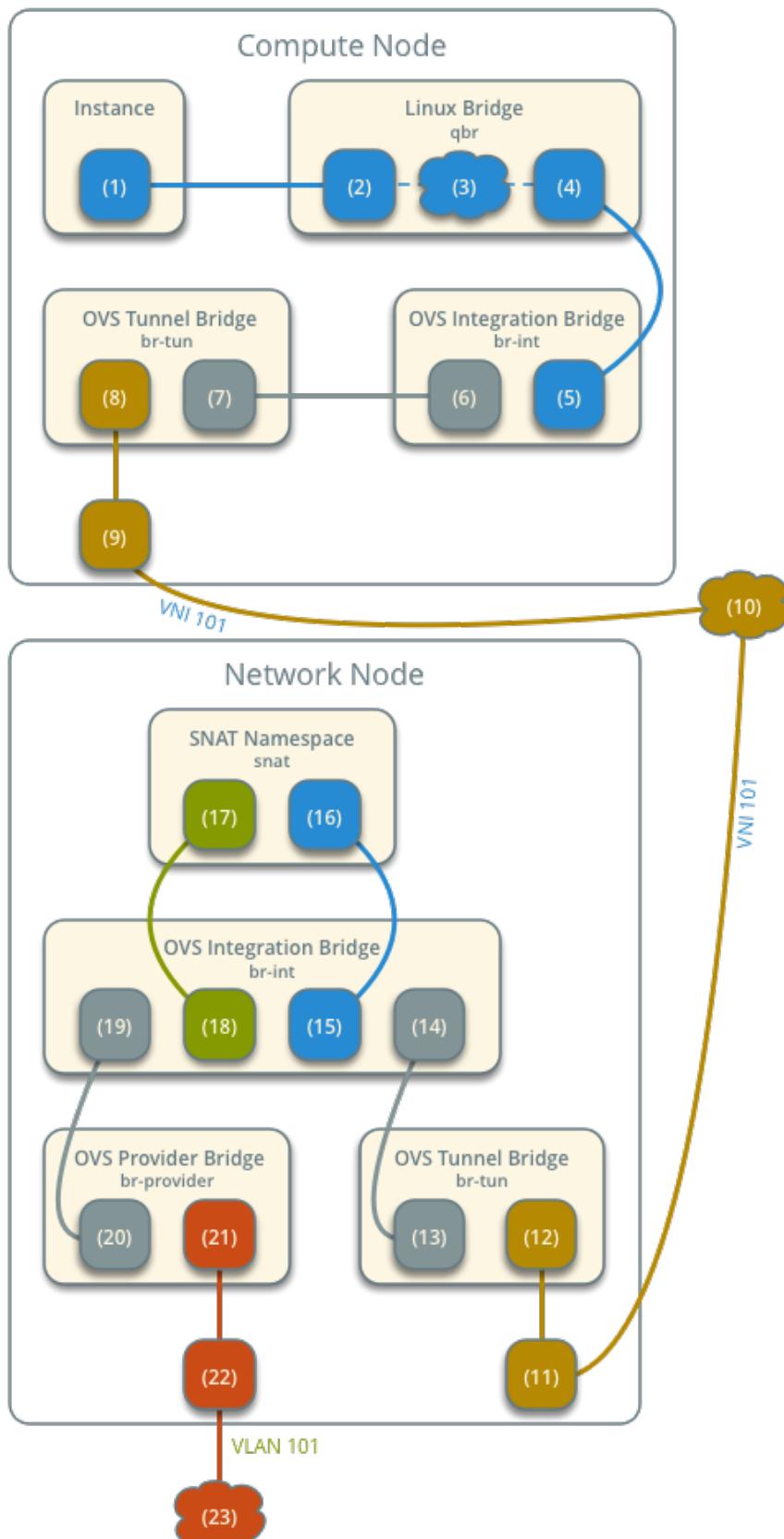
This section only contains flow scenarios that benefit from distributed virtual routing or that differ from conventional operation. For other flow scenarios, see [Network traffic flow](#).

### North-south scenario 1: Instance with a fixed IP address

Similar to [North-south scenario 1: Instance with a fixed IP address](#), except the router namespace on the network node becomes the SNAT namespace. The network node still contains the router namespace, but it serves no purpose in this case.

## Open vSwitch - High-availability with DVR

### Network Traffic Flow - North/South Scenario 1



### North-south scenario 2: Instance with a floating IPv4 address

For instances with a floating IPv4 address using a self-service network on a distributed router, the compute node containing the instance performs SNAT on north-south traffic passing from the instance to external networks such as the Internet and DNAT on north-south traffic passing from external networks to the instance. Floating IP addresses and NAT do not apply to IPv6. Thus, the network node routes IPv6 traffic in this scenario. north-south traffic passing between the instance and external networks such as the Internet.

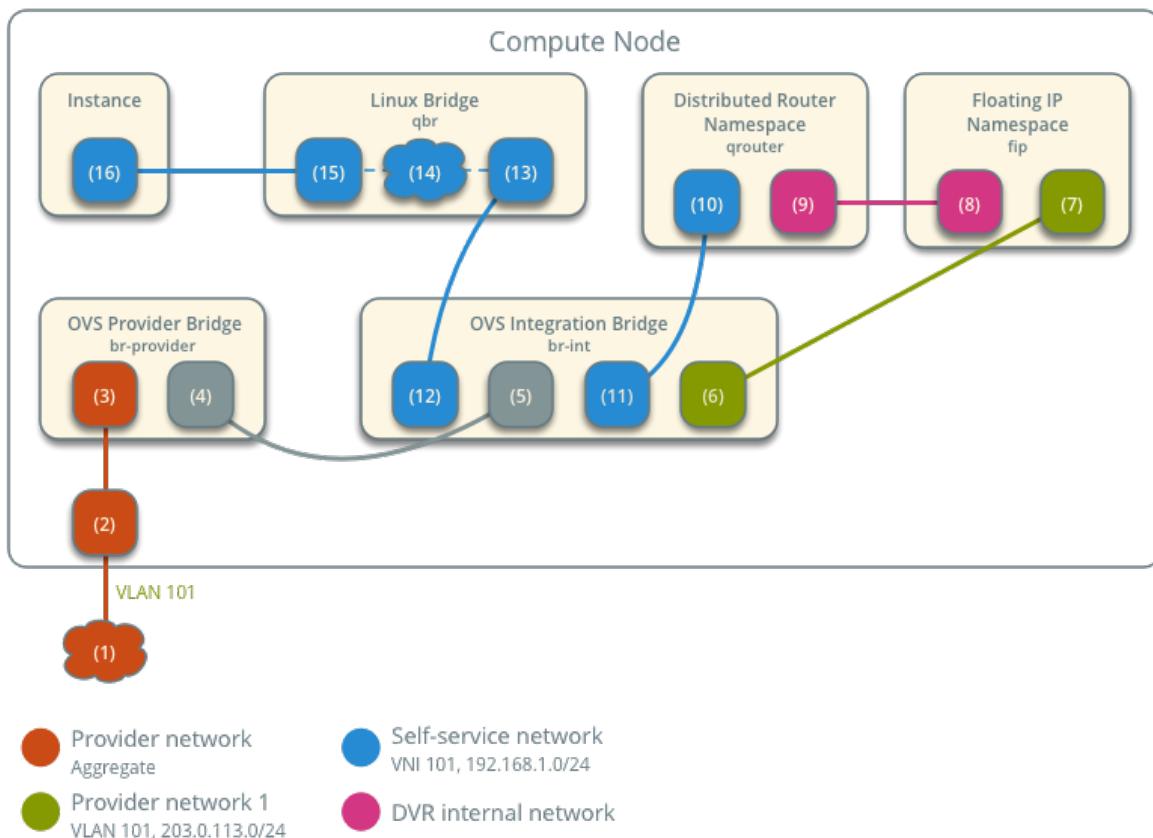
- Instance 1 resides on compute node 1 and uses self-service network 1.
- A host on the Internet sends a packet to the instance.

The following steps involve the compute node:

1. The physical network infrastructure (1) forwards the packet to the provider physical network interface (2).
2. The provider physical network interface forwards the packet to the OVS provider bridge provider network port (3).
3. The OVS provider bridge swaps actual VLAN tag 101 with the internal VLAN tag.
4. The OVS provider bridge `phy-br-provider` port (4) forwards the packet to the OVS integration bridge `int-br-provider` port (5).
5. The OVS integration bridge port for the provider network (6) removes the internal VLAN tag and forwards the packet to the provider network interface (7) in the floating IP namespace. This interface responds to any ARP requests for the instance floating IPv4 address.
6. The floating IP namespace routes the packet (8) to the distributed router namespace (9) using a pair of IP addresses on the DVR internal network. This namespace contains the instance floating IPv4 address.
7. The router performs DNAT on the packet which changes the destination IP address to the instance IP address on the self-service network via the self-service network interface (10).
8. The router forwards the packet to the OVS integration bridge port for the self-service network (11).
9. The OVS integration bridge adds an internal VLAN tag to the packet.
10. The OVS integration bridge removes the internal VLAN tag from the packet.
11. The OVS integration bridge security group port (12) forwards the packet to the security group bridge OVS port (13) via `veth` pair.
12. Security group rules (14) on the security group bridge handle firewalling and connection tracking for the packet.
13. The security group bridge instance port (15) forwards the packet to the instance interface (16) via `veth` pair.

## Open vSwitch - High-availability with DVR

### Network Traffic Flow - North/South Scenario 2




---

**Note:** Egress traffic follows similar steps in reverse, except SNAT changes the source IPv4 address of the packet to the floating IPv4 address.

---

#### East-west scenario 1: Instances on different networks on the same router

Instances with fixed IPv4/IPv6 address or floating IPv4 address on the same compute node communicate via router on the compute node. Instances on different compute nodes communicate via an instance of the router on each compute node.

---

**Note:** This scenario places the instances on different compute nodes to show the most complex situation.

---

The following steps involve compute node 1:

1. The instance interface (1) forwards the packet to the security group bridge instance port (2) via veth pair.
2. Security group rules (3) on the security group bridge handle firewalling and connection tracking for the packet.
3. The security group bridge OVS port (4) forwards the packet to the OVS integration bridge security group.

port (5) via veth pair.

4. The OVS integration bridge adds an internal VLAN tag to the packet.
5. The OVS integration bridge port for self-service network 1 (6) removes the internal VLAN tag and forwards the packet to the self-service network 1 interface in the distributed router namespace (6).
6. The distributed router namespace routes the packet to self-service network 2.
7. The self-service network 2 interface in the distributed router namespace (8) forwards the packet to the OVS integration bridge port for self-service network 2 (9).
8. The OVS integration bridge adds an internal VLAN tag to the packet.
9. The OVS integration bridge exchanges the internal VLAN tag for an internal tunnel ID.
10. The OVS integration bridge patch-tun port (10) forwards the packet to the OVS tunnel bridge patch-int port (11).
11. The OVS tunnel bridge (12) wraps the packet using VNI 101.
12. The underlying physical interface (13) for overlay networks forwards the packet to compute node 2 via the overlay network (14).

The following steps involve compute node 2:

1. The underlying physical interface (15) for overlay networks forwards the packet to the OVS tunnel bridge (16).
2. The OVS tunnel bridge unwraps the packet and adds an internal tunnel ID to it.
3. The OVS tunnel bridge exchanges the internal tunnel ID for an internal VLAN tag.
4. The OVS tunnel bridge patch-int patch port (17) forwards the packet to the OVS integration bridge patch-tun patch port (18).
5. The OVS integration bridge removes the internal VLAN tag from the packet.
6. The OVS integration bridge security group port (19) forwards the packet to the security group bridge OVS port (20) via veth pair.
7. Security group rules (21) on the security group bridge handle firewalls and connection tracking for the packet.
8. The security group bridge instance port (22) forwards the packet to the instance 2 interface (23) via veth pair.

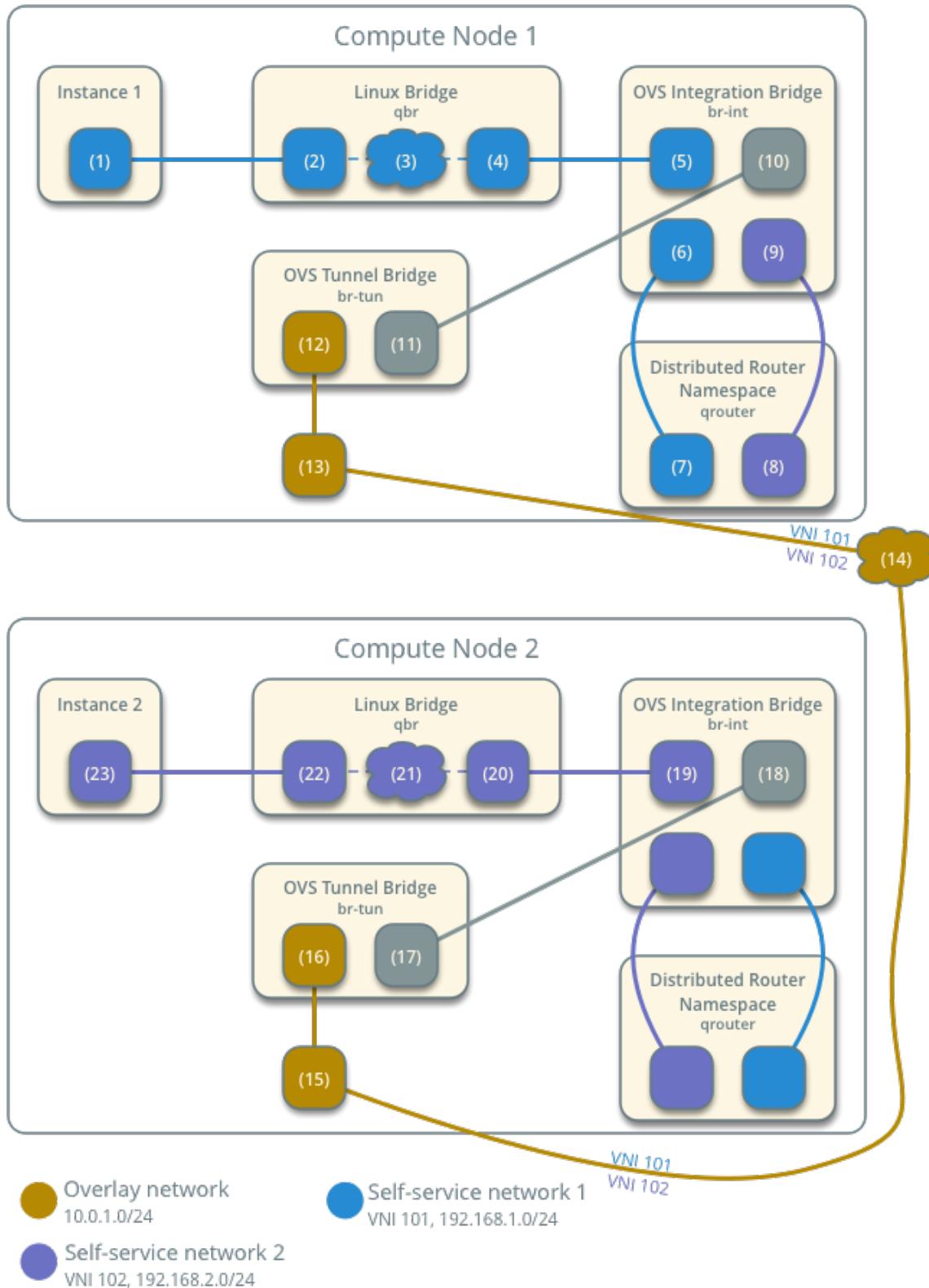
---

**Note:** Routing between self-service networks occurs on the compute node containing the instance sending the packet. In this scenario, routing occurs on compute node 1 for packets from instance 1 to instance 2 and on compute node 2 for packets from instance 2 to instance 1.

---

## Open vSwitch - High-availability with DVR

### Network Traffic Flow - East/West Scenario 1



## Operations

### IP availability metrics

Network IP Availability is an information-only API extension that allows a user or process to determine the number of IP addresses that are consumed across networks and the allocation pools of their subnets. This extension was added to neutron in the Mitaka release.

This section illustrates how you can get the Network IP address availability through the command-line interface.

Get Network IP address availability for all IPv4 networks:

```
$ openstack ip availability list
```

| Network ID                           | Network Name | Total IPs | Used IPs |
|--------------------------------------|--------------|-----------|----------|
| 363a611a-b08b-4281-b64e-198d90cb94fd | private      | 253       | 3        |
| c92d0605-caf2-4349-b1b8-8d5f9ac91df8 | public       | 253       | 1        |

Get Network IP address availability for all IPv6 networks:

```
$ openstack ip availability list --ip-version 6
```

| Network ID                           | Network Name | Total IPs            | Used IPs |
|--------------------------------------|--------------|----------------------|----------|
| 363a611a-b08b-4281-b64e-198d90cb94fd | private      | 18446744073709551614 | 3        |
| c92d0605-caf2-4349-b1b8-8d5f9ac91df8 | public       | 18446744073709551614 | 1        |

Get Network IP address availability statistics for a specific network:

```
$ openstack ip availability show NETWORKUUID
```

| Field                  | Value                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| network_id             | 0bf90de6-fc0f-4dba-b80d-96670dfb331a                                                                                                               |
| network_name           | public                                                                                                                                             |
| project_id             | 5669caad86a04256994cdf755df4d3c1                                                                                                                   |
| subnet_ip_availability | cidr='192.0.2.224/28', ip_version='4', subnet_id='346806ee-a53e-44fd-968a-ddb2bcd2ba96', subnet_name='public_subnet', total_ips='13', used_ips='5' |
| total_ips              | 13                                                                                                                                                 |
| used_ips               | 5                                                                                                                                                  |

### Resource tags

Various virtual networking resources support tags for use by external systems or any other clients of the Networking service API.

The currently supported resources are:

- networks
- subnets
- subnetpools
- ports
- routers

## Use cases

The following use cases refer to adding tags to networks, but the same can be applicable to any other supported Networking service resource:

1. Ability to map different networks in different OpenStack locations to one logically same network (for multi-site OpenStack).
2. Ability to map IDs from different management/orchestration systems to OpenStack networks in mixed environments. For example, in the Kuryr project, the Docker network ID is mapped to the Neutron network ID.
3. Ability to leverage tags by deployment tools.
4. Ability to tag information about provider networks (for example, high-bandwidth, low-latency, and so on).

## Filtering with tags

The API allows searching/filtering of the `GET /v2.0/networks` API. The following query parameters are supported:

- tags
- tags-any
- not-tags
- not-tags-any

To request the list of networks that have a single tag, `tags` argument should be set to the desired tag name. Example:

```
GET /v2.0/networks?tags=red
```

To request the list of networks that have two or more tags, the `tags` argument should be set to the list of tags, separated by commas. In this case, the tags given must all be present for a network to be included in the query result. Example that returns networks that have the “red” and “blue” tags:

```
GET /v2.0/networks?tags=red,blue
```

To request the list of networks that have one or more of a list of given tags, the `tags-any` argument should be set to the list of tags, separated by commas. In this case, as long as one of the given tags is present, the network will be included in the query result. Example that returns the networks that have the “red” or the “blue” tag:

```
GET /v2.0/networks?tags-any=red,blue
```

To request the list of networks that do not have one or more tags, the `not-tags` argument should be set to the list of tags, separated by commas. In this case, only the networks that do not have any of the given tags will be included in the query results. Example that returns the networks that do not have either “red” or “blue” tag:

```
GET /v2.0/networks?not-tags=red,blue
```

To request the list of networks that do not have at least one of a list of tags, the `not-tags-any` argument should be set to the list of tags, separated by commas. In this case, only the networks that do not have at least one of the given tags will be included in the query result. Example that returns the networks that do not have the “red” tag, or do not have the “blue” tag:

```
GET /v2.0/networks?not-tags-any=red,blue
```

The `tags`, `tags-any`, `not-tags`, and `not-tags-any` arguments can be combined to build more complex queries. Example:

```
GET /v2.0/networks?tags=red,blue&tags-any=green,orange
```

The above example returns any networks that have the “red” and “blue” tags, plus at least one of “green” and “orange”.

Complex queries may have contradictory parameters. Example:

```
GET /v2.0/networks?tags=blue¬-tags=blue
```

In this case, we should let the Networking service find these networks. Obviously, there are no such networks and the service will return an empty list.

## User workflow

Add a tag to a resource:

```
$ neutron tag-add --resource-type network --resource ab442634-1cc9-49e5-bd49-0dac9c811f69 --
→tag red
$ neutron net-show net
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	True
availability_zone_hints	
availability_zones	
id	ab442634-1cc9-49e5-bd49-0dac9c811f69
ipv4_address_scope	
ipv6_address_scope	
mtu	1450
name	net
port_security_enabled	True
router:external	False
shared	False
status	ACTIVE
subnets	
tags	red
tenant_id	e6710680bfd14555891f265644e1dd5c
+-----+-----+
```

Remove a tag from a resource:

```
$ neutron tag-remove --resource-type network --resource ab442634-1cc9-49e5-bd49-
0dac9c811f69 --tag red
$ neutron net-show net
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	True
availability_zone_hints	
availability_zones	
id	ab442634-1cc9-49e5-bd49-0dac9c811f69
ipv4_address_scope	
ipv6_address_scope	
mtu	1450
name	net
port_security_enabled	True
router:external	False
shared	False
status	ACTIVE
subnets	
tags	
tenant_id	e6710680bfd14555891f265644e1dd5c
+-----+-----+
```

Replace all tags on the resource:

```
$ neutron tag-replace --resource-type network --resource ab442634-1cc9-49e5-bd49-
0dac9c811f69 --tag red --tag blue
$ neutron net-show net
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	True
availability_zone_hints	
availability_zones	
id	ab442634-1cc9-49e5-bd49-0dac9c811f69
ipv4_address_scope	
ipv6_address_scope	
mtu	1450
name	net
port_security_enabled	True
router:external	False
shared	False
status	ACTIVE
subnets	
tags	red
	blue
tenant_id	e6710680bfd14555891f265644e1dd5c
+-----+-----+
```

Clear tags from a resource:

```
$ neutron tag-remove --resource-type network --resource ab442634-1cc9-49e5-bd49-
0dac9c811f69 --all
$ neutron net-show net
+-----+-----+
| Field | Value |
```

| admin_state_up          | True                                 |  |
|-------------------------|--------------------------------------|--|
| availability_zone_hints |                                      |  |
| availability_zones      |                                      |  |
| id                      | ab442634-1cc9-49e5-bd49-0dac9c811f69 |  |
| ipv4_address_scope      |                                      |  |
| ipv6_address_scope      |                                      |  |
| mtu                     | 1450                                 |  |
| name                    | net                                  |  |
| port_security_enabled   | True                                 |  |
| router:external         | False                                |  |
| shared                  | False                                |  |
| status                  | ACTIVE                               |  |
| subnets                 |                                      |  |
| tags                    |                                      |  |
| tenant_id               | e6710680bfd14555891f265644e1dd5c     |  |

Get list of resources with tag filters from networks. The networks are: test-net1 with “red” tag, test-net2 with “red” and “blue” tags, test-net3 with “red”, “blue”, and “green” tags, and test-net4 with “green” tag.

Get list of resources with tags filter:

| \$ neutron net-list --tags red,blue  |                     |         |
|--------------------------------------|---------------------|---------|
|                                      |                     |         |
| id                                   | name                | subnets |
| +-----+-----+-----+                  | +-----+-----+-----+ | +-----+ |
| 8ca3b9ed-f578-45fa-8c44-c53f13aec05a | test-net3           |         |
| e736e63d-42e4-4f4c-836c-6ad286ffd68a | test-net2           |         |
| +-----+-----+-----+                  | +-----+-----+-----+ | +-----+ |

Get list of resources with tags-any filter:

| \$ neutron net-list --tags-any red,blue |                     |         |
|-----------------------------------------|---------------------|---------|
|                                         |                     |         |
| id                                      | name                | subnets |
| +-----+-----+-----+                     | +-----+-----+-----+ | +-----+ |
| 30491224-3855-431f-a688-fb29df004d82    | test-net1           |         |
| 8ca3b9ed-f578-45fa-8c44-c53f13aec05a    | test-net3           |         |
| e736e63d-42e4-4f4c-836c-6ad286ffd68a    | test-net2           |         |
| +-----+-----+-----+                     | +-----+-----+-----+ | +-----+ |

Get list of resources with not-tags filter:

| \$ neutron net-list --not-tags red,blue |                     |         |
|-----------------------------------------|---------------------|---------|
|                                         |                     |         |
| id                                      | name                | subnets |
| +-----+-----+-----+                     | +-----+-----+-----+ | +-----+ |
| 30491224-3855-431f-a688-fb29df004d82    | test-net1           |         |
| cdb3ed08-ca63-4090-ba12-30b366372993    | test-net4           |         |
| +-----+-----+-----+                     | +-----+-----+-----+ | +-----+ |

Get list of resources with not-tags-any filter:

| \$ neutron net-list --not-tags-any red,blue |                     |         |
|---------------------------------------------|---------------------|---------|
|                                             |                     |         |
| id                                          | name                | subnets |
| +-----+-----+-----+                         | +-----+-----+-----+ | +-----+ |

|                                                  |  |  |
|--------------------------------------------------|--|--|
| +-----+-----+-----+                              |  |  |
| cdb3ed08-ca63-4090-ba12-30b366372993   test-net4 |  |  |
| +-----+-----+-----+                              |  |  |

## Limitations

Filtering resources with a tag whose name contains a comma is not supported. Thus, do not put such a tag name to resources.

## Future support

In future releases, the Networking service may support setting tags for additional resources.

## Resource purge

The Networking service provides a purge mechanism to delete the following network resources for a project:

- Networks
- Subnets
- Ports
- Router interfaces
- Routers
- Floating IP addresses
- Security groups

Typically, one uses this mechanism to delete networking resources for a defunct project regardless of its existence in the Identity service.

## Usage

1. Source the necessary project credentials. The administrative project can delete resources for all other projects. A regular project can delete its own network resources and those belonging to other projects for which it has sufficient access.
2. Delete the network resources for a particular project.

```
$ neutron purge PROJECT_ID
```

Replace `PROJECT_ID` with the project ID.

The command provides output that includes a completion percentage and the quantity of successful or unsuccessful network resource deletions. An unsuccessful deletion usually indicates sharing of a resource with one or more additional projects.

```
Purging resources: 100% complete.
Deleted 1 security_group, 2 ports, 1 router, 1 floatingip, 2 networks.
The following resources could not be deleted: 1 network.
```

The command also indicates if a project lacks network resources.

```
Tenant has no supported resources.
```

## Migration

### Database

The upgrade of the Networking service database is implemented with Alembic migration chains. The migrations in the alembic/versions contain the changes needed to migrate from older Networking service releases to newer ones.

Since Liberty, Networking maintains two parallel Alembic migration branches.

The first branch is called expand and is used to store expansion-only migration rules. These rules are strictly additive and can be applied while the Neutron server is running.

The second branch is called contract and is used to store those migration rules that are not safe to apply while Neutron server is running.

The intent of separate branches is to allow invoking those safe migrations from the expand branch while the Neutron server is running and therefore reducing downtime needed to upgrade the service.

A database management command-line tool uses the Alembic library to manage the migration.

### Database management command-line tool

The database management command-line tool is called **neutron-db-manage**. Pass the `--help` option to the tool for usage information.

The tool takes some options followed by some commands:

```
$ neutron-db-manage <options> <commands>
```

The tool needs to access the database connection string, which is provided in the `neutron.conf` configuration file in an installation. The tool automatically reads from `/etc/neutron/neutron.conf` if it is present. If the configuration is in a different location, use the following command:

```
$ neutron-db-manage --config-file /path/to/neutron.conf <commands>
```

Multiple `--config-file` options can be passed if needed.

Instead of reading the DB connection from the configuration file(s), you can use the `--database-connection` option:

```
$ neutron-db-manage --database-connection
mysql+pymysql://root:secret@127.0.0.1/neutron?charset=utf8 <commands>
```

The `branches`, `current`, and `history` commands all accept a `--verbose` option, which, when passed, will instruct **neutron-db-manage** to display more verbose output for the specified command:

```
$ neutron-db-manage current --verbose
```

---

**Note:** The tool usage examples below do not show the options. It is assumed that you use the options that you need for your environment.

---

In new deployments, you start with an empty database and then upgrade to the latest database version using the following command:

```
$ neutron-db-manage upgrade heads
```

After installing a new version of the Neutron server, upgrade the database using the following command:

```
$ neutron-db-manage upgrade heads
```

In existing deployments, check the current database version using the following command:

```
$ neutron-db-manage current
```

To apply the expansion migration rules, use the following command:

```
$ neutron-db-manage upgrade --expand
```

To apply the non-expansive migration rules, use the following command:

```
$ neutron-db-manage upgrade --contract
```

To check if any contract migrations are pending and therefore if offline migration is required, use the following command:

```
$ neutron-db-manage has_offline_migrations
```

---

**Note:** Offline migration requires all Neutron server instances in the cluster to be shutdown before you apply any contract scripts.

---

To generate a script of the command instead of operating immediately on the database, use the following command:

```
$ neutron-db-manage upgrade heads --sql
```

.. note::

The `--sql` option causes the command to generate a script. The script can be run later (online or offline), perhaps after verifying and/or modifying it.

To migrate between specific migration versions, use the following command:

```
$ neutron-db-manage upgrade <start version>:<end version>
```

To upgrade the database incrementally, use the following command:

```
$ neutron-db-manage upgrade --delta <# of revs>
```

---

**Note:** Database downgrade is not supported.

---

To look for differences between the schema generated by the upgrade command and the schema defined by the models, use the **revision --autogenerate** command:

```
neutron-db-manage revision -m REVISION_DESCRIPTION --autogenerate
```

---

**Note:** This generates a prepopulated template with the changes needed to match the database state with the models.

---

## Legacy nova-network to OpenStack Networking (neutron)

Two networking models exist in OpenStack. The first is called legacy networking (*nova-network*) and it is a sub-process embedded in the Compute project (nova). This model has some limitations, such as creating complex network topologies, extending its back-end implementation to vendor-specific technologies, and providing project-specific networking elements. These limitations are the main reasons the OpenStack Networking (neutron) model was created.

This section describes the process of migrating clouds based on the legacy networking model to the OpenStack Networking model. This process requires additional changes to both compute and networking to support the migration. This document describes the overall process and the features required in both Networking and Compute.

The current process as designed is a minimally viable migration with the goal of deprecating and then removing legacy networking. Both the Compute and Networking teams agree that a one-button migration process from legacy networking to OpenStack Networking (neutron) is not an essential requirement for the deprecation and removal of the legacy networking at a future date. This section includes a process and tools which are designed to solve a simple use case migration.

Users are encouraged to take these tools, test them, provide feedback, and then expand on the feature set to suit their own deployments; deployers that refrain from participating in this process intending to wait for a path that better suits their use case are likely to be disappointed.

### Impact and limitations

The migration process from the legacy nova-network networking service to OpenStack Networking (neutron) has some limitations and impacts on the operational state of the cloud. It is critical to understand them in order to decide whether or not this process is acceptable for your cloud and all users.

### Management impact

The Networking REST API is publicly read-only until after the migration is complete. During the migration, Networking REST API is read-write only to nova-api, and changes to Networking are only allowed via nova-api.

The Compute REST API is available throughout the entire process, although there is a brief period where it is made read-only during a database migration. The Networking REST API will need to expose (to nova-api) all details necessary for reconstructing the information previously held in the legacy networking database.

Compute needs a per-hypervisor “has\_transitioned” boolean change in the data model to be used during the migration process. This flag is no longer required once the process is complete.

### Operations impact

In order to support a wide range of deployment options, the migration process described here requires a rolling restart of hypervisors. The rate and timing of specific hypervisor restarts is under the control of the operator.

The migration may be paused, even for an extended period of time (for example, while testing or investigating issues) with some hypervisors on legacy networking and some on Networking, and Compute API remains fully functional. Individual hypervisors may be rolled back to legacy networking during this stage of the migration, although this requires an additional restart.

In order to support the widest range of deployer needs, the process described here is easy to automate but is not already automated. Deployers should expect to perform multiple manual steps or write some simple scripts in order to perform this migration.

### Performance impact

During the migration, nova-network API calls will go through an additional internal conversion to Networking calls. This will have different and likely poorer performance characteristics compared with either the pre-migration or post-migration APIs.

### Migration process overview

1. Start neutron-server in intended final config, except with REST API restricted to read-write only by nova-api.
2. Make the Compute REST API read-only.
3. Run a DB dump/restore tool that creates Networking data structures representing current legacy networking config.
4. Enable a nova-api proxy that recreates internal Compute objects from Networking information (via the Networking REST API).
5. Make Compute REST API read-write again. This means legacy networking DB is now unused, new changes are now stored in the Networking DB, and no rollback is possible from here without losing those new changes.

---

**Note:** At this moment the Networking DB is the source of truth, but nova-api is the only public read-write API.

---

Next, you’ll need to migrate each hypervisor. To do that, follow these steps:

1. Disable the hypervisor. This would be a good time to live migrate or evacuate the compute node, if supported.
2. Disable nova-compute.
3. Enable the Networking agent.
4. Set the “has\_transitioned” flag in the Compute hypervisor database/config.

5. Reboot the hypervisor (or run “smart” live transition tool if available).
6. Re-enable the hypervisor.

At this point, all compute nodes have been migrated, but they are still using the nova-api API and Compute gateways. Finally, enable OpenStack Networking by following these steps:

1. Bring up the Networking (l3) nodes. The new routers will have identical MAC+IPs as old Compute gateways so some sort of immediate cutover is possible, except for stateful connections issues such as NAT.
2. Make the Networking API read-write and disable legacy networking.

Migration Completed!

### Add VRRP to an existing router

This section describes the process of migrating from a classic router to an L3 HA router, which is available starting from the Mitaka release.

Similar to the classic scenario, all network traffic on a project network that requires routing actively traverses only one network node regardless of the quantity of network nodes providing HA for the router. Therefore, this high-availability implementation primarily addresses failure situations instead of bandwidth constraints that limit performance. However, it supports random distribution of routers on different network nodes to reduce the chances of bandwidth constraints and to improve scaling.

This section references parts of [Linux bridge: High availability using VRRP](#) and [Open vSwitch: High availability using VRRP](#). For details regarding needed infrastructure and configuration to allow actual L3 HA deployment, read the relevant guide before continuing with the migration process.

### Migration

The migration process is quite simple, it involves turning down the router by setting the router’s `admin_state_up` attribute to `False`, upgrading the router to L3 HA and then setting the router’s `admin_state_up` attribute back to `True`.

**Warning:** Once starting the migration, south-north connections (instances to internet) will be severed. New connections will be able to start only when the migration is complete.

Here is the router we have used in our demonstration:

```
$ openstack router show router1
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	UP
distributed	False
external_gateway_info	
ha	False
id	6b793b46-d082-4fd5-980f-a6f80cbb0f2a
name	router1
project_id	bb8b84ab75be4e19bd0dfe02f6c3f5c1
routes	
```

|         |         |         |
|---------|---------|---------|
| status  | ACTIVE  |         |
| +-----+ | +-----+ | +-----+ |

1. Source the administrative project credentials.
2. Set the admin\_state\_up to False. This will sever south-north connections until admin\_state\_up is set to True again.

```
$ openstack router set router1 --disable
```

3. Set the ha attribute of the router to True.

```
$ openstack router set router1 --ha
```

4. Set the admin\_state\_up to True. After this, south-north connections can start.

```
$ openstack router set router1 --enable
```

5. Make sure that the router's ha attribute has changed to True.

| \$ openstack router show router1 |                                      |  |
|----------------------------------|--------------------------------------|--|
| Field                            | Value                                |  |
| admin_state_up                   | UP                                   |  |
| distributed                      | False                                |  |
| external_gateway_info            |                                      |  |
| ha                               | True                                 |  |
| id                               | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a |  |
| name                             | router1                              |  |
| project_id                       | bb8b84ab75be4e19bd0dfe02f6c3f5c1     |  |
| routes                           |                                      |  |
| status                           | ACTIVE                               |  |

### L3 HA to Legacy

To return to classic mode, turn down the router again, turning off L3 HA and starting the router again.

**Warning:** Once starting the migration, south-north connections (instances to internet) will be severed. New connections will be able to start only when the migration is complete.

Here is the router we have used in our demonstration:

| \$ openstack router show router1 |                                      |  |
|----------------------------------|--------------------------------------|--|
| Field                            | Value                                |  |
| admin_state_up                   | DOWN                                 |  |
| distributed                      | False                                |  |
| external_gateway_info            |                                      |  |
| ha                               | True                                 |  |
| id                               | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a |  |

|                     |                                  |  |
|---------------------|----------------------------------|--|
| name                | router1                          |  |
| project_id          | bb8b84ab75be4e19bd0dfe02f6c3f5c1 |  |
| routes              |                                  |  |
| status              | ACTIVE                           |  |
| +-----+-----+-----+ |                                  |  |

1. Source the administrative project credentials.
2. Set the admin\_state\_up to False. This will sever south-north connections until admin\_state\_up is set to True again.

```
$ openstack router set router1 --disable
```

3. Set the ha attribute of the router to True.

```
$ openstack router set router1 --no-ha
```

4. Set the admin\_state\_up to True. After this, south-north connections can start.

```
$ openstack router set router1 --enable
```

5. Make sure that the router's ha attribute has changed to False.

| \$ openstack router show router1 |                                      |  |
|----------------------------------|--------------------------------------|--|
| Field                            | Value                                |  |
| admin_state_up                   | UP                                   |  |
| distributed                      | False                                |  |
| external_gateway_info            |                                      |  |
| ha                               | False                                |  |
| id                               | 6b793b46-d082-4fd5-980f-a6f80cbb0f2a |  |
| name                             | router1                              |  |
| project_id                       | bb8b84ab75be4e19bd0dfe02f6c3f5c1     |  |
| routes                           |                                      |  |
| status                           | ACTIVE                               |  |
| +-----+-----+-----+              |                                      |  |

## Miscellaneous

### Firewall-as-a-Service (FWaaS) v2 scenario

#### Enable FWaaS v2

1. Enable the FWaaS plug-in in the /etc/neutron/neutron.conf file:

```
service_plugins = firewall_v2

[service_providers]
...
service_provider = FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
→OVSHybridIptablesFirewallDriver:default

[fwaas]
```

```
agent_version = v2
driver = neutron_fwaas.services.firewall.drivers.linux.iptables_fwaas_v2.
↪IptablesFwaasDriver
enabled = True
```

---

**Note:** On Ubuntu, modify the [fwaas] section in the /etc/neutron/fwaas\_driver.ini file instead of /etc/neutron/neutron.conf.

---

2. Configure the FWaaS plugin for the L3 agent.

In the AGENT section of l3\_agent.ini, make sure the FWaaS extension is loaded:

```
[AGENT]
extensions = fwaas
```

3. Create the required tables in the database:

```
neutron-db-manage --subproject neutron-fwaas upgrade head
```

4. Restart the neutron-l3-agent and neutron-server services to apply the settings.

---

**Note:** Firewall v2 is not supported by horizon yet.

---

## Configure Firewall-as-a-Service v2

Create the firewall rules and create a policy that contains them. Then, create a firewall that applies the policy.

1. Create a firewall rule:

```
$ neutron firewall-rule-create --protocol {tcp,udp,icmp,any} \
--source-ip-address SOURCE_IP_ADDRESS \
--destination-ip-address DESTINATION_IP_ADDRESS \
--source-port SOURCE_PORT_RANGE --destination-port DEST_PORT_RANGE \
--action {allow,deny,reject}
```

The Networking client requires a protocol value. If the rule is protocol agnostic, you can use the any value.

---

**Note:** When the source or destination IP address are not of the same IP version (for example, IPv6), the command returns an error.

---

2. Create a firewall policy:

```
$ neutron firewall-policy-create --firewall-rules \
"FIREFWALL_RULE_IDS_OR_NAMES" myfirewallpolicy
```

Separate firewall rule IDs or names with spaces. The order in which you specify the rules is important.

You can create a firewall policy without any rules and add rules later, as follows:

- To add multiple rules, use the update operation.

- To add a single rule, use the insert-rule operation.

For more details, see [Networking command-line client](#) in the OpenStack Command-Line Interface Reference.

---

**Note:** FWaaS always adds a default deny all rule at the lowest precedence of each policy. Consequently, a firewall policy with no rules blocks all traffic by default.

---

3. Create a firewall:

```
$ neutron firewall-create FIREWALL_POLICY_UUID
```

---

**Note:** The firewall remains in PENDING\_CREATE state until you create a Networking router and attach an interface to it.

---

## Firewall-as-a-Service (FWaaS) v1 scenario

### Enable FWaaS v1

FWaaS management options are also available in the Dashboard.

1. Enable the FWaaS plug-in in the /etc/neutron/neutron.conf file:

```
service_plugins = firewall

[service_providers]
...
service_provider = FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
 ↪OVSHybridIptablesFirewallDriver:default

[fwaas]
driver = neutron_fwaas.services.firewall.drivers.linux.iptables_fwaas.
 ↪IptablesFwaasDriver
enabled = True
```

---

**Note:** On Ubuntu, modify the [fwaas] section in the /etc/neutron/fwaas\_driver.ini file instead of /etc/neutron/neutron.conf.

---

2. Configure the FWaaS plugin for the L3 agent.

In the AGENT section of l3\_agent.ini, make sure the FWaaS extension is loaded:

```
[AGENT]
extensions = fwaas
```

Edit the FWaaS section in the /etc/neutron/neutron.conf file to indicate the agent version and driver:

```
[fwaas]
agent_version = v1
```

```
driver = iptables
enabled = True
conntrack_driver = conntrack
```

3. Create the required tables in the database:

```
neutron-db-manage --subproject neutron-fwaas upgrade head
```

4. Enable the option in the local\_settings.py file, which is typically located on the controller node:

```
OPENSTACK_NEUTRON_NETWORK = {
 # ...
 'enable_firewall' = True,
 # ...
}
```

---

**Note:** By default, enable\_firewall option value is True in local\_settings.py file.

---

Apply the settings by restarting the web server.

5. Restart the neutron-l3-agent and neutron-server services to apply the settings.

## Configure Firewall-as-a-Service v1

Create the firewall rules and create a policy that contains them. Then, create a firewall that applies the policy.

1. Create a firewall rule:

```
$ neutron firewall-rule-create --protocol {tcp,udp,icmp,any} \
 --source-ip-address SOURCE_IP_ADDRESS \
 --destination-ip-address DESTINATION_IP_ADDRESS \
 --source-port SOURCE_PORT_RANGE --destination-port DEST_PORT_RANGE \
 --action {allow,deny,reject}
```

The Networking client requires a protocol value. If the rule is protocol agnostic, you can use the any value.

---

**Note:** When the source or destination IP address are not of the same IP version (for example, IPv6), the command returns an error.

---

2. Create a firewall policy:

```
$ neutron firewall-policy-create --firewall-rules \
 "FIREWALL_RULE_IDS_OR_NAMES" myfirewallpolicy
```

Separate firewall rule IDs or names with spaces. The order in which you specify the rules is important.

You can create a firewall policy without any rules and add rules later, as follows:

- To add multiple rules, use the update operation.
- To add a single rule, use the insert-rule operation.

For more details, see [Networking command-line client](#) in the OpenStack Command-Line Interface Reference.

---

**Note:** FWaaS always adds a default deny all rule at the lowest precedence of each policy. Consequently, a firewall policy with no rules blocks all traffic by default.

---

### 3. Create a firewall:

```
$ neutron firewall-create FIREWALL_POLICY_UUID
```

---

**Note:** The firewall remains in PENDING\_CREATE state until you create a Networking router and attach an interface to it.

---

## Disable libvirt networking

Most OpenStack deployments use the [libvirt](#) toolkit for interacting with the hypervisor. Specifically, OpenStack Compute uses libvirt for tasks such as booting and terminating virtual machine instances. When OpenStack Compute boots a new instance, libvirt provides OpenStack with the VIF associated with the instance, and OpenStack Compute plugs the VIF into a virtual device provided by OpenStack Network. The libvirt toolkit itself does not provide any networking functionality in OpenStack deployments.

However, libvirt is capable of providing networking services to the virtual machines that it manages. In particular, libvirt can be configured to provide networking functionality akin to a simplified, single-node version of OpenStack. Users can use libvirt to create layer 2 networks that are similar to OpenStack Networking's networks, confined to a single node.

## libvirt network implementation

By default, libvirt's networking functionality is enabled, and libvirt creates a network when the system boots. To implement this network, libvirt leverages some of the same technologies that OpenStack Network does. In particular, libvirt uses:

- Linux bridging for implementing a layer 2 network
- dnsmasq for providing IP addresses to virtual machines using DHCP
- iptables to implement SNAT so instances can connect out to the public internet, and to ensure that virtual machines are permitted to communicate with dnsmasq using DHCP

By default, libvirt creates a network named *default*. The details of this network may vary by distribution; on Ubuntu this network involves:

- a Linux bridge named `virbr0` with an IP address of `192.0.2.1/24`
- a dnsmasq process that listens on the `virbr0` interface and hands out IP addresses in the range `192.0.2.2-192.0.2.254`
- a set of iptables rules

When libvirt boots a virtual machine, it places the machine's VIF in the bridge `virbr0` unless explicitly told not to.

On Ubuntu, the iptables ruleset that libvirt creates includes the following rules:

```
*nat
-A POSTROUTING -s 192.0.2.0/24 -d 224.0.0.0/24 -j RETURN
-A POSTROUTING -s 192.0.2.0/24 -d 255.255.255.255/32 -j RETURN
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -p tcp -j MASQUERADE --to-ports 1024-65535
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -p udp -j MASQUERADE --to-ports 1024-65535
-A POSTROUTING -s 192.0.2.0/24 ! -d 192.0.2.0/24 -j MASQUERADE
*mangle
-A POSTROUTING -o virbr0 -p udp -m udp --dport 68 -j CHECKSUM --checksum-fill
*filter
-A INPUT -i virbr0 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i virbr0 -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -i virbr0 -p udp -m udp --dport 67 -j ACCEPT
-A INPUT -i virbr0 -p tcp -m tcp --dport 67 -j ACCEPT
-A FORWARD -d 192.0.2.0/24 -o virbr0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -s 192.0.2.0/24 -i virbr0 -j ACCEPT
-A FORWARD -i virbr0 -o virbr0 -j ACCEPT
-A FORWARD -o virbr0 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -i virbr0 -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -o virbr0 -p udp -m udp --dport 68 -j ACCEPT
```

The following shows the dnsmasq process that libvirt manages as it appears in the output of `ps`:

|        |   |                                                                      |
|--------|---|----------------------------------------------------------------------|
| 2881 ? | S | 0:00 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default. |
| ↳ conf |   |                                                                      |

## How to disable libvirt networks

Although OpenStack does not make use of libvirt's networking, this networking will not interfere with OpenStack's behavior, and can be safely left enabled. However, libvirt's networking can be a nuisance when debugging OpenStack networking issues. Because libvirt creates an additional bridge, dnsmasq process, and iptables ruleset, these may distract an operator engaged in network troubleshooting. Unless you need to start up virtual machines using libvirt directly, you can safely disable libvirt's network.

To view the defined libvirt networks and their state:

|                                 |
|---------------------------------|
| # virsh net-list                |
| Name State Autostart Persistent |
| -----                           |
| default active yes yes          |

To deactivate the libvirt network named `default`:

|                             |
|-----------------------------|
| # virsh net-destroy default |
|-----------------------------|

Deactivating the network will remove the `virbr0` bridge, terminate the dnsmasq process, and remove the iptables rules.

To prevent the network from automatically starting on boot:

|                                                   |
|---------------------------------------------------|
| # virsh net-autostart --network default --disable |
|---------------------------------------------------|

To activate the network after it has been deactivated:

```
virsh net-start default
```

## neutron-linuxbridge-cleanup utility

### Description

Automated removal of empty bridges has been disabled to fix a race condition between the Compute (nova) and Networking (neutron) services. Previously, it was possible for a bridge to be deleted during the time when the only instance using it was rebooted.

### Usage

Use this script to remove empty bridges on compute nodes by running the following command:

```
$ neutron-linuxbridge-cleanup
```

---

**Important:** Do not use this tool when creating or migrating an instance as it throws an error when the bridge does not exist.

---

**Note:** Using this script can still trigger the original race condition. Only run this script if you have evacuated all instances off a compute node and you want to clean up the bridges. In addition to evacuating all instances, you should fence off the compute node where you are going to run this script so new instances do not get scheduled on it.

---

## Community support

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support and troubleshoot your installations.

### Documentation

For the available OpenStack documentation, see [docs.openstack.org](https://docs.openstack.org).

To provide feedback on documentation, join and use the [openstack-docs@lists.openstack.org](mailto:openstack-docs@lists.openstack.org) mailing list at [OpenStack Documentation Mailing List](#), join our IRC channel #openstack-doc on the freenode IRC network, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- Installation Tutorial for openSUSE Leap 42.2 and SUSE Linux Enterprise Server 12 SP2
- Installation Tutorial for Red Hat Enterprise Linux 7 and CentOS 7
- Installation Tutorial for Ubuntu 16.04 (LTS)

The following books explain how to configure and run an OpenStack cloud:

- Architecture Design Guide
- Administrator Guide
- Configuration Reference
- Operations Guide
- Networking Guide
- High Availability Guide
- Security Guide
- Virtual Machine Image Guide

The following books explain how to use the OpenStack Dashboard and command-line clients:

- End User Guide
- Command-Line Interface Reference

The following documentation provides reference and guidance information for the OpenStack APIs:

- API Guide

The following guide provides how to contribute to OpenStack documentation:

- Documentation Contributor Guide

## [ask.openstack.org](http://ask.openstack.org)

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the [ask.openstack.org](http://ask.openstack.org) site to ask questions and get answers. When you visit the [Ask OpenStack](#) site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

## OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to the [general OpenStack mailing list](#). If you are interested in the other mailing lists for specific projects or development, refer to [Mailing Lists](#).

## The OpenStack wiki

The [OpenStack wiki](#) contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or OpenStack Compute, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

## The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must [sign up for a Launchpad account](#). You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, "Kilo release" vs git commit `bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- [Bugs: OpenStack Block Storage \(cinder\)](#)
- [Bugs: OpenStack Compute \(nova\)](#)
- [Bugs: OpenStack Dashboard \(horizon\)](#)
- [Bugs: OpenStack Identity \(keystone\)](#)

- Bugs: OpenStack Image service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Application catalog (murano)
- Bugs: Bare metal service (ironic)
- Bugs: Clustering service (senlin)
- Bugs: Container Infrastructure Management service (magnum)
- Bugs: Data processing service (sahara)
- Bugs: Database service (trove)
- Bugs: Deployment service (fuel)
- Bugs: DNS service (designate)
- Bugs: Key Manager Service (barbican)
- Bugs: Monitoring (monasca)
- Bugs: Orchestration (heat)
- Bugs: Rating (cloudkitty)
- Bugs: Shared file systems (manila)
- Bugs: Telemetry (ceilometer)
- Bugs: Telemetry v3 (gnocchi)
- Bugs: Workflow service (mistral)
- Bugs: Messaging service (zaqar)
- Bugs: OpenStack API Documentation ([developer.openstack.org](https://developer.openstack.org))
- Bugs: OpenStack Documentation ([docs.openstack.org](https://docs.openstack.org))

## The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to <https://webchat.freenode.net/>. You can also use [Colloquy](#) (Mac OS X), [mIRC](#) (Windows), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at [Paste](#). Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is #openstack on irc.freenode.net. You can find a list of all OpenStack IRC channels on the [IRC](#) page on the wiki.

## Documentation feedback

To provide feedback on documentation, join and use the [openstack-docs@lists.openstack.org](mailto:openstack-docs@lists.openstack.org) mailing list at [OpenStack Documentation Mailing List](#), or report a bug.

## OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** <https://wiki.debian.org/OpenStack>
- **CentOS, Fedora, and Red Hat Enterprise Linux:** <https://www.rdoproject.org/>
- **openSUSE and SUSE Linux Enterprise Server:** <https://en.opensuse.org/Portal:OpenStack>
- **Ubuntu:** <https://wiki.ubuntu.com/ServerTeam/CloudArchive>

## Glossary

This glossary offers a list of terms and definitions to define a vocabulary for OpenStack-related concepts.

To add to OpenStack glossary, clone the [openstack/openstack-manuals](#) repository and update the source file doc/common/glossary.rst through the OpenStack contribution process.

### 0-9

**6to4** A mechanism that allows IPv6 packets to be transmitted over an IPv4 network, providing a strategy for migrating to IPv6.

### A

**absolute limit** Impassable limits for guest VMs. Settings include total RAM size, maximum number of vCPUs, and maximum disk size.

**access control list (ACL)** A list of permissions attached to an object. An ACL specifies which users or system processes have access to objects. It also defines which operations can be performed on specified objects. Each entry in a typical ACL specifies a subject and an operation. For instance, the ACL entry (Alice, delete) for a file gives Alice permission to delete the file.

**access key** Alternative term for an Amazon EC2 access key. See EC2 access key.

**account** The Object Storage context of an account. Do not confuse with a user account from an authentication service, such as Active Directory, /etc/passwd, OpenLDAP, OpenStack Identity, and so on.

**account auditor** Checks for missing replicas and incorrect or corrupted objects in a specified Object Storage account by running queries against the back-end SQLite database.

**account database** A SQLite database that contains Object Storage accounts and related metadata and that the accounts server accesses.

**account reaper** An Object Storage worker that scans for and deletes account databases and that the account server has marked for deletion.

**account server** Lists containers in Object Storage and stores container information in the account database.

**account service** An Object Storage component that provides account services such as list, create, modify, and audit. Do not confuse with OpenStack Identity service, OpenLDAP, or similar user-account services.

**accounting** The Compute service provides accounting information through the event notification and system usage data facilities.

**Active Directory** Authentication and identity service by Microsoft, based on LDAP. Supported in OpenStack.

**active/active configuration** In a high-availability setup with an active/active configuration, several systems share the load together and if one fails, the load is distributed to the remaining systems.

**active/passive configuration** In a high-availability setup with an active/passive configuration, systems are set up to bring additional resources online to replace those that have failed.

**address pool** A group of fixed and/or floating IP addresses that are assigned to a project and can be used by or assigned to the VM instances in a project.

**Address Resolution Protocol (ARP)** The protocol by which layer-3 IP addresses are resolved into layer-2 link local addresses.

**admin API** A subset of API calls that are accessible to authorized administrators and are generally not accessible to end users or the public Internet. They can exist as a separate service (keystone) or can be a subset of another API (nova).

**admin server** In the context of the Identity service, the worker process that provides access to the admin API.

**administrator** The person responsible for installing, configuring, and managing an OpenStack cloud.

**Advanced Message Queuing Protocol (AMQP)** The open standard messaging protocol used by OpenStack components for intra-service communications, provided by RabbitMQ, Qpid, or ZeroMQ.

**Advanced RISC Machine (ARM)** Lower power consumption CPU often found in mobile and embedded devices. Supported by OpenStack.

**alert** The Compute service can send alerts through its notification system, which includes a facility to create custom notification drivers. Alerts can be sent to and displayed on the dashboard.

**allocate** The process of taking a floating IP address from the address pool so it can be associated with a fixed IP on a guest VM instance.

**Amazon Kernel Image (AKI)** Both a VM container format and disk format. Supported by Image service.

**Amazon Machine Image (AMI)** Both a VM container format and disk format. Supported by Image service.

**Amazon Ramdisk Image (ARI)** Both a VM container format and disk format. Supported by Image service.

**Anvil** A project that ports the shell script-based project named DevStack to Python.

**aodh** Part of the OpenStack [Telemetry service](#); provides alarming functionality.

**Apache** The Apache Software Foundation supports the Apache community of open-source software projects. These projects provide software products for the public good.

**Apache License 2.0** All OpenStack core projects are provided under the terms of the Apache License 2.0 license.

**Apache Web Server** The most common web server software currently used on the Internet.

**API endpoint** The daemon, worker, or service that a client communicates with to access an API. API endpoints can provide any number of services, such as authentication, sales data, performance meters, Compute VM commands, census data, and so on.

**API extension** Custom modules that extend some OpenStack core APIs.

**API extension plug-in** Alternative term for a Networking plug-in or Networking API extension.

**API key** Alternative term for an API token.

**API server** Any node running a daemon or worker that provides an API endpoint.

**API token** Passed to API requests and used by OpenStack to verify that the client is authorized to run the requested operation.

**API version** In OpenStack, the API version for a project is part of the URL. For example, `example.com/nova/v1/foobar`.

**applet** A Java program that can be embedded into a web page.

**Application Catalog service (murano)** The project that provides an application catalog service so that users can compose and deploy composite environments on an application abstraction level while managing the application lifecycle.

**Application Programming Interface (API)** A collection of specifications used to access a service, application, or program. Includes service calls, required parameters for each call, and the expected return values.

**application server** A piece of software that makes available another piece of software over a network.

**Application Service Provider (ASP)** Companies that rent specialized applications that help businesses and organizations provide additional services with lower cost.

**arptables** Tool used for maintaining Address Resolution Protocol packet filter rules in the Linux kernel firewall modules. Used along with iptables, ebtables, and ip6tables in Compute to provide firewall services for VMs.

**associate** The process associating a Compute floating IP address with a fixed IP address.

**Asynchronous JavaScript and XML (AJAX)** A group of interrelated web development techniques used on the client-side to create asynchronous web applications. Used extensively in horizon.

**ATA over Ethernet (AoE)** A disk storage protocol tunneled within Ethernet.

**attach** The process of connecting a VIF or vNIC to a L2 network in Networking. In the context of Compute, this process connects a storage volume to an instance.

**attachment (network)** Association of an interface ID to a logical port. Plugs an interface into a port.

**auditing** Provided in Compute through the system usage data facility.

**auditor** A worker process that verifies the integrity of Object Storage objects, containers, and accounts. Auditors is the collective term for the Object Storage account auditor, container auditor, and object auditor.

**Austin** The code name for the initial release of OpenStack. The first design summit took place in Austin, Texas, US.

**auth node** Alternative term for an Object Storage authorization node.

**authentication** The process that confirms that the user, process, or client is really who they say they are through private key, secret token, password, fingerprint, or similar method.

**authentication token** A string of text provided to the client after authentication. Must be provided by the user or process in subsequent requests to the API endpoint.

**AuthN** The Identity service component that provides authentication services.

**authorization** The act of verifying that a user, process, or client is authorized to perform an action.

**authorization node** An Object Storage node that provides authorization services.

**AuthZ** The Identity component that provides high-level authorization services.

**Auto ACK** Configuration setting within RabbitMQ that enables or disables message acknowledgment. Enabled by default.

**auto declare** A Compute RabbitMQ setting that determines whether a message exchange is automatically created when the program starts.

**availability zone** An Amazon EC2 concept of an isolated area that is used for fault tolerance. Do not confuse with an OpenStack Compute zone or cell.

**AWS CloudFormation template** AWS CloudFormation allows Amazon Web Services (AWS) users to create and manage a collection of related resources. The Orchestration service supports a CloudFormation-compatible format (CFN).

## B

**back end** Interactions and processes that are obfuscated from the user, such as Compute volume mount, data transmission to an iSCSI target by a daemon, or Object Storage object integrity checks.

**back-end catalog** The storage method used by the Identity service catalog service to store and retrieve information about API endpoints that are available to the client. Examples include an SQL database, LDAP database, or KVS back end.

**back-end store** The persistent data store used to save and retrieve information for a service, such as lists of Object Storage objects, current state of guest VMs, lists of user names, and so on. Also, the method that the Image service uses to get and store VM images. Options include Object Storage, locally mounted file system, RADOS block devices, VMware datastore, and HTTP.

**Backup, Restore, and Disaster Recovery service (freezer)** The project that provides integrated tooling for backing up, restoring, and recovering file systems, instances, or database backups.

**bandwidth** The amount of available data used by communication resources, such as the Internet. Represents the amount of data that is used to download things or the amount of data available to download.

**barbican** Code name of the [Key Manager service](#).

**bare** An Image service container format that indicates that no container exists for the VM image.

**Bare Metal service (ironic)** The OpenStack service that provides a service and associated libraries capable of managing and provisioning physical machines in a security-aware and fault-tolerant manner.

**base image** An OpenStack-provided image.

**Bell-LaPadula model** A security model that focuses on data confidentiality and controlled access to classified information. This model divides the entities into subjects and objects. The clearance of a subject is compared to the classification of the object to determine if the subject is authorized for the specific access mode. The clearance or classification scheme is expressed in terms of a lattice.

**Benchmark service (rally)** OpenStack project that provides a framework for performance analysis and benchmarking of individual OpenStack components as well as full production OpenStack cloud deployments.

**Bexar** A grouped release of projects related to OpenStack that came out in February of 2011. It included only Compute (nova) and Object Storage (swift). Bexar is the code name for the second release of OpenStack. The design summit took place in San Antonio, Texas, US, which is the county seat for Bexar county.

**binary** Information that consists solely of ones and zeroes, which is the language of computers.

**bit** A bit is a single digit number that is in base of 2 (either a zero or one). Bandwidth usage is measured in bits per second.

**bits per second (BPS)** The universal measurement of how quickly data is transferred from place to place.

**block device** A device that moves data in the form of blocks. These device nodes interface the devices, such as hard disks, CD-ROM drives, flash drives, and other addressable regions of memory.

**block migration** A method of VM live migration used by KVM to evacuate instances from one host to another with very little downtime during a user-initiated switchover. Does not require shared storage. Supported by Compute.

**Block Storage API** An API on a separate endpoint for attaching, detaching, and creating block storage for compute VMs.

**Block Storage service (cinder)** The OpenStack service that implement services and libraries to provide on-demand, self-service access to Block Storage resources via abstraction and automation on top of other block storage devices.

**BMC (Baseboard Management Controller)** The intelligence in the IPMI architecture, which is a specialized micro-controller that is embedded on the motherboard of a computer and acts as a server. Manages the interface between system management software and platform hardware.

**bootable disk image** A type of VM image that exists as a single, bootable file.

**Bootstrap Protocol (BOOTP)** A network protocol used by a network client to obtain an IP address from a configuration server. Provided in Compute through the dnsmasq daemon when using either the FlatDHCP manager or VLAN manager network manager.

**Border Gateway Protocol (BGP)** The Border Gateway Protocol is a dynamic routing protocol that connects autonomous systems. Considered the backbone of the Internet, this protocol connects disparate networks to form a larger network.

**browser** Any client software that enables a computer or device to access the Internet.

**builder file** Contains configuration information that Object Storage uses to reconfigure a ring or to re-create it from scratch after a serious failure.

**bursting** The practice of utilizing a secondary environment to elastically build instances on-demand when the primary environment is resource constrained.

**button class** A group of related button types within horizon. Buttons to start, stop, and suspend VMs are in one class. Buttons to associate and disassociate floating IP addresses are in another class, and so on.

**byte** Set of bits that make up a single character; there are usually 8 bits to a byte.

## C

**cache pruner** A program that keeps the Image service VM image cache at or below its configured maximum size.

**Cactus** An OpenStack grouped release of projects that came out in the spring of 2011. It included Compute (nova), Object Storage (swift), and the Image service (glance). Cactus is a city in Texas, US and is the code name for the third release of OpenStack. When OpenStack releases went from three to six months long, the code name of the release changed to match a geography nearest the previous summit.

**CALL** One of the RPC primitives used by the OpenStack message queue software. Sends a message and waits for a response.

**capability** Defines resources for a cell, including CPU, storage, and networking. Can apply to the specific services within a cell or a whole cell.

**capacity cache** A Compute back-end database table that contains the current workload, amount of free RAM, and number of VMs running on each host. Used to determine on which host a VM starts.

**capacity updater** A notification driver that monitors VM instances and updates the capacity cache as needed.

**CAST** One of the RPC primitives used by the OpenStack message queue software. Sends a message and does not wait for a response.

**catalog** A list of API endpoints that are available to a user after authentication with the Identity service.

**catalog service** An Identity service that lists API endpoints that are available to a user after authentication with the Identity service.

**ceilometer** Part of the OpenStack *Telemetry service*; gathers and stores metrics from other OpenStack services.

**cell** Provides logical partitioning of Compute resources in a child and parent relationship. Requests are passed from parent cells to child cells if the parent cannot provide the requested resource.

**cell forwarding** A Compute option that enables parent cells to pass resource requests to child cells if the parent cannot provide the requested resource.

**cell manager** The Compute component that contains a list of the current capabilities of each host within the cell and routes requests as appropriate.

**CentOS** A Linux distribution that is compatible with OpenStack.

**Ceph** Massively scalable distributed storage system that consists of an object store, block store, and POSIX-compatible distributed file system. Compatible with OpenStack.

**CephFS** The POSIX-compliant file system provided by Ceph.

**certificate authority (CA)** In cryptography, an entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This enables others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the certified public key. In this model of trust relationships, a CA is a trusted third party for both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes. In OpenStack, a simple certificate authority is provided by Compute for cloudpipe VPNs and VM image decryption.

**Challenge-Handshake Authentication Protocol (CHAP)** An iSCSI authentication method supported by Compute.

**chance scheduler** A scheduling method used by Compute that randomly chooses an available host from the pool.

**changes since** A Compute API parameter that downloads changes to the requested item since your last request, instead of downloading a new, fresh set of data and comparing it against the old data.

**Chef** An operating system configuration management tool supporting OpenStack deployments.

**child cell** If a requested resource such as CPU time, disk storage, or memory is not available in the parent cell, the request is forwarded to its associated child cells. If the child cell can fulfill the request, it does. Otherwise, it attempts to pass the request to any of its children.

**cinder** Codename for *Block Storage service*.

**CirrOS** A minimal Linux distribution designed for use as a test image on clouds such as OpenStack.

**Cisco neutron plug-in** A Networking plug-in for Cisco devices and technologies, including UCS and Nexus.

**cloud architect** A person who plans, designs, and oversees the creation of clouds.

**Cloud Auditing Data Federation (CADF)** Cloud Auditing Data Federation (CADF) is a specification for audit event data. CADF is supported by OpenStack Identity.

**cloud computing** A model that enables access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, that can be rapidly provisioned and released with minimal management effort or service provider interaction.

**cloud controller** Collection of Compute components that represent the global state of the cloud; talks to services, such as Identity authentication, Object Storage, and node/storage workers through a queue.

**cloud controller node** A node that runs network, volume, API, scheduler, and image services. Each service may be broken out into separate nodes for scalability or availability.

**Cloud Data Management Interface (CDMI)** SINA standard that defines a RESTful API for managing objects in the cloud, currently unsupported in OpenStack.

**Cloud Infrastructure Management Interface (CIMI)** An in-progress specification for cloud management. Currently unsupported in OpenStack.

**cloud-init** A package commonly installed in VM images that performs initialization of an instance after boot using information that it retrieves from the metadata service, such as the SSH public key and user data.

**cloudadmin** One of the default roles in the Compute RBAC system. Grants complete system access.

**Cloudbase-Init** A Windows project providing guest initialization features, similar to cloud-init.

**clouppipe** A compute service that creates VPNs on a per-project basis.

**clouppipe image** A pre-made VM image that serves as a clouppipe server. Essentially, OpenVPN running on Linux.

**Clustering service (senlin)** The project that implements clustering services and libraries for the management of groups of homogeneous objects exposed by other OpenStack services.

**command filter** Lists allowed commands within the Compute rootwrap facility.

**Common Internet File System (CIFS)** A file sharing protocol. It is a public or open variation of the original Server Message Block (SMB) protocol developed and used by Microsoft. Like the SMB protocol, CIFS runs at a higher level and uses the TCP/IP protocol.

**Common Libraries (oslo)** The project that produces a set of python libraries containing code shared by OpenStack projects. The APIs provided by these libraries should be high quality, stable, consistent, documented and generally applicable.

**community project** A project that is not officially endorsed by the OpenStack Foundation. If the project is successful enough, it might be elevated to an incubated project and then to a core project, or it might be merged with the main code trunk.

**compression** Reducing the size of files by special encoding, the file can be decompressed again to its original content. OpenStack supports compression at the Linux file system level but does not support compression for things such as Object Storage objects or Image service VM images.

**Compute API (Nova API)** The nova-api daemon provides access to nova services. Can communicate with other APIs, such as the Amazon EC2 API.

**compute controller** The Compute component that chooses suitable hosts on which to start VM instances.

**compute host** Physical host dedicated to running compute nodes.

**compute node** A node that runs the nova-compute daemon that manages VM instances that provide a wide range of services, such as web applications and analytics.

**Compute service (nova)** The OpenStack core project that implements services and associated libraries to provide massively-scalable, on-demand, self-service access to compute resources, including bare metal, virtual machines, and containers.

**compute worker** The Compute component that runs on each compute node and manages the VM instance lifecycle, including run, reboot, terminate, attach/detach volumes, and so on. Provided by the nova-compute daemon.

**concatenated object** A set of segment objects that Object Storage combines and sends to the client.

**conductor** In Compute, conductor is the process that proxies database requests from the compute process. Using conductor improves security because compute nodes do not need direct access to the database.

**congress** Code name for the *Governance service*.

**consistency window** The amount of time it takes for a new Object Storage object to become accessible to all clients.

**console log** Contains the output from a Linux VM console in Compute.

**container** Organizes and stores objects in Object Storage. Similar to the concept of a Linux directory but cannot be nested. Alternative term for an Image service container format.

**container auditor** Checks for missing replicas or incorrect objects in specified Object Storage containers through queries to the SQLite back-end database.

**container database** A SQLite database that stores Object Storage containers and container metadata. The container server accesses this database.

**container format** A wrapper used by the Image service that contains a VM image and its associated metadata, such as machine state, OS disk size, and so on.

**Container Infrastructure Management service (magnum)** The project which provides a set of services for provisioning, scaling, and managing container orchestration engines.

**container server** An Object Storage server that manages containers.

**container service** The Object Storage component that provides container services, such as create, delete, list, and so on.

**content delivery network (CDN)** A content delivery network is a specialized network that is used to distribute content to clients, typically located close to the client for increased performance.

**controller node** Alternative term for a cloud controller node.

**core API** Depending on context, the core API is either the OpenStack API or the main API of a specific core project, such as Compute, Networking, Image service, and so on.

**core service** An official OpenStack service defined as core by DefCore Committee. Currently, consists of Block Storage service (cinder), Compute service (nova), Identity service (keystone), Image service (glance), Networking service (neutron), and Object Storage service (swift).

**cost** Under the Compute distributed scheduler, this is calculated by looking at the capabilities of each host relative to the flavor of the VM instance being requested.

**credentials** Data that is only known to or accessible by a user and used to verify that the user is who he says he is. Credentials are presented to the server during authentication. Examples include a password, secret key, digital certificate, and fingerprint.

**CRL** A Certificate Revocation List (CRL) in a PKI model is a list of certificates that have been revoked. End entities presenting these certificates should not be trusted.

**Cross-Origin Resource Sharing (CORS)** A mechanism that allows many resources (for example, fonts, JavaScript) on a web page to be requested from another domain outside the domain from which the resource originated. In particular, JavaScript's AJAX calls can use the XMLHttpRequest mechanism.

**Crowbar** An open source community project by SUSE that aims to provide all necessary services to quickly deploy and manage clouds.

**current workload** An element of the Compute capacity cache that is calculated based on the number of build, snapshot, migrate, and resize operations currently in progress on a given host.

**customer** Alternative term for project.

**customization module** A user-created Python module that is loaded by horizon to change the look and feel of the dashboard.

## D

**daemon** A process that runs in the background and waits for requests. May or may not listen on a TCP or UDP port. Do not confuse with a worker.

**Dashboard (horizon)** OpenStack project which provides an extensible, unified, web-based user interface for all OpenStack services.

**data encryption** Both Image service and Compute support encrypted virtual machine (VM) images (but not instances). In-transit data encryption is supported in OpenStack using technologies such as HTTPS, SSL, TLS, and SSH. Object Storage does not support object encryption at the application level but may support storage that uses disk encryption.

**Data loss prevention (DLP) software** Software programs used to protect sensitive information and prevent it from leaking outside a network boundary through the detection and denying of the data transportation.

**Data Processing service (sahara)** OpenStack project that provides a scalable data-processing stack and associated management interfaces.

**data store** A database engine supported by the Database service.

**database ID** A unique ID given to each replica of an Object Storage database.

**database replicator** An Object Storage component that copies changes in the account, container, and object databases to other nodes.

**Database service (trove)** An integrated project that provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

**deallocate** The process of removing the association between a floating IP address and a fixed IP address. Once this association is removed, the floating IP returns to the address pool.

**Debian** A Linux distribution that is compatible with OpenStack.

**deduplication** The process of finding duplicate data at the disk block, file, and/or object level to minimize storage use—currently unsupported within OpenStack.

**default panel** The default panel that is displayed when a user accesses the dashboard.

**default project** New users are assigned to this project if no project is specified when a user is created.

**default token** An Identity service token that is not associated with a specific project and is exchanged for a scoped token.

**delayed delete** An option within Image service so that an image is deleted after a predefined number of seconds instead of immediately.

**delivery mode** Setting for the Compute RabbitMQ message delivery mode; can be set to either transient or persistent.

**denial of service (DoS)** Denial of service (DoS) is a short form for denial-of-service attack. This is a malicious attempt to prevent legitimate users from using a service.

**deprecated auth** An option within Compute that enables administrators to create and manage users through the nova-manage command as opposed to using the Identity service.

**designate** Code name for the [DNS service](#).

**Desktop-as-a-Service** A platform that provides a suite of desktop environments that users access to receive a desktop experience from any location. This may provide general use, development, or even homogeneous testing environments.

**developer** One of the default roles in the Compute RBAC system and the default role assigned to a new user.

**device ID** Maps Object Storage partitions to physical storage devices.

**device weight** Distributes partitions proportionately across Object Storage devices based on the storage capacity of each device.

**DevStack** Community project that uses shell scripts to quickly build complete OpenStack development environments.

**DHCP agent** OpenStack Networking agent that provides DHCP services for virtual networks.

**Diablo** A grouped release of projects related to OpenStack that came out in the fall of 2011, the fourth release of OpenStack. It included Compute (nova 2011.3), Object Storage (swift 1.4.3), and the Image service (glance). Diablo is the code name for the fourth release of OpenStack. The design summit took place in the Bay Area near Santa Clara, California, US and Diablo is a nearby city.

**direct consumer** An element of the Compute RabbitMQ that comes to life when a RPC call is executed. It connects to a direct exchange through a unique exclusive queue, sends the message, and terminates.

**direct exchange** A routing table that is created within the Compute RabbitMQ during RPC calls; one is created for each RPC call that is invoked.

**direct publisher** Element of RabbitMQ that provides a response to an incoming MQ message.

**disassociate** The process of removing the association between a floating IP address and fixed IP and thus returning the floating IP address to the address pool.

**Discretionary Access Control (DAC)** Governs the ability of subjects to access objects, while enabling users to make policy decisions and assign security attributes. The traditional UNIX system of users, groups, and read-write-execute permissions is an example of DAC.

**disk encryption** The ability to encrypt data at the file system, disk partition, or whole-disk level. Supported within Compute VMs.

**disk format** The underlying format that a disk image for a VM is stored as within the Image service back-end store. For example, AMI, ISO, QCOW2, VMDK, and so on.

**dispersion** In Object Storage, tools to test and ensure dispersion of objects and containers to ensure fault tolerance.

**distributed virtual router (DVR)** Mechanism for highly available multi-host routing when using OpenStack Networking (neutron).

**Django** A web framework used extensively in horizon.

**DNS record** A record that specifies information about a particular domain and belongs to the domain.

**DNS service (designate)** OpenStack project that provides scalable, on demand, self service access to authoritative DNS services, in a technology-agnostic manner.

**dnsmasq** Daemon that provides DNS, DHCP, BOOTP, and TFTP services for virtual networks.

**domain** An Identity API v3 entity. Represents a collection of projects, groups and users that defines administrative boundaries for managing OpenStack Identity entities. On the Internet, separates a website from other sites. Often, the domain name has two or more parts that are separated by dots. For example,

yahoo.com, usa.gov, harvard.edu, or mail.yahoo.com. Also, a domain is an entity or container of all DNS-related information containing one or more records.

**Domain Name System (DNS)** A system by which Internet domain name-to-address and address-to-name resolutions are determined. DNS helps navigate the Internet by translating the IP address into an address that is easier to remember. For example, translating 111.111.111.1 into www.yahoo.com. All domains and their components, such as mail servers, utilize DNS to resolve to the appropriate locations. DNS servers are usually set up in a master-slave relationship such that failure of the master invokes the slave. DNS servers might also be clustered or replicated such that changes made to one DNS server are automatically propagated to other active servers. In Compute, the support that enables associating DNS entries with floating IP addresses, nodes, or cells so that hostnames are consistent across reboots.

**download** The transfer of data, usually in the form of files, from one computer to another.

**durable exchange** The Compute RabbitMQ message exchange that remains active when the server restarts.

**durable queue** A Compute RabbitMQ message queue that remains active when the server restarts.

**Dynamic Host Configuration Protocol (DHCP)** A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data, such as an IP address, a default route, and one or more DNS server addresses from a DHCP server. A method to automatically configure networking for a host at boot time. Provided by both Networking and Compute.

**Dynamic HyperText Markup Language (DHTML)** Pages that use HTML, JavaScript, and Cascading Style Sheets to enable users to interact with a web page or show simple animation.

## E

**east-west traffic** Network traffic between servers in the same cloud or data center. See also north-south traffic.

**EBS boot volume** An Amazon EBS storage volume that contains a bootable VM image, currently unsupported in OpenStack.

**ebtables** Filtering tool for a Linux bridging firewall, enabling filtering of network traffic passing through a Linux bridge. Used in Compute along with arptables, iptables, and ip6tables to ensure isolation of network communications.

**EC2** The Amazon commercial compute product, similar to Compute.

**EC2 access key** Used along with an EC2 secret key to access the Compute EC2 API.

**EC2 API** OpenStack supports accessing the Amazon EC2 API through Compute.

**EC2 Compatibility API** A Compute component that enables OpenStack to communicate with Amazon EC2.

**EC2 secret key** Used along with an EC2 access key when communicating with the Compute EC2 API; used to digitally sign each request.

**Elastic Block Storage (EBS)** The Amazon commercial block storage product.

**encapsulation** The practice of placing one packet type within another for the purposes of abstracting or securing data. Examples include GRE, MPLS, or IPsec.

**encryption** OpenStack supports encryption technologies such as HTTPS, SSH, SSL, TLS, digital certificates, and data encryption.

**endpoint** See API endpoint.

**endpoint registry** Alternative term for an Identity service catalog.

**endpoint template** A list of URL and port number endpoints that indicate where a service, such as Object Storage, Compute, Identity, and so on, can be accessed.

**entity** Any piece of hardware or software that wants to connect to the network services provided by Networking, the network connectivity service. An entity can make use of Networking by implementing a VIF.

**ephemeral image** A VM image that does not save changes made to its volumes and reverts them to their original state after the instance is terminated.

**ephemeral volume** Volume that does not save the changes made to it and reverts to its original state when the current user relinquishes control.

**Essex** A grouped release of projects related to OpenStack that came out in April 2012, the fifth release of OpenStack. It included Compute (nova 2012.1), Object Storage (swift 1.4.8), Image (glance), Identity (keystone), and Dashboard (horizon). Essex is the code name for the fifth release of OpenStack. The design summit took place in Boston, Massachusetts, US and Essex is a nearby city.

**ESXi** An OpenStack-supported hypervisor.

**ETag** MD5 hash of an object within Object Storage, used to ensure data integrity.

**euca2ools** A collection of command-line tools for administering VMs; most are compatible with OpenStack.

**Eucalyptus Kernel Image (EKI)** Used along with an ERI to create an EMI.

**Eucalyptus Machine Image (EMI)** VM image container format supported by Image service.

**Eucalyptus Ramdisk Image (ERI)** Used along with an EKI to create an EMI.

**evacuate** The process of migrating one or all virtual machine (VM) instances from one host to another, compatible with both shared storage live migration and block migration.

**exchange** Alternative term for a RabbitMQ message exchange.

**exchange type** A routing algorithm in the Compute RabbitMQ.

**exclusive queue** Connected to by a direct consumer in RabbitMQ—Compute, the message can be consumed only by the current connection.

**extended attributes (xattr)** File system option that enables storage of additional information beyond owner, group, permissions, modification time, and so on. The underlying Object Storage file system must support extended attributes.

**extension** Alternative term for an API extension or plug-in. In the context of Identity service, this is a call that is specific to the implementation, such as adding support for OpenID.

**external network** A network segment typically used for instance Internet access.

**extra specs** Specifies additional requirements when Compute determines where to start a new instance. Examples include a minimum amount of network bandwidth or a GPU.

## F

**FakeLDAP** An easy method to create a local LDAP directory for testing Identity and Compute. Requires Redis.

**fan-out exchange** Within RabbitMQ and Compute, it is the messaging interface that is used by the scheduler service to receive capability messages from the compute, volume, and network nodes.

**federated identity** A method to establish trusts between identity providers and the OpenStack cloud.

**Fedora** A Linux distribution compatible with OpenStack.

**Fibre Channel** Storage protocol similar in concept to TCP/IP; encapsulates SCSI commands and data.

**Fibre Channel over Ethernet (FCoE)** The fibre channel protocol tunneled within Ethernet.

**fill-first scheduler** The Compute scheduling method that attempts to fill a host with VMs rather than starting new VMs on a variety of hosts.

**filter** The step in the Compute scheduling process when hosts that cannot run VMs are eliminated and not chosen.

**firewall** Used to restrict communications between hosts and/or nodes, implemented in Compute using iptables, arptables, ip6tables, and ebtables.

**FireWall-as-a-Service (FWaaS)** A Networking extension that provides perimeter firewall functionality.

**fixed IP address** An IP address that is associated with the same instance each time that instance boots, is generally not accessible to end users or the public Internet, and is used for management of the instance.

**Flat Manager** The Compute component that gives IP addresses to authorized nodes and assumes DHCP, DNS, and routing configuration and services are provided by something else.

**flat mode injection** A Compute networking method where the OS network configuration information is injected into the VM image before the instance starts.

**flat network** Virtual network type that uses neither VLANs nor tunnels to segregate project traffic. Each flat network typically requires a separate underlying physical interface defined by bridge mappings. However, a flat network can contain multiple subnets.

**FlatDHCP Manager** The Compute component that provides dnsmasq (DHCP, DNS, BOOTP, TFTP) and radvd (routing) services.

**flavor** Alternative term for a VM instance type.

**flavor ID** UUID for each Compute or Image service VM flavor or instance type.

**floating IP address** An IP address that a project can associate with a VM so that the instance has the same public IP address each time that it boots. You create a pool of floating IP addresses and assign them to instances as they are launched to maintain a consistent IP address for maintaining DNS assignment.

**Folsom** A grouped release of projects related to OpenStack that came out in the fall of 2012, the sixth release of OpenStack. It includes Compute (nova), Object Storage (swift), Identity (keystone), Networking (neutron), Image service (glance), and Volumes or Block Storage (cinder). Folsom is the code name for the sixth release of OpenStack. The design summit took place in San Francisco, California, US and Folsom is a nearby city.

**FormPost** Object Storage middleware that uploads (posts) an image through a form on a web page.

**freezer** Code name for the *Backup, Restore, and Disaster Recovery service*.

**front end** The point where a user interacts with a service; can be an API endpoint, the dashboard, or a command-line tool.

## G

**gateway** An IP address, typically assigned to a router, that passes network traffic between different networks.

**generic receive offload (GRO)** Feature of certain network interface drivers that combines many smaller received packets into a large packet before delivery to the kernel IP stack.

**generic routing encapsulation (GRE)** Protocol that encapsulates a wide variety of network layer protocols inside virtual point-to-point links.

**glance** Codename for the *Image service*.

**glance API server** Alternative name for the *Image API*.

**glance registry** Alternative term for the Image service *image registry*.

**global endpoint template** The Identity service endpoint template that contains services available to all projects.

**GlusterFS** A file system designed to aggregate NAS hosts, compatible with OpenStack.

**gnocchi** Part of the OpenStack *Telemetry service*; provides an indexer and time-series database.

**golden image** A method of operating system installation where a finalized disk image is created and then used by all nodes without modification.

**Governance service (congress)** The project that provides Governance-as-a-Service across any collection of cloud services in order to monitor, enforce, and audit policy over dynamic infrastructure.

**Graphic Interchange Format (GIF)** A type of image file that is commonly used for animated images on web pages.

**Graphics Processing Unit (GPU)** Choosing a host based on the existence of a GPU is currently unsupported in OpenStack.

**Green Threads** The cooperative threading model used by Python; reduces race conditions and only context switches when specific library calls are made. Each OpenStack service is its own thread.

**Grizzly** The code name for the seventh release of OpenStack. The design summit took place in San Diego, California, US and Grizzly is an element of the state flag of California.

**Group** An Identity v3 API entity. Represents a collection of users that is owned by a specific domain.

**guest OS** An operating system instance running under the control of a hypervisor.

## H

**Hadoop** Apache Hadoop is an open source software framework that supports data-intensive distributed applications.

**Hadoop Distributed File System (HDFS)** A distributed, highly fault-tolerant file system designed to run on low-cost commodity hardware.

**handover** An object state in Object Storage where a new replica of the object is automatically created due to a drive failure.

**HAProxy** Provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers.

**hard reboot** A type of reboot where a physical or virtual power button is pressed as opposed to a graceful, proper shutdown of the operating system.

**Havana** The code name for the eighth release of OpenStack. The design summit took place in Portland, Oregon, US and Havana is an unincorporated community in Oregon.

**health monitor** Determines whether back-end members of a VIP pool can process a request. A pool can have several health monitors associated with it. When a pool has several monitors associated with it, all monitors check each member of the pool. All monitors must declare a member to be healthy for it to stay active.

**heat** Codename for the *Orchestration service*.

**Heat Orchestration Template (HOT)** Heat input in the format native to OpenStack.

**high availability (HA)** A high availability system design approach and associated service implementation ensures that a prearranged level of operational performance will be met during a contractual measurement period. High availability systems seek to minimize system downtime and data loss.

**horizon** Codename for the *Dashboard*.

**horizon plug-in** A plug-in for the OpenStack Dashboard (horizon).

**host** A physical computer, not a VM instance (node).

**host aggregate** A method to further subdivide availability zones into hypervisor pools, a collection of common hosts.

**Host Bus Adapter (HBA)** Device plugged into a PCI slot, such as a fibre channel or network card.

**hybrid cloud** A hybrid cloud is a composition of two or more clouds (private, community or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment models. Hybrid cloud can also mean the ability to connect colocation, managed and/or dedicated services with cloud resources.

**Hyper-V** One of the hypervisors supported by OpenStack.

**hyperlink** Any kind of text that contains a link to some other site, commonly found in documents where clicking on a word or words opens up a different website.

**Hypertext Transfer Protocol (HTTP)** An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

**Hypertext Transfer Protocol Secure (HTTPS)** An encrypted communications protocol for secure communication over a computer network, with especially wide deployment on the Internet. Technically, it is not a protocol in and of itself; rather, it is the result of simply layering the Hypertext Transfer Protocol (HTTP) on top of the TLS or SSL protocol, thus adding the security capabilities of TLS or SSL to standard HTTP communications. Most OpenStack API endpoints and many inter-component communications support HTTPS communication.

**hypervisor** Software that arbitrates and controls VM access to the actual underlying hardware.

**hypervisor pool** A collection of hypervisors grouped together through host aggregates.

## I

**Icehouse** The code name for the ninth release of OpenStack. The design summit took place in Hong Kong and Ice House is a street in that city.

**ID number** Unique numeric ID associated with each user in Identity, conceptually similar to a Linux or LDAP UID.

**Identity API** Alternative term for the Identity service API.

**Identity back end** The source used by Identity service to retrieve user information; an OpenLDAP server, for example.

**identity provider** A directory service, which allows users to login with a user name and password. It is a typical source of authentication tokens.

**Identity service (keystone)** The project that facilitates API client authentication, service discovery, distributed multi-project authorization, and auditing. It provides a central directory of users mapped to the OpenStack services they can access. It also registers endpoints for OpenStack services and acts as a common authentication system.

**Identity service API** The API used to access the OpenStack Identity service provided through keystone.

**IETF** Internet Engineering Task Force (IETF) is an open standards organization that develops Internet standards, particularly the standards pertaining to TCP/IP.

**image** A collection of files for a specific operating system (OS) that you use to create or rebuild a server. OpenStack provides pre-built images. You can also create custom images, or snapshots, from servers that you have launched. Custom images can be used for data backups or as “gold” images for additional servers.

**Image API** The Image service API endpoint for management of VM images. Processes client requests for VMs, updates Image service metadata on the registry server, and communicates with the store adapter to upload VM images from the back-end store.

**image cache** Used by Image service to obtain images on the local host rather than re-downloading them from the image server each time one is requested.

**image ID** Combination of a URI and UUID used to access Image service VM images through the image API.

**image membership** A list of projects that can access a given VM image within Image service.

**image owner** The project who owns an Image service virtual machine image.

**image registry** A list of VM images that are available through Image service.

**Image service (glance)** The OpenStack service that provide services and associated libraries to store, browse, share, distribute and manage bootable disk images, other data closely associated with initializing compute resources, and metadata definitions.

**image status** The current status of a VM image in Image service, not to be confused with the status of a running instance.

**image store** The back-end store used by Image service to store VM images, options include Object Storage, locally mounted file system, RADOS block devices, VMware datastore, or HTTP.

**image UUID** UUID used by Image service to uniquely identify each VM image.

**incubated project** A community project may be elevated to this status and is then promoted to a core project.

**Infrastructure Optimization service (watcher)** OpenStack project that aims to provide a flexible and scalable resource optimization service for multi-project OpenStack-based clouds.

**Infrastructure-as-a-Service (IaaS)** IaaS is a provisioning model in which an organization outsources physical components of a data center, such as storage, hardware, servers, and networking components. A service provider owns the equipment and is responsible for housing, operating and maintaining it. The client typically pays on a per-use basis. IaaS is a model for providing cloud services.

**ingress filtering** The process of filtering incoming network traffic. Supported by Compute.

**INI format** The OpenStack configuration files use an INI format to describe options and their values. It consists of sections and key value pairs.

**injection** The process of putting a file into a virtual machine image before the instance is started.

**Input/Output Operations Per Second (IOPS)** IOPS are a common performance measurement used to benchmark computer storage devices like hard disk drives, solid state drives, and storage area networks.

**instance** A running VM, or a VM in a known state such as suspended, that can be used like a hardware server.

**instance ID** Alternative term for instance UUID.

**instance state** The current state of a guest VM image.

**instance tunnels network** A network segment used for instance traffic tunnels between compute nodes and the network node.

**instance type** Describes the parameters of the various virtual machine images that are available to users; includes parameters such as CPU, storage, and memory. Alternative term for flavor.

**instance type ID** Alternative term for a flavor ID.

**instance UUID** Unique ID assigned to each guest VM instance.

**Intelligent Platform Management Interface (IPMI)** IPMI is a standardized computer system interface used by system administrators for out-of-band management of computer systems and monitoring of their operation. In layman's terms, it is a way to manage a computer using a direct network connection, whether it is turned on or not; connecting to the hardware rather than an operating system or login shell.

**interface** A physical or virtual device that provides connectivity to another device or medium.

**interface ID** Unique ID for a Networking VIF or vNIC in the form of a UUID.

**Internet Control Message Protocol (ICMP)** A network protocol used by network devices for control messages. For example, **ping** uses ICMP to test connectivity.

**Internet protocol (IP)** Principal communications protocol in the internet protocol suite for relaying datagrams across network boundaries.

**Internet Service Provider (ISP)** Any business that provides Internet access to individuals or businesses.

**Internet Small Computer System Interface (iSCSI)** Storage protocol that encapsulates SCSI frames for transport over IP networks. Supported by Compute, Object Storage, and Image service.

**IP address** Number that is unique to every computer system on the Internet. Two versions of the Internet Protocol (IP) are in use for addresses: IPv4 and IPv6.

**IP Address Management (IPAM)** The process of automating IP address allocation, deallocation, and management. Currently provided by Compute, melange, and Networking.

**ip6tables** Tool used to set up, maintain, and inspect the tables of IPv6 packet filter rules in the Linux kernel. In OpenStack Compute, ip6tables is used along with arptables, ebtables, and iptables to create firewalls for both nodes and VMs.

**ipset** Extension to iptables that allows creation of firewall rules that match entire “sets” of IP addresses simultaneously. These sets reside in indexed data structures to increase efficiency, particularly on systems with a large quantity of rules.

**iptables** Used along with arptables and ebtables, iptables create firewalls in Compute. iptables are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols: iptables

applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames. Requires root privilege to manipulate.

**ironic** Codename for the *Bare Metal service*.

**iSCSI Qualified Name (IQN)** IQN is the format most commonly used for iSCSI names, which uniquely identify nodes in an iSCSI network. All IQNs follow the pattern `iqn.yyyy-mm.domain:identifier`, where ‘`yyyy-mm`’ is the year and month in which the domain was registered, ‘`domain`’ is the reversed domain name of the issuing organization, and ‘`identifier`’ is an optional string which makes each IQN under the same domain unique. For example, ‘`iqn.2015-10.org.openstack.408ae959bce1`’.

**ISO9660** One of the VM image disk formats supported by Image service.

**itsec** A default role in the Compute RBAC system that can quarantine an instance in any project.

## J

**Java** A programming language that is used to create systems that involve more than one computer by way of a network.

**JavaScript** A scripting language that is used to build web pages.

**JavaScript Object Notation (JSON)** One of the supported response formats in OpenStack.

**jumbo frame** Feature in modern Ethernet networks that supports frames up to approximately 9000 bytes.

**Juno** The code name for the tenth release of OpenStack. The design summit took place in Atlanta, Georgia, US and Juno is an unincorporated community in Georgia.

## K

**Kerberos** A network authentication protocol which works on the basis of tickets. Kerberos allows nodes communication over a non-secure network, and allows nodes to prove their identity to one another in a secure manner.

**kernel-based VM (KVM)** An OpenStack-supported hypervisor. KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V), ARM, IBM Power, and IBM zSeries. It consists of a loadable kernel module, that provides the core virtualization infrastructure and a processor specific module.

**Key Manager service (barbican)** The project that produces a secret storage and generation system capable of providing key management for services wishing to enable encryption features.

**keystone** Codename of the *Identity service*.

**Kickstart** A tool to automate system configuration and installation on Red Hat, Fedora, and CentOS-based Linux distributions.

**Kilo** The code name for the eleventh release of OpenStack. The design summit took place in Paris, France. Due to delays in the name selection, the release was known only as K. Because k is the unit symbol for kilo and the kilogram reference artifact is stored near Paris in the Pavillon de Breteuil in Sèvres, the community chose Kilo as the release name.

## L

**large object** An object within Object Storage that is larger than 5 GB.

**Launchpad** The collaboration site for OpenStack.

**Layer-2 (L2) agent** OpenStack Networking agent that provides layer-2 connectivity for virtual networks.

**Layer-2 network** Term used in the OSI network architecture for the data link layer. The data link layer is responsible for media access control, flow control and detecting and possibly correcting errors that may occur in the physical layer.

**Layer-3 (L3) agent** OpenStack Networking agent that provides layer-3 (routing) services for virtual networks.

**Layer-3 network** Term used in the OSI network architecture for the network layer. The network layer is responsible for packet forwarding including routing from one node to another.

**Liberty** The code name for the twelfth release of OpenStack. The design summit took place in Vancouver, Canada and Liberty is the name of a village in the Canadian province of Saskatchewan.

**libvirt** Virtualization API library used by OpenStack to interact with many of its supported hypervisors.

**Lightweight Directory Access Protocol (LDAP)** An application protocol for accessing and maintaining distributed directory information services over an IP network.

**Linux** Unix-like computer operating system assembled under the model of free and open-source software development and distribution.

**Linux bridge** Software that enables multiple VMs to share a single physical NIC within Compute.

**Linux Bridge neutron plug-in** Enables a Linux bridge to understand a Networking port, interface attachment, and other abstractions.

**Linux containers (LXC)** An OpenStack-supported hypervisor.

**live migration** The ability within Compute to move running virtual machine instances from one host to another with only a small service interruption during switchover.

**load balancer** A load balancer is a logical device that belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration.

**load balancing** The process of spreading client requests between two or more nodes to improve performance and availability.

**Load-Balancer-as-a-Service (LBaaS)** Enables Networking to distribute incoming requests evenly between designated instances.

**Load-balancing service (octavia)** The project that aims to provide scalable, on demand, self service access to load-balancer services, in technology-agnostic manner.

**Logical Volume Manager (LVM)** Provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes.

## M

**magnum** Code name for the *Containers Infrastructure Management service*.

**management API** Alternative term for an admin API.

**management network** A network segment used for administration, not accessible to the public Internet.

**manager** Logical groupings of related code, such as the Block Storage volume manager or network manager.

**manifest** Used to track segments of a large object within Object Storage.

**manifest object** A special Object Storage object that contains the manifest for a large object.

**manila** Codename for OpenStack *Shared File Systems service*.

**manila-share** Responsible for managing Shared File System Service devices, specifically the back-end devices.

**maximum transmission unit (MTU)** Maximum frame or packet size for a particular network medium. Typically 1500 bytes for Ethernet networks.

**mechanism driver** A driver for the Modular Layer 2 (ML2) neutron plug-in that provides layer-2 connectivity for virtual instances. A single OpenStack installation can use multiple mechanism drivers.

**melange** Project name for OpenStack Network Information Service. To be merged with Networking.

**membership** The association between an Image service VM image and a project. Enables images to be shared with specified projects.

**membership list** A list of projects that can access a given VM image within Image service.

**memcached** A distributed memory object caching system that is used by Object Storage for caching.

**memory overcommit** The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as RAM overcommit.

**message broker** The software package used to provide AMQP messaging capabilities within Compute. Default package is RabbitMQ.

**message bus** The main virtual communication line used by all AMQP messages for inter-cloud communications within Compute.

**message queue** Passes requests from clients to the appropriate workers and returns the output to the client after the job completes.

**Message service (zaqar)** The project that provides a messaging service that affords a variety of distributed application patterns in an efficient, scalable and highly available manner, and to create and maintain associated Python libraries and documentation.

**Meta-Data Server (MDS)** Stores CephFS metadata.

**Metadata agent** OpenStack Networking agent that provides metadata services for instances.

**migration** The process of moving a VM instance from one host to another.

**mistral** Code name for *Workflow service*.

**Mitaka** The code name for the thirteenth release of OpenStack. The design summit took place in Tokyo, Japan. Mitaka is a city in Tokyo.

**Modular Layer 2 (ML2) neutron plug-in** Can concurrently use multiple layer-2 networking technologies, such as 802.1Q and VXLAN, in Networking.

**monasca** Codename for OpenStack *Monitoring*.

**Monitor (LBaaS)** LBaaS feature that provides availability monitoring using the ping command, TCP, and HTTP/HTTPS GET.

**Monitor (Mon)** A Ceph component that communicates with external clients, checks data state and consistency, and performs quorum functions.

**Monitoring (monasca)** The OpenStack service that provides a multi-project, highly scalable, performant, fault-tolerant monitoring-as-a-service solution for metrics, complex event processing and logging. To

build an extensible platform for advanced monitoring services that can be used by both operators and projects to gain operational insight and visibility, ensuring availability and stability.

**multi-factor authentication** Authentication method that uses two or more credentials, such as a password and a private key. Currently not supported in Identity.

**multi-host** High-availability mode for legacy (nova) networking. Each compute node handles NAT and DHCP and acts as a gateway for all of the VMs on it. A networking failure on one compute node doesn't affect VMs on other compute nodes.

**multinic** Facility in Compute that allows each virtual machine instance to have more than one VIF connected to it.

**murano** Codename for the *Application Catalog service*.

## N

**Nebula** Released as open source by NASA in 2010 and is the basis for Compute.

**netadmin** One of the default roles in the Compute RBAC system. Enables the user to allocate publicly accessible IP addresses to instances and change firewall rules.

**NetApp volume driver** Enables Compute to communicate with NetApp storage devices through the NetApp OnCommand Provisioning Manager.

**network** A virtual network that provides connectivity between entities. For example, a collection of virtual ports that share network connectivity. In Networking terminology, a network is always a layer-2 network.

**Network Address Translation (NAT)** Process of modifying IP address information while in transit. Supported by Compute and Networking.

**network controller** A Compute daemon that orchestrates the network configuration of nodes, including IP addresses, VLANs, and bridging. Also manages routing for both public and private networks.

**Network File System (NFS)** A method for making file systems available over the network. Supported by OpenStack.

**network ID** Unique ID assigned to each network segment within Networking. Same as network UUID.

**network manager** The Compute component that manages various network components, such as firewall rules, IP address allocation, and so on.

**network namespace** Linux kernel feature that provides independent virtual networking instances on a single host with separate routing tables and interfaces. Similar to virtual routing and forwarding (VRF) services on physical network equipment.

**network node** Any compute node that runs the network worker daemon.

**network segment** Represents a virtual, isolated OSI layer-2 subnet in Networking.

**Network Service Header (NSH)** Provides a mechanism for metadata exchange along the instantiated service path.

**Network Time Protocol (NTP)** Method of keeping a clock for a host or node correct via communication with a trusted, accurate time source.

**network UUID** Unique ID for a Networking network segment.

**network worker** The nova-network worker daemon; provides services such as giving an IP address to a booting nova instance.

**Networking API (Neutron API)** API used to access OpenStack Networking. Provides an extensible architecture to enable custom plug-in creation.

**Networking service (neutron)** The OpenStack project which implements services and associated libraries to provide on-demand, scalable, and technology-agnostic network abstraction.

**neutron** Codename for OpenStack *Networking service*.

**neutron API** An alternative name for *Networking API*.

**neutron manager** Enables Compute and Networking integration, which enables Networking to perform network management for guest VMs.

**neutron plug-in** Interface within Networking that enables organizations to create custom plug-ins for advanced features, such as QoS, ACLs, or IDS.

**Newton** The code name for the fourteenth release of OpenStack. The design summit took place in Austin, Texas, US. The release is named after “Newton House” which is located at 1013 E. Ninth St., Austin, TX. which is listed on the National Register of Historic Places.

**Nexenta volume driver** Provides support for NexentaStor devices in Compute.

**NFV Orchestration Service (tacker)** OpenStack service that aims to implement Network Function Virtualization (NFV) orchestration services and libraries for end-to-end life-cycle management of network services and Virtual Network Functions (VNFs).

**Nginx** An HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server.

**No ACK** Disables server-side message acknowledgment in the Compute RabbitMQ. Increases performance but decreases reliability.

**node** A VM instance that runs on a host.

**non-durable exchange** Message exchange that is cleared when the service restarts. Its data is not written to persistent storage.

**non-durable queue** Message queue that is cleared when the service restarts. Its data is not written to persistent storage.

**non-persistent volume** Alternative term for an ephemeral volume.

**north-south traffic** Network traffic between a user or client (north) and a server (south), or traffic into the cloud (south) and out of the cloud (north). See also east-west traffic.

**nova** Codename for OpenStack *Compute service*.

**Nova API** Alternative term for the *Compute API*.

**nova-network** A Compute component that manages IP address allocation, firewalls, and other network-related tasks. This is the legacy networking option and an alternative to Networking.

## O

**object** A BLOB of data held by Object Storage; can be in any format.

**object auditor** Opens all objects for an object server and verifies the MD5 hash, size, and metadata for each object.

**object expiration** A configurable option within Object Storage to automatically delete objects after a specified amount of time has passed or a certain date is reached.

**object hash** Unique ID for an Object Storage object.

**object path hash** Used by Object Storage to determine the location of an object in the ring. Maps objects to partitions.

**object replicator** An Object Storage component that copies an object to remote partitions for fault tolerance.

**object server** An Object Storage component that is responsible for managing objects.

**Object Storage API** API used to access OpenStack *Object Storage*.

**Object Storage Device (OSD)** The Ceph storage daemon.

**Object Storage service (swift)** The OpenStack core project that provides eventually consistent and redundant storage and retrieval of fixed digital content.

**object versioning** Allows a user to set a flag on an *Object Storage* container so that all objects within the container are versioned.

**Ocata** The code name for the fifteenth release of OpenStack. The design summit will take place in Barcelona, Spain. Ocata is a beach north of Barcelona.

**Octavia** Code name for the *Load-balancing service*.

**Oldie** Term for an *Object Storage* process that runs for a long time. Can indicate a hung process.

**Open Cloud Computing Interface (OCCI)** A standardized interface for managing compute, data, and network resources, currently unsupported in OpenStack.

**Open Virtualization Format (OVF)** Standard for packaging VM images. Supported in OpenStack.

**Open vSwitch** Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (for example NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).

**Open vSwitch (OVS) agent** Provides an interface to the underlying Open vSwitch service for the Networking plug-in.

**Open vSwitch neutron plug-in** Provides support for Open vSwitch in Networking.

**OpenLDAP** An open source LDAP server. Supported by both Compute and Identity.

**OpenStack** OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack is an open source project licensed under the Apache License 2.0.

**OpenStack code name** Each OpenStack release has a code name. Code names ascend in alphabetical order: Austin, Bexar, Cactus, Diablo, Essex, Folsom, Grizzly, Havana, Icehouse, Juno, Kilo, Liberty, Mitaka, Newton, Ocata, Pike, and Queens. Code names are cities or counties near where the corresponding OpenStack design summit took place. An exception, called the Waldon exception, is granted to elements of the state flag that sound especially cool. Code names are chosen by popular vote.

**openSUSE** A Linux distribution that is compatible with OpenStack.

**operator** The person responsible for planning and maintaining an OpenStack installation.

**optional service** An official OpenStack service defined as optional by DefCore Committee. Currently, consists of Dashboard (horizon), Telemetry service (Telemetry), Orchestration service (heat), Database service (trove), Bare Metal service (ironic), and so on.

**Orchestration service (heat)** The OpenStack service which orchestrates composite cloud applications using a declarative template format through an OpenStack-native REST API.

**orphan** In the context of Object Storage, this is a process that is not terminated after an upgrade, restart, or reload of the service.

**Oslo** Codename for the *Common Libraries project*.

## P

**panko** Part of the OpenStack *Telemetry service*; provides event storage.

**parent cell** If a requested resource, such as CPU time, disk storage, or memory, is not available in the parent cell, the request is forwarded to associated child cells.

**partition** A unit of storage within Object Storage used to store objects. It exists on top of devices and is replicated for fault tolerance.

**partition index** Contains the locations of all Object Storage partitions within the ring.

**partition shift value** Used by Object Storage to determine which partition data should reside on.

**path MTU discovery (PMTUD)** Mechanism in IP networks to detect end-to-end MTU and adjust packet size accordingly.

**pause** A VM state where no changes occur (no changes in memory, network communications stop, etc); the VM is frozen but not shut down.

**PCI passthrough** Gives guest VMs exclusive access to a PCI device. Currently supported in OpenStack Havana and later releases.

**persistent message** A message that is stored both in memory and on disk. The message is not lost after a failure or restart.

**persistent volume** Changes to these types of disk volumes are saved.

**personality file** A file used to customize a Compute instance. It can be used to inject SSH keys or a specific network configuration.

**Pike** The code name for the sixteenth release of OpenStack. The design summit will take place in Boston, Massachusetts, US. The release is named after the Massachusetts Turnpike, abbreviated commonly as the Mass Pike, which is the easternmost stretch of Interstate 90.

**Platform-as-a-Service (PaaS)** Provides to the consumer the ability to deploy applications through a programming language or tools supported by the cloud platform provider. An example of Platform-as-a-Service is an Eclipse/Java programming platform provided with no downloads required.

**plug-in** Software component providing the actual implementation for Networking APIs, or for Compute APIs, depending on the context.

**policy service** Component of Identity that provides a rule-management interface and a rule-based authorization engine.

**policy-based routing (PBR)** Provides a mechanism to implement packet forwarding and routing according to the policies defined by the network administrator.

**pool** A logical set of devices, such as web servers, that you group together to receive and process traffic. The load balancing function chooses which member of the pool handles the new requests or connections received on the VIP address. Each VIP has one pool.

**pool member** An application that runs on the back-end server in a load-balancing system.

**port** A virtual network port within Networking; VIFs / vNICs are connected to a port.

**port UUID** Unique ID for a Networking port.

**preseed** A tool to automate system configuration and installation on Debian-based Linux distributions.

**private image** An Image service VM image that is only available to specified projects.

**private IP address** An IP address used for management and administration, not available to the public Internet.

**private network** The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A private network interface can be a flat or VLAN network interface. A flat network interface is controlled by the flat\_interface with flat managers. A VLAN network interface is controlled by the vlan\_interface option with VLAN managers.

**project** Projects represent the base unit of “ownership” in OpenStack, in that all resources in OpenStack should be owned by a specific project. In OpenStack Identity, a project must be owned by a specific domain.

**project ID** Unique ID assigned to each project by the Identity service.

**project VPN** Alternative term for a cloupipe.

**promiscuous mode** Causes the network interface to pass all traffic it receives to the host rather than passing only the frames addressed to it.

**protected property** Generally, extra properties on an Image service image to which only cloud administrators have access. Limits which user roles can perform CRUD operations on that property. The cloud administrator can configure any image property as protected.

**provider** An administrator who has access to all hosts and instances.

**proxy node** A node that provides the Object Storage proxy service.

**proxy server** Users of Object Storage interact with the service through the proxy server, which in turn looks up the location of the requested data within the ring and returns the results to the user.

**public API** An API endpoint used for both service-to-service communication and end-user interactions.

**public image** An Image service VM image that is available to all projects.

**public IP address** An IP address that is accessible to end-users.

**public key authentication** Authentication method that uses keys rather than passwords.

**public network** The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. The public network interface is controlled by the public\_interface option.

**Puppet** An operating system configuration-management tool supported by OpenStack.

**Python** Programming language used extensively in OpenStack.

## Q

**QEMU Copy On Write 2 (QCOW2)** One of the VM image disk formats supported by Image service.

**Qpid** Message queue software supported by OpenStack; an alternative to RabbitMQ.

**Quality of Service (QoS)** The ability to guarantee certain network or storage requirements to satisfy a Service Level Agreement (SLA) between an application provider and end users. Typically includes performance requirements like networking bandwidth, latency, jitter correction, and reliability as well as storage performance in Input/Output Operations Per Second (IOPS), throttling agreements, and performance expectations at peak load.

**quarantine** If Object Storage finds objects, containers, or accounts that are corrupt, they are placed in this state, are not replicated, cannot be read by clients, and a correct copy is re-replicated.

**Queens** The code name for the seventeenth release of OpenStack. The design summit will take place in Sydney, Australia. The release is named after the Queens Pound river in the South Coast region of New South Wales.

**Quick EMULATOR (QEMU)** QEMU is a generic and open source machine emulator and virtualizer. One of the hypervisors supported by OpenStack, generally used for development purposes.

**quota** In Compute and Block Storage, the ability to set resource limits on a per-project basis.

## R

**RabbitMQ** The default message queue software used by OpenStack.

**Rackspace Cloud Files** Released as open source by Rackspace in 2010; the basis for Object Storage.

**RADOS Block Device (RBD)** Ceph component that enables a Linux block device to be striped over multiple distributed data stores.

**radvd** The router advertisement daemon, used by the Compute VLAN manager and FlatDHCP manager to provide routing services for VM instances.

**rally** Codename for the *Benchmark service*.

**RAM filter** The Compute setting that enables or disables RAM overcommitment.

**RAM overcommit** The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as memory overcommit.

**rate limit** Configurable option within Object Storage to limit database writes on a per-account and/or per-container basis.

**raw** One of the VM image disk formats supported by Image service; an unstructured disk image.

**rebalance** The process of distributing Object Storage partitions across all drives in the ring; used during initial ring creation and after ring reconfiguration.

**reboot** Either a soft or hard reboot of a server. With a soft reboot, the operating system is signaled to restart, which enables a graceful shutdown of all processes. A hard reboot is the equivalent of power cycling the server. The virtualization platform should ensure that the reboot action has completed successfully, even in cases in which the underlying domain/VM is paused or halted/stopped.

**rebuild** Removes all data on the server and replaces it with the specified image. Server ID and IP addresses remain the same.

**Recon** An Object Storage component that collects meters.

**record** Belongs to a particular domain and is used to specify information about the domain. There are several types of DNS records. Each record type contains particular information used to describe the purpose of

that record. Examples include mail exchange (MX) records, which specify the mail server for a particular domain; and name server (NS) records, which specify the authoritative name servers for a domain.

**record ID** A number within a database that is incremented each time a change is made. Used by Object Storage when replicating.

**Red Hat Enterprise Linux (RHEL)** A Linux distribution that is compatible with OpenStack.

**reference architecture** A recommended architecture for an OpenStack cloud.

**region** A discrete OpenStack environment with dedicated API endpoints that typically shares only the Identity (keystone) with other regions.

**registry** Alternative term for the Image service registry.

**registry server** An Image service that provides VM image metadata information to clients.

**Reliable, Autonomic Distributed Object Store (RADOS)**

A collection of components that provides object storage within Ceph. Similar to OpenStack Object Storage.

**Remote Procedure Call (RPC)** The method used by the Compute RabbitMQ for intra-service communications.

**replica** Provides data redundancy and fault tolerance by creating copies of Object Storage objects, accounts, and containers so that they are not lost when the underlying storage fails.

**replica count** The number of replicas of the data in an Object Storage ring.

**replication** The process of copying data to a separate physical device for fault tolerance and performance.

**replicator** The Object Storage back-end process that creates and manages object replicas.

**request ID** Unique ID assigned to each request sent to Compute.

**rescue image** A special type of VM image that is booted when an instance is placed into rescue mode. Allows an administrator to mount the file systems for an instance to correct the problem.

**resize** Converts an existing server to a different flavor, which scales the server up or down. The original server is saved to enable rollback if a problem occurs. All resizes must be tested and explicitly confirmed, at which time the original server is removed.

**RESTful** A kind of web service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.

**ring** An entity that maps Object Storage data to partitions. A separate ring exists for each service, such as account, object, and container.

**ring builder** Builds and manages rings within Object Storage, assigns partitions to devices, and pushes the configuration to other storage nodes.

**role** A personality that a user assumes to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.

**Role Based Access Control (RBAC)** Provides a predefined list of actions that the user can perform, such as start or stop VMs, reset passwords, and so on. Supported in both Identity and Compute and can be configured using the dashboard.

**role ID** Alphanumeric ID assigned to each Identity service role.

**Root Cause Analysis (RCA) service (Vitrage)** OpenStack project that aims to organize, analyze and visualize OpenStack alarms and events, yield insights regarding the root cause of problems and deduce their existence before they are directly detected.

**rootwrap** A feature of Compute that allows the unprivileged “nova” user to run a specified list of commands as the Linux root user.

**round-robin scheduler** Type of Compute scheduler that evenly distributes instances among available hosts.

**router** A physical or virtual network device that passes network traffic between different networks.

**routing key** The Compute direct exchanges, fanout exchanges, and topic exchanges use this key to determine how to process a message; processing varies depending on exchange type.

**RPC driver** Modular system that allows the underlying message queue software of Compute to be changed. For example, from RabbitMQ to ZeroMQ or Qpid.

**rsync** Used by Object Storage to push object replicas.

**RXTX cap** Absolute limit on the amount of network traffic a Compute VM instance can send and receive.

**RXTX quota** Soft limit on the amount of network traffic a Compute VM instance can send and receive.

## S

**sahara** Codename for the *Data Processing service*.

**SAML assertion** Contains information about a user as provided by the identity provider. It is an indication that a user has been authenticated.

**scheduler manager** A Compute component that determines where VM instances should start. Uses modular design to support a variety of scheduler types.

**scoped token** An Identity service API access token that is associated with a specific project.

**scrubber** Checks for and deletes unused VMs; the component of Image service that implements delayed delete.

**secret key** String of text known only by the user; used along with an access key to make requests to the Compute API.

**secure boot** Process whereby the system firmware validates the authenticity of the code involved in the boot process.

**secure shell (SSH)** Open source tool used to access remote hosts through an encrypted communications channel, SSH key injection is supported by Compute.

**security group** A set of network traffic filtering rules that are applied to a Compute instance.

**segmented object** An Object Storage large object that has been broken up into pieces. The re-assembled object is called a concatenated object.

**self-service** For IaaS, ability for a regular (non-privileged) account to manage a virtual infrastructure component such as networks without involving an administrator.

**SELinux** Linux kernel security module that provides the mechanism for supporting access control policies.

**senlin** Code name for the *Clustering service*.

**server** Computer that provides explicit services to the client software running on that system, often managing a variety of computer operations. A server is a VM instance in the Compute system. Flavor and image are requisite elements when creating a server.

**server image** Alternative term for a VM image.

**server UUID** Unique ID assigned to each VM instance.

**service** An OpenStack service, such as Compute, Object Storage, or Image service. Provides one or more endpoints through which users can access resources and perform operations.

**service catalog** Alternative term for the Identity service catalog.

**Service Function Chain (SFC)** For a given service, SFC is the abstracted view of the required service functions and the order in which they are to be applied.

**service ID** Unique ID assigned to each service that is available in the Identity service catalog.

**Service Level Agreement (SLA)** Contractual obligations that ensure the availability of a service.

**service project** Special project that contains all services that are listed in the catalog.

**service provider** A system that provides services to other system entities. In case of federated identity, OpenStack Identity is the service provider.

**service registration** An Identity service feature that enables services, such as Compute, to automatically register with the catalog.

**service token** An administrator-defined token used by Compute to communicate securely with the Identity service.

**session back end** The method of storage used by horizon to track client sessions, such as local memory, cookies, a database, or memcached.

**session persistence** A feature of the load-balancing service. It attempts to force subsequent connections to a service to be redirected to the same node as long as it is online.

**session storage** A horizon component that stores and tracks client session information. Implemented through the Django sessions framework.

**share** A remote, mountable file system in the context of the *Shared File Systems service*. You can mount a share to, and access a share from, several hosts by several users at a time.

**share network** An entity in the context of the *Shared File Systems service* that encapsulates interaction with the Networking service. If the driver you selected runs in the mode requiring such kind of interaction, you need to specify the share network to create a share.

**Shared File Systems API** A Shared File Systems service that provides a stable RESTful API. The service authenticates and routes requests throughout the Shared File Systems service. There is python-manilaclient to interact with the API.

**Shared File Systems service (manila)** The service that provides a set of services for management of shared file systems in a multi-project cloud environment, similar to how OpenStack provides block-based storage management through the OpenStack *Block Storage service* project. With the Shared File Systems service, you can create a remote file system and mount the file system on your instances. You can also read and write data from your instances to and from your file system.

**shared IP address** An IP address that can be assigned to a VM instance within the shared IP group. Public IP addresses can be shared across multiple servers for use in various high-availability scenarios. When an IP address is shared to another server, the cloud network restrictions are modified to enable each server to listen to and respond on that IP address. You can optionally specify that the target server network

configuration be modified. Shared IP addresses can be used with many standard heartbeat facilities, such as keepalive, that monitor for failure and manage IP failover.

**shared IP group** A collection of servers that can share IPs with other members of the group. Any server in a group can share one or more public IPs with any other server in the group. With the exception of the first server in a shared IP group, servers must be launched into shared IP groups. A server may be a member of only one shared IP group.

**shared storage** Block storage that is simultaneously accessible by multiple clients, for example, NFS.

**Sheepdog** Distributed block storage system for QEMU, supported by OpenStack.

**Simple Cloud Identity Management (SCIM)** Specification for managing identity in the cloud, currently unsupported by OpenStack.

**Simple Protocol for Independent Computing Environments (SPICE)** SPICE provides remote desktop access to guest virtual machines. It is an alternative to VNC. SPICE is supported by OpenStack.

**Single-root I/O Virtualization (SR-IOV)** A specification that, when implemented by a physical PCIe device, enables it to appear as multiple separate PCIe devices. This enables multiple virtualized guests to share direct access to the physical device, offering improved performance over an equivalent virtual device. Currently supported in OpenStack Havana and later releases.

**SmokeStack** Runs automated tests against the core OpenStack API; written in Rails.

**snapshot** A point-in-time copy of an OpenStack storage volume or image. Use storage volume snapshots to back up volumes. Use image snapshots to back up data, or as “gold” images for additional servers.

**soft reboot** A controlled reboot where a VM instance is properly restarted through operating system commands.

**Software Development Lifecycle Automation service (solum)** OpenStack project that aims to make cloud services easier to consume and integrate with application development process by automating the source-to-image process, and simplifying app-centric deployment.

**Software-defined networking (SDN)** Provides an approach for network administrators to manage computer network services through abstraction of lower-level functionality.

**SolidFire Volume Driver** The Block Storage driver for the SolidFire iSCSI storage appliance.

**solum** Code name for the *Software Development Lifecycle Automation service*.

**spread-first scheduler** The Compute VM scheduling algorithm that attempts to start a new VM on the host with the least amount of load.

**SQLAlchemy** An open source SQL toolkit for Python, used in OpenStack.

**SQLite** A lightweight SQL database, used as the default persistent storage method in many OpenStack services.

**stack** A set of OpenStack resources created and managed by the Orchestration service according to a given template (either an AWS CloudFormation template or a Heat Orchestration Template (HOT)).

**StackTach** Community project that captures Compute AMQP communications; useful for debugging.

**static IP address** Alternative term for a fixed IP address.

**StaticWeb** WSGI middleware component of Object Storage that serves container data as a static web page.

**storage back end** The method that a service uses for persistent storage, such as iSCSI, NFS, or local disk.

**storage manager** A XenAPI component that provides a pluggable interface to support a wide variety of persistent storage back ends.

**storage manager back end** A persistent storage method supported by XenAPI, such as iSCSI or NFS.

**storage node** An Object Storage node that provides container services, account services, and object services; controls the account databases, container databases, and object storage.

**storage services** Collective name for the Object Storage object services, container services, and account services.

**strategy** Specifies the authentication source used by Image service or Identity. In the Database service, it refers to the extensions implemented for a data store.

**subdomain** A domain within a parent domain. Subdomains cannot be registered. Subdomains enable you to delegate domains. Subdomains can themselves have subdomains, so third-level, fourth-level, fifth-level, and deeper levels of nesting are possible.

**subnet** Logical subdivision of an IP network.

**SUSE Linux Enterprise Server (SLES)** A Linux distribution that is compatible with OpenStack.

**suspend** Alternative term for a paused VM instance.

**swap** Disk-based virtual memory used by operating systems to provide more memory than is actually available on the system.

**swauth** An authentication and authorization service for Object Storage, implemented through WSGI middleware; uses Object Storage itself as the persistent backing store.

**swift** Codename for OpenStack *Object Storage service*.

**swift All in One (SAIO)** Creates a full Object Storage development environment within a single VM.

**swift middleware** Collective term for Object Storage components that provide additional functionality.

**swift proxy server** Acts as the gatekeeper to Object Storage and is responsible for authenticating the user.

**swift storage node** A node that runs Object Storage account, container, and object services.

**sync point** Point in time since the last container and accounts database sync among nodes within Object Storage.

**sysadmin** One of the default roles in the Compute RBAC system. Enables a user to add other users to a project, interact with VM images that are associated with the project, and start and stop VM instances.

**system usage** A Compute component that, along with the notification system, collects meters and usage information. This information can be used for billing.

## T

**tacker** Code name for the *NFV Orchestration service*

**Telemetry service (telemetry)** The OpenStack project which collects measurements of the utilization of the physical and virtual resources comprising deployed clouds, persists this data for subsequent retrieval and analysis, and triggers actions when defined criteria are met.

**TempAuth** An authentication facility within Object Storage that enables Object Storage itself to perform authentication and authorization. Frequently used in testing and development.

**Tempest** Automated software test suite designed to run against the trunk of the OpenStack core project.

**TempURL** An Object Storage middleware component that enables creation of URLs for temporary object access.

**tenant** A group of users; used to isolate access to Compute resources. An alternative term for a project.

**Tenant API** An API that is accessible to projects.

**tenant endpoint** An Identity service API endpoint that is associated with one or more projects.

**tenant ID** An alternative term for *project ID*.

**token** An alpha-numeric string of text used to access OpenStack APIs and resources.

**token services** An Identity service component that manages and validates tokens after a user or project has been authenticated.

**tombstone** Used to mark Object Storage objects that have been deleted; ensures that the object is not updated on another node after it has been deleted.

**topic publisher** A process that is created when a RPC call is executed; used to push the message to the topic exchange.

**Torpedo** Community project used to run automated tests against the OpenStack API.

**transaction ID** Unique ID assigned to each Object Storage request; used for debugging and tracing.

**transient** Alternative term for non-durable.

**transient exchange** Alternative term for a non-durable exchange.

**transient message** A message that is stored in memory and is lost after the server is restarted.

**transient queue** Alternative term for a non-durable queue.

**TripleO** OpenStack-on-OpenStack program. The code name for the OpenStack Deployment program.

**trove** Codename for OpenStack *Database service*.

**trusted platform module (TPM)** Specialized microprocessor for incorporating cryptographic keys into devices for authenticating and securing a hardware platform.

## U

**Ubuntu** A Debian-based Linux distribution.

**unscoped token** Alternative term for an Identity service default token.

**updater** Collective term for a group of Object Storage components that processes queued and failed updates for containers and objects.

**user** In OpenStack Identity, entities represent individual API consumers and are owned by a specific domain. In OpenStack Compute, a user can be associated with roles, projects, or both.

**user data** A blob of data that the user can specify when they launch an instance. The instance can access this data through the metadata service or config drive. Commonly used to pass a shell script that the instance runs on boot.

**User Mode Linux (UML)** An OpenStack-supported hypervisor.

## V

**VIF UUID** Unique ID assigned to each Networking VIF.

**Virtual Central Processing Unit (vCPU)** Subdivides physical CPUs. Instances can then use those divisions.

**Virtual Disk Image (VDI)** One of the VM image disk formats supported by Image service.

**Virtual Extensible LAN (VXLAN)** A network virtualization technology that attempts to reduce the scalability problems associated with large cloud computing deployments. It uses a VLAN-like encapsulation technique to encapsulate Ethernet frames within UDP packets.

**Virtual Hard Disk (VHD)** One of the VM image disk formats supported by Image service.

**virtual IP address (VIP)** An Internet Protocol (IP) address configured on the load balancer for use by clients connecting to a service that is load balanced. Incoming connections are distributed to back-end nodes based on the configuration of the load balancer.

**virtual machine (VM)** An operating system instance that runs on top of a hypervisor. Multiple VMs can run at the same time on the same physical host.

**virtual network** An L2 network segment within Networking.

**Virtual Network Computing (VNC)** Open source GUI and CLI tools used for remote console access to VMs. Supported by Compute.

**Virtual Network Interface (VIF)** An interface that is plugged into a port in a Networking network. Typically a virtual network interface belonging to a VM.

**virtual networking** A generic term for virtualization of network functions such as switching, routing, load balancing, and security using a combination of VMs and overlays on physical network infrastructure.

**virtual port** Attachment point where a virtual interface connects to a virtual network.

**virtual private network (VPN)** Provided by Compute in the form of cloudpipes, specialized instances that are used to create VPNs on a per-project basis.

**virtual server** Alternative term for a VM or guest.

**virtual switch (vSwitch)** Software that runs on a host or node and provides the features and functions of a hardware-based network switch.

**virtual VLAN** Alternative term for a virtual network.

**VirtualBox** An OpenStack-supported hypervisor.

**Vitrage** Code name for the *Root Cause Analysis service*.

**VLAN manager** A Compute component that provides dnsmasq and radvd and sets up forwarding to and from cloudpipe instances.

**VLAN network** The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A VLAN network is a private network interface, which is controlled by the `vlan_interface` option with VLAN managers.

**VM disk (VMDK)** One of the VM image disk formats supported by Image service.

**VM image** Alternative term for an image.

**VM Remote Control (VMRC)** Method to access VM instance consoles using a web browser. Supported by Compute.

**VMware API** Supports interaction with VMware products in Compute.

**VMware NSX Neutron plug-in** Provides support for VMware NSX in Neutron.

**VNC proxy** A Compute component that provides users access to the consoles of their VM instances through VNC or VMRC.

**volume** Disk-based data storage generally represented as an iSCSI target with a file system that supports extended attributes; can be persistent or ephemeral.

**Volume API** Alternative name for the Block Storage API.

**volume controller** A Block Storage component that oversees and coordinates storage volume actions.

**volume driver** Alternative term for a volume plug-in.

**volume ID** Unique ID applied to each storage volume under the Block Storage control.

**volume manager** A Block Storage component that creates, attaches, and detaches persistent storage volumes.

**volume node** A Block Storage node that runs the cinder-volume daemon.

**volume plug-in** Provides support for new and specialized types of back-end storage for the Block Storage volume manager.

**volume worker** A cinder component that interacts with back-end storage to manage the creation and deletion of volumes and the creation of compute volumes, provided by the cinder-volume daemon.

**vSphere** An OpenStack-supported hypervisor.

## W

**Watcher** Code name for the *Infrastructure Optimization service*.

**weight** Used by Object Storage devices to determine which storage devices are suitable for the job. Devices are weighted by size.

**weighted cost** The sum of each cost used when deciding where to start a new VM instance in Compute.

**weighting** A Compute process that determines the suitability of the VM instances for a job for a particular host. For example, not enough RAM on the host, too many CPUs on the host, and so on.

**worker** A daemon that listens to a queue and carries out tasks in response to messages. For example, the cinder-volume worker manages volume creation and deletion on storage arrays.

**Workflow service (mistral)** The OpenStack service that provides a simple YAML-based language to write workflows (tasks and transition rules) and a service that allows to upload them, modify, run them at scale and in a highly available manner, manage and monitor workflow execution state and state of individual tasks.

## X

**X.509** X.509 is the most widely used standard for defining digital certificates. It is a data structure that contains the subject (entity) identifiable information such as its name along with its public key. The certificate can contain a few other attributes as well depending upon the version. The most recent and standard version of X.509 is v3.

**Xen** Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.

**Xen API** The Xen administrative API, which is supported by Compute.

**Xen Cloud Platform (XCP)** An OpenStack-supported hypervisor.

**Xen Storage Manager Volume Driver** A Block Storage volume plug-in that enables communication with the Xen Storage Manager API.

**XenServer** An OpenStack-supported hypervisor.

**XFS** High-performance 64-bit file system created by Silicon Graphics. Excels in parallel I/O operations and data consistency.

## Z

**zaqar** Codename for the *Message service*.

**ZeroMQ** Message queue software supported by OpenStack. An alternative to RabbitMQ. Also spelled 0MQ.

**Zuul** Tool used in OpenStack development to ensure correctly ordered testing of changes in parallel.

## INDEX

### Symbols

6to4, [306](#)

### A

absolute limit, [306](#)

access control list (ACL), [306](#)

access key, [306](#)

account, [306](#)

account auditor, [306](#)

account database, [306](#)

account reaper, [306](#)

account server, [306](#)

account service, [306](#)

accounting, [306](#)

Active Directory, [306](#)

active/active configuration, [306](#)

active/passive configuration, [306](#)

address pool, [307](#)

Address Resolution Protocol (ARP), [307](#)

admin API, [307](#)

admin server, [307](#)

administrator, [307](#)

Advanced Message Queuing Protocol (AMQP), [307](#)

Advanced RISC Machine (ARM), [307](#)

alert, [307](#)

allocate, [307](#)

Amazon Kernel Image (AKI), [307](#)

Amazon Machine Image (AMI), [307](#)

Amazon Ramdisk Image (ARI), [307](#)

Anvil, [307](#)

aodh, [307](#)

Apache, [307](#)

Apache License 2.0, [307](#)

Apache Web Server, [307](#)

API endpoint, [307](#)

API extension, [307](#)

API extension plug-in, [307](#)

API key, [307](#)

API server, [307](#)

API token, [307](#)

API version, [307](#)

applet, [307](#)

Application Catalog service (murano), [308](#)

Application Programming Interface (API), [308](#)

application server, [308](#)

Application Service Provider (ASP), [308](#)

arptables, [308](#)

associate, [308](#)

Asynchronous JavaScript and XML (AJAX), [308](#)

ATA over Ethernet (AoE), [308](#)

attach, [308](#)

attachment (network), [308](#)

auditing, [308](#)

auditor, [308](#)

Austin, [308](#)

auth node, [308](#)

authentication, [308](#)

authentication token, [308](#)

AuthN, [308](#)

authorization, [308](#)

authorization node, [308](#)

AuthZ, [308](#)

Auto ACK, [308](#)

auto declare, [308](#)

availability zone, [308](#)

AWS CloudFormation template, [309](#)

### B

back end, [309](#)

back-end catalog, [309](#)

back-end store, [309](#)

Backup, Restore, and Disaster Recovery service (freezer), [309](#)

bandwidth, [309](#)

barbican, [309](#)

bare, [309](#)

Bare Metal service (ironic), [309](#)

base image, [309](#)

Bell-LaPadula model, [309](#)

Benchmark service (rally), [309](#)

Bexar, [309](#)

binary, [309](#)

- bit, [309](#)
- bits per second (BPS), [309](#)
- block device, [309](#)
- block migration, [309](#)
- Block Storage API, [310](#)
- Block Storage service (cinder), [310](#)
- BMC (Baseboard Management Controller), [310](#)
- bootable disk image, [310](#)
- Bootstrap Protocol (BOOTP), [310](#)
- Border Gateway Protocol (BGP), [310](#)
- browser, [310](#)
- builder file, [310](#)
- bursting, [310](#)
- button class, [310](#)
- byte, [310](#)
  
- C**
- cache pruner, [310](#)
- Cactus, [310](#)
- CALL, [310](#)
- capability, [310](#)
- capacity cache, [310](#)
- capacity updaters, [310](#)
- CAST, [310](#)
- catalog, [310](#)
- catalog service, [311](#)
- ceilometer, [311](#)
- cell, [311](#)
- cell forwarding, [311](#)
- cell manager, [311](#)
- CentOS, [311](#)
- Ceph, [311](#)
- CephFS, [311](#)
- certificate authority (CA), [311](#)
- Challenge-Handshake Authentication Protocol (CHAP), [311](#)
- chance scheduler, [311](#)
- changes since, [311](#)
- Chef, [311](#)
- child cell, [311](#)
- cinder, [311](#)
- CirrOS, [311](#)
- Cisco neutron plug-in, [311](#)
- cloud architect, [311](#)
- Cloud Auditing Data Federation (CADF), [311](#)
- cloud computing, [311](#)
- cloud controller, [311](#)
- cloud controller node, [312](#)
- Cloud Data Management Interface (CDMI), [312](#)
- Cloud Infrastructure Management Interface (CIMI), [312](#)
- cloud-init, [312](#)
- cloudadmin, [312](#)
- Cloudbase-Init, [312](#)
- cloudpipe, [312](#)
- cloudpipe image, [312](#)
- Clustering service (senlin), [312](#)
- command filter, [312](#)
- Common Internet File System (CIFS), [312](#)
- Common Libraries (oslo), [312](#)
- community project, [312](#)
- compression, [312](#)
- Compute API (Nova API), [312](#)
- compute controller, [312](#)
- compute host, [312](#)
- compute node, [312](#)
- Compute service (nova), [312](#)
- compute worker, [312](#)
- concatenated object, [312](#)
- conductor, [313](#)
- congress, [313](#)
- consistency window, [313](#)
- console log, [313](#)
- container, [313](#)
- container auditor, [313](#)
- container database, [313](#)
- container format, [313](#)
- Container Infrastructure Management service (magnum), [313](#)
- container server, [313](#)
- container service, [313](#)
- content delivery network (CDN), [313](#)
- controller node, [313](#)
- core API, [313](#)
- core service, [313](#)
- cost, [313](#)
- credentials, [313](#)
- CRL, [313](#)
- Cross-Origin Resource Sharing (CORS), [313](#)
- Crowbar, [313](#)
- current workload, [313](#)
- customer, [314](#)
- customization module, [314](#)
  
- D**
- daemon, [314](#)
- Dashboard (horizon), [314](#)
- data encryption, [314](#)
- Data loss prevention (DLP) software, [314](#)
- Data Processing service (sahara), [314](#)
- data store, [314](#)
- database ID, [314](#)

database replicator, [314](#)

Database service (trove), [314](#)

deallocate, [314](#)

Debian, [314](#)

deduplication, [314](#)

default panel, [314](#)

default project, [314](#)

default token, [314](#)

delayed delete, [314](#)

delivery mode, [314](#)

denial of service (DoS), [314](#)

deprecated auth, [314](#)

designate, [315](#)

Desktop-as-a-Service, [315](#)

developer, [315](#)

device ID, [315](#)

device weight, [315](#)

DevStack, [315](#)

DHCP agent, [315](#)

Diablo, [315](#)

direct consumer, [315](#)

direct exchange, [315](#)

direct publisher, [315](#)

disassociate, [315](#)

Discretionary Access Control (DAC), [315](#)

disk encryption, [315](#)

disk format, [315](#)

dispersion, [315](#)

distributed virtual router (DVR), [315](#)

Django, [315](#)

DNS record, [315](#)

DNS service (designate), [315](#)

dnsmasq, [315](#)

domain, [315](#)

Domain Name System (DNS), [316](#)

download, [316](#)

durable exchange, [316](#)

durable queue, [316](#)

Dynamic Host Configuration Protocol (DHCP), [316](#)

Dynamic HyperText Markup Language (DHTML),  
[316](#)

## E

east-west traffic, [316](#)

EBS boot volume, [316](#)

ebtables, [316](#)

EC2, [316](#)

EC2 access key, [316](#)

EC2 API, [316](#)

EC2 Compatibility API, [316](#)

EC2 secret key, [316](#)

Elastic Block Storage (EBS), [316](#)

encapsulation, [316](#)

encryption, [316](#)

endpoint, [316](#)

endpoint registry, [317](#)

endpoint template, [317](#)

entity, [317](#)

ephemeral image, [317](#)

ephemeral volume, [317](#)

Essex, [317](#)

ESXi, [317](#)

ETag, [317](#)

euca2ools, [317](#)

Eucalyptus Kernel Image (EKI), [317](#)

Eucalyptus Machine Image (EMI), [317](#)

Eucalyptus Ramdisk Image (ERI), [317](#)

evacuate, [317](#)

exchange, [317](#)

exchange type, [317](#)

exclusive queue, [317](#)

extended attributes (xattr), [317](#)

extension, [317](#)

external network, [317](#)

extra specs, [317](#)

## F

FakeLDAP, [317](#)

fan-out exchange, [317](#)

federated identity, [318](#)

Fedora, [318](#)

Fibre Channel, [318](#)

Fibre Channel over Ethernet (FCoE), [318](#)

fill-first scheduler, [318](#)

filter, [318](#)

firewall, [318](#)

FireWall-as-a-Service (FWaaS), [318](#)

fixed IP address, [318](#)

Flat Manager, [318](#)

flat mode injection, [318](#)

flat network, [318](#)

FlatDHCP Manager, [318](#)

flavor, [318](#)

flavor ID, [318](#)

floating IP address, [318](#)

Folsom, [318](#)

FormPost, [318](#)

freezer, [318](#)

front end, [318](#)

## G

gateway, [318](#)

generic receive offload (GRO), [319](#)

- generic routing encapsulation (GRE), [319](#)
- glance, [319](#)
  - glance API server, [319](#)
  - glance registry, [319](#)
  - global endpoint template, [319](#)
  - GlusterFS, [319](#)
  - gnocchi, [319](#)
  - golden image, [319](#)
  - Governance service (congress), [319](#)
  - Graphic Interchange Format (GIF), [319](#)
  - Graphics Processing Unit (GPU), [319](#)
  - Green Threads, [319](#)
  - Grizzly, [319](#)
  - Group, [319](#)
  - guest OS, [319](#)
- H**
  - Hadoop, [319](#)
  - Hadoop Distributed File System (HDFS), [319](#)
  - handover, [319](#)
  - HAProxy, [319](#)
  - hard reboot, [319](#)
  - Havana, [319](#)
  - health monitor, [320](#)
  - heat, [320](#)
  - Heat Orchestration Template (HOT), [320](#)
  - high availability (HA), [320](#)
  - horizon, [320](#)
  - horizon plug-in, [320](#)
  - host, [320](#)
  - host aggregate, [320](#)
  - Host Bus Adapter (HBA), [320](#)
  - hybrid cloud, [320](#)
  - Hyper-V, [320](#)
  - hyperlink, [320](#)
  - Hypertext Transfer Protocol (HTTP), [320](#)
  - Hypertext Transfer Protocol Secure (HTTPS), [320](#)
  - hypervisor, [320](#)
  - hypervisor pool, [320](#)
- I**
  - Icehouse, [320](#)
  - ID number, [320](#)
  - Identity API, [320](#)
  - Identity back end, [321](#)
  - identity provider, [321](#)
  - Identity service (keystone), [321](#)
  - Identity service API, [321](#)
  - IETF, [321](#)
  - image, [321](#)
  - Image API, [321](#)
  - image cache, [321](#)
- image ID, [321](#)
  - image membership, [321](#)
  - image owner, [321](#)
  - image registry, [321](#)
  - Image service (glance), [321](#)
  - image status, [321](#)
  - image store, [321](#)
  - image UUID, [321](#)
  - incubated project, [321](#)
  - Infrastructure Optimization service (watcher), [321](#)
  - Infrastructure-as-a-Service (IaaS), [321](#)
  - ingress filtering, [321](#)
  - INI format, [322](#)
  - injection, [322](#)
  - Input/Output Operations Per Second (IOPS), [322](#)
  - instance, [322](#)
  - instance ID, [322](#)
  - instance state, [322](#)
  - instance tunnels network, [322](#)
  - instance type, [322](#)
  - instance type ID, [322](#)
  - instance UUID, [322](#)
  - Intelligent Platform Management Interface (IPMI), [322](#)
  - interface, [322](#)
  - interface ID, [322](#)
  - Internet Control Message Protocol (ICMP), [322](#)
  - Internet protocol (IP), [322](#)
  - Internet Service Provider (ISP), [322](#)
  - Internet Small Computer System Interface (iSCSI), [322](#)
  - IP address, [322](#)
  - IP Address Management (IPAM), [322](#)
  - ip6tables, [322](#)
  - ipset, [322](#)
  - iptables, [322](#)
  - ironic, [323](#)
  - iSCSI Qualified Name (IQN), [323](#)
  - ISO9660, [323](#)
  - itsec, [323](#)
- J**
  - Java, [323](#)
  - JavaScript, [323](#)
  - JavaScript Object Notation (JSON), [323](#)
  - jumbo frame, [323](#)
  - Juno, [323](#)
- K**
  - Kerberos, [323](#)
  - kernel-based VM (KVM), [323](#)
  - Key Manager service (barbican), [323](#)

keystone, [323](#)

Kickstart, [323](#)

Kilo, [323](#)

## L

large object, [323](#)

Launchpad, [324](#)

Layer-2 (L2) agent, [324](#)

Layer-2 network, [324](#)

Layer-3 (L3) agent, [324](#)

Layer-3 network, [324](#)

Liberty, [324](#)

libvirt, [324](#)

Lightweight Directory Access Protocol (LDAP), [324](#)

Linux, [324](#)

Linux bridge, [324](#)

Linux Bridge neutron plug-in, [324](#)

Linux containers (LXC), [324](#)

live migration, [324](#)

load balancer, [324](#)

load balancing, [324](#)

Load-Balancer-as-a-Service (LBaaS), [324](#)

Load-balancing service (octavia), [324](#)

Logical Volume Manager (LVM), [324](#)

## M

magnum, [324](#)

management API, [324](#)

management network, [324](#)

manager, [324](#)

manifest, [324](#)

manifest object, [325](#)

manila, [325](#)

manila-share, [325](#)

maximum transmission unit (MTU), [325](#)

mechanism driver, [325](#)

melange, [325](#)

membership, [325](#)

membership list, [325](#)

memcached, [325](#)

memory overcommit, [325](#)

message broker, [325](#)

message bus, [325](#)

message queue, [325](#)

Message service (zaqar), [325](#)

Meta-Data Server (MDS), [325](#)

Metadata agent, [325](#)

migration, [325](#)

mistral, [325](#)

Mitaka, [325](#)

Modular Layer 2 (ML2) neutron plug-in, [325](#)

monasca, [325](#)

Monitor (LBaaS), [325](#)

Monitor (Mon), [325](#)

Monitoring (monasca), [325](#)

multi-factor authentication, [326](#)

multi-host, [326](#)

multinic, [326](#)

murano, [326](#)

## N

Nebula, [326](#)

netadmin, [326](#)

NetApp volume driver, [326](#)

network, [326](#)

Network Address Translation (NAT), [326](#)

network controller, [326](#)

Network File System (NFS), [326](#)

network ID, [326](#)

network manager, [326](#)

network namespace, [326](#)

network node, [326](#)

network segment, [326](#)

Network Service Header (NSH), [326](#)

Network Time Protocol (NTP), [326](#)

network UUID, [326](#)

network worker, [326](#)

Networking API (Neutron API), [327](#)

Networking service (neutron), [327](#)

neutron, [327](#)

neutron API, [327](#)

neutron manager, [327](#)

neutron plug-in, [327](#)

Newton, [327](#)

Nexenta volume driver, [327](#)

NFV Orchestration Service (tacker), [327](#)

Nginx, [327](#)

No ACK, [327](#)

node, [327](#)

non-durable exchange, [327](#)

non-durable queue, [327](#)

non-persistent volume, [327](#)

north-south traffic, [327](#)

nova, [327](#)

Nova API, [327](#)

nova-network, [327](#)

## O

object, [327](#)

object auditor, [327](#)

object expiration, [327](#)

object hash, [328](#)

object path hash, [328](#)

object replicator, [328](#)

- object server, [328](#)
- Object Storage API, [328](#)
- Object Storage Device (OSD), [328](#)
- Object Storage service (swift), [328](#)
- object versioning, [328](#)
- Ocata, [328](#)
- Octavia, [328](#)
- Oldie, [328](#)
- Open Cloud Computing Interface (OCCI), [328](#)
- Open Virtualization Format (OVF), [328](#)
- Open vSwitch, [328](#)
- Open vSwitch (OVS) agent, [328](#)
- Open vSwitch neutron plug-in, [328](#)
- OpenLDAP, [328](#)
- OpenStack, [328](#)
- OpenStack code name, [328](#)
- openSUSE, [328](#)
- operator, [328](#)
- optional service, [328](#)
- Orchestration service (heat), [329](#)
- orphan, [329](#)
- Oslo, [329](#)
  
- P**
- panko, [329](#)
- parent cell, [329](#)
- partition, [329](#)
- partition index, [329](#)
- partition shift value, [329](#)
- path MTU discovery (PMTUD), [329](#)
- pause, [329](#)
- PCI passthrough, [329](#)
- persistent message, [329](#)
- persistent volume, [329](#)
- personality file, [329](#)
- Pike, [329](#)
- Platform-as-a-Service (PaaS), [329](#)
- plug-in, [329](#)
- policy service, [329](#)
- policy-based routing (PBR), [329](#)
- pool, [329](#)
- pool member, [330](#)
- port, [330](#)
- port UUID, [330](#)
- preseed, [330](#)
- private image, [330](#)
- private IP address, [330](#)
- private network, [330](#)
- project, [330](#)
- project ID, [330](#)
- project VPN, [330](#)
- promiscuous mode, [330](#)
- protected property, [330](#)
- provider, [330](#)
- proxy node, [330](#)
- proxy server, [330](#)
- public API, [330](#)
- public image, [330](#)
- public IP address, [330](#)
- public key authentication, [330](#)
- public network, [330](#)
- Puppet, [330](#)
- Python, [330](#)
  
- Q**
- QEMU Copy On Write 2 (QCOW2), [330](#)
- Qpid, [330](#)
- Quality of Service (QoS), [331](#)
- quarantine, [331](#)
- Queens, [331](#)
- Quick EMULATOR (QEMU), [331](#)
- quota, [331](#)
  
- R**
- RabbitMQ, [331](#)
- Rackspace Cloud Files, [331](#)
- RADOS Block Device (RBD), [331](#)
- radvd, [331](#)
- rally, [331](#)
- RAM filter, [331](#)
- RAM overcommit, [331](#)
- rate limit, [331](#)
- raw, [331](#)
- rebalance, [331](#)
- reboot, [331](#)
- rebuild, [331](#)
- Recon, [331](#)
- record, [331](#)
- record ID, [332](#)
- Red Hat Enterprise Linux (RHEL), [332](#)
- reference architecture, [332](#)
- region, [332](#)
- registry, [332](#)
- registry server, [332](#)
- Reliable, Autonomic Distributed Object Store, [332](#)
- Remote Procedure Call (RPC), [332](#)
- replica, [332](#)
- replica count, [332](#)
- replication, [332](#)
- replicator, [332](#)
- request ID, [332](#)
- rescue image, [332](#)
- resize, [332](#)

- RESTful, [332](#)
- ring, [332](#)
- ring builder, [332](#)
- role, [332](#)
  - Role Based Access Control (RBAC), [332](#)
  - role ID, [332](#)
- Root Cause Analysis (RCA) service (Vitrage), [333](#)
- rootwrap, [333](#)
- round-robin scheduler, [333](#)
- router, [333](#)
- routing key, [333](#)
- RPC driver, [333](#)
- rsync, [333](#)
- RXTX cap, [333](#)
- RXTX quota, [333](#)
- S**
  - sahara, [333](#)
  - SAML assertion, [333](#)
  - scheduler manager, [333](#)
  - scoped token, [333](#)
  - scrubber, [333](#)
  - secret key, [333](#)
  - secure boot, [333](#)
  - secure shell (SSH), [333](#)
  - security group, [333](#)
  - segmented object, [333](#)
  - self-service, [333](#)
  - SELinux, [333](#)
  - senlin, [333](#)
  - server, [334](#)
  - server image, [334](#)
  - server UUID, [334](#)
  - service, [334](#)
  - service catalog, [334](#)
  - Service Function Chain (SFC), [334](#)
  - service ID, [334](#)
  - Service Level Agreement (SLA), [334](#)
  - service project, [334](#)
  - service provider, [334](#)
  - service registration, [334](#)
  - service token, [334](#)
  - session back end, [334](#)
  - session persistence, [334](#)
  - session storage, [334](#)
  - share, [334](#)
  - share network, [334](#)
  - Shared File Systems API, [334](#)
  - Shared File Systems service (manila), [334](#)
  - shared IP address, [334](#)
  - shared IP group, [335](#)
  - shared storage, [335](#)
  - Sheepdog, [335](#)
  - Simple Cloud Identity Management (SCIM), [335](#)
  - Simple Protocol for Independent Computing Environments (SPICE), [335](#)
  - Single-root I/O Virtualization (SR-IOV), [335](#)
  - SmokeStack, [335](#)
  - snapshot, [335](#)
  - soft reboot, [335](#)
  - Software Development Lifecycle Automation service (solum), [335](#)
  - Software-defined networking (SDN), [335](#)
  - SolidFire Volume Driver, [335](#)
  - solum, [335](#)
  - spread-first scheduler, [335](#)
  - SQLAlchemy, [335](#)
  - SQLite, [335](#)
  - stack, [335](#)
  - StackTach, [335](#)
  - static IP address, [335](#)
  - StaticWeb, [335](#)
  - storage back end, [335](#)
  - storage manager, [336](#)
  - storage manager back end, [336](#)
  - storage node, [336](#)
  - storage services, [336](#)
  - strategy, [336](#)
  - subdomain, [336](#)
  - subnet, [336](#)
  - SUSE Linux Enterprise Server (SLES), [336](#)
  - suspend, [336](#)
  - swap, [336](#)
  - swauth, [336](#)
  - swift, [336](#)
  - swift All in One (SAIO), [336](#)
  - swift middleware, [336](#)
  - swift proxy server, [336](#)
  - swift storage node, [336](#)
  - sync point, [336](#)
  - sysadmin, [336](#)
  - system usage, [336](#)
- T**
  - tacker, [336](#)
  - Telemetry service (telemetry), [336](#)
  - TempAuth, [336](#)
  - Tempest, [336](#)
  - TempURL, [337](#)
  - tenant, [337](#)
  - Tenant API, [337](#)
  - tenant endpoint, [337](#)

- tenant ID, [337](#)
- token, [337](#)
- token services, [337](#)
- tombstone, [337](#)
- topic publisher, [337](#)
- Torpedo, [337](#)
- transaction ID, [337](#)
- transient, [337](#)
- transient exchange, [337](#)
- transient message, [337](#)
- transient queue, [337](#)
- TripleO, [337](#)
- trove, [337](#)
- trusted platform module (TPM), [337](#)
  
- U**
- Ubuntu, [337](#)
- unscoped token, [337](#)
- updater, [337](#)
- user, [337](#)
- user data, [337](#)
- User Mode Linux (UML), [337](#)
  
- V**
- VIF UUID, [338](#)
- Virtual Central Processing Unit (vCPU), [338](#)
- Virtual Disk Image (VDI), [338](#)
- Virtual Extensible LAN (VXLAN), [338](#)
- Virtual Hard Disk (VHD), [338](#)
- virtual IP address (VIP), [338](#)
- virtual machine (VM), [338](#)
- virtual network, [338](#)
- Virtual Network Computing (VNC), [338](#)
- Virtual Network InterFace (VIF), [338](#)
- virtual networking, [338](#)
- virtual port, [338](#)
- virtual private network (VPN), [338](#)
- virtual server, [338](#)
- virtual switch (vSwitch), [338](#)
- virtual VLAN, [338](#)
- VirtualBox, [338](#)
- Vitrage, [338](#)
- VLAN manager, [338](#)
- VLAN network, [338](#)
- VM disk (VMDK), [338](#)
- VM image, [338](#)
- VM Remote Control (VMRC), [338](#)
- VMware API, [339](#)
- VMware NSX Neutron plug-in, [339](#)
- VNC proxy, [339](#)
- volume, [339](#)
- Volume API, [339](#)
  
- volume controller, [339](#)
- volume driver, [339](#)
- volume ID, [339](#)
- volume manager, [339](#)
- volume node, [339](#)
- volume plug-in, [339](#)
- volume worker, [339](#)
- vSphere, [339](#)
  
- W**
- Watcher, [339](#)
- weight, [339](#)
- weighted cost, [339](#)
- weighting, [339](#)
- worker, [339](#)
- Workflow service (mistral), [339](#)
  
- X**
- X.509, [339](#)
- Xen, [339](#)
- Xen API, [340](#)
- Xen Cloud Platform (XCP), [340](#)
- Xen Storage Manager Volume Driver, [340](#)
- XenServer, [340](#)
- XFS, [340](#)
  
- Z**
- zaqar, [340](#)
- ZeroMQ, [340](#)
- Zuul, [340](#)