OpenStack Networking Essentials

OpenStack Networking Essentials kicks off by describing the various components of OpenStack Neutron and walks you through installing OpenStack on CentOS using RDO and Packstack. Further on, you will use various methods to interface with Neutron in order to create and manage network resources. You will also get to grips with the relationships among ports, networks, and subnets through diagrams and explanations and see how the logical components are implemented via plugins and agents. Moving forward, you will learn how virtual switches are implemented and how to build Neutron routers. You will also configure networks, subnets, and routers to provide connectivity to instances using simple examples. Towards the end, you will configure and manage security groups, and will observe how these rules translate to iptables rules on host machines.

What you will learn from this book

- Install the latest Liberty (2015.2) release of OpenStack using RDO in VirtualBox
- Discover the basics of the Neutron API, including networks, subnets, and ports
- Interact with Neutron using the CLI and the Horizon dashboard
- Create networks and subnets that provide connectivity to instances
- Secure instances using Neutron’s security group functionality

Who this book is written for

The book is for those who are new to OpenStack and Neutron and want to learn cloud networking fundamentals and get started with OpenStack networking. Prior networking experience and a virtual or physical server is recommended in order to follow along with the concepts demonstrated in the book.


What this book is about

- Build and manage networks in OpenStack using Neutron
- Install the latest Liberty (2015.2) release of OpenStack using RDO in VirtualBox
- Discover the basics of the Neutron API, including networks, subnets, and ports
- Interact with Neutron using the CLI and the Horizon dashboard
- Create networks and subnets that provide connectivity to instances
- Secure instances using Neutron’s security group functionality

James Denton

OpenStack Networking Essentials

Build and manage networks in OpenStack using Neutron
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 6 'Routing'
- A synopsis of the book’s content
- More information on OpenStack Networking Essentials
About the Author

James Denton has more than 15 years of experience in system administration and networking and has been deploying, operating, and maintaining OpenStack clouds since late 2012. He is a Principal Architect at Rackspace, and prior to joining the Rackspace Private Cloud team, he spent 5 years as an enterprise network security engineer. James has a bachelor's degree in business management, with a focus on computer information systems, from Texas State University in San Marcos, Texas. In his spare time, James enjoys spending time with his wife and son and camping in the Texas hill country.
Preface

OpenStack is an open source cloud operating system designed to control pools of compute, storage, and networking resources. This powerful system fosters rapid innovation while decreasing operational and capital costs. OpenStack has exploded in popularity in recent years, thanks to its features, flexibility, and overall maturity.

In this book, we will explore the networking component of OpenStack, known as Neutron. Neutron provides an API for users to build virtual network resources such as switches, routers, load balancers, and firewalls. We will walk through the installation of OpenStack using RDO and will look at the core components of the API, made up of networks, subnets, and ports. By the end of the book, you will have harnessed the power of OpenStack and Neutron to create and access virtual network resources of your own.

What this book covers

Chapter 1, OpenStack Networking Components – an Overview, provides an introduction to OpenStack Networking features, components, and the basic physical architectures required to support an OpenStack cloud.

Chapter 2, Installing OpenStack Using RDO, provides instructions for installing the Liberty release of OpenStack using RDO on the CentOS 7.1 operating system.

Chapter 3, Neutron API Basics, looks at the core components of the Neutron API, made up of networks, subnets, and ports, and how they're used to construct virtual networks.

Chapter 4, Interfacing with Neutron, explores the use of the Horizon dashboard and the Neutron command-line client to interface with the Neutron API.

Chapter 5, Switching, looks at how Neutron constructs and implements the virtual network infrastructure to enable the flow of traffic across the cloud.
Chapter 6, Routing, discusses how Neutron implements virtual routers that provide routing between Neutron networks and the outside world using source network address translation and floating IPs.

Chapter 7, Building Networks and Routers, covers basic virtual network architectures and showcases the traffic flow from client workstations to virtual machine instances via fixed and floating IPs.

Chapter 8, Security Group Fundamentals, examines the use of Neutron security groups to secure instance traffic at the virtual switch port and walks you through creating and managing security groups and associated rules.

Appendix, Configuring VirtualBox, is meant to assist with the setup of a virtual environment using VirtualBox so that many of the examples throughout the book can be followed.

What you need for this book

For this book, the following is required:

- Operating system:
  - CentOS Linux 7.1

- Software:
  - VirtualBox 5.0 or higher
  - RDO (Liberty release)

This book assumes a beginner-to-moderate level of networking experience and experience with Linux operating systems. While this book will walk you through a basic installation of OpenStack using RDO, little time will be spent on services other than Neutron as well as any configuration of OpenStack outside of what's available via the API. It will be helpful for you have a basic understanding of OpenStack and its components prior to reading this book.

Internet connectivity is required to install OpenStack packages. An all-in-one OpenStack deployment will be performed on a single virtual machine within VirtualBox. CentOS must be installed prior to installing RDO. Alternative virtualization platforms such as VMware, or physical hardware, are optional.
Routing

In the previous chapter, we discovered how Neutron builds out logical networks using two of the most popular open source virtual switching platforms for OpenStack: Open vSwitch and LinuxBridge. In this chapter, we will take a look at how Neutron implements virtual routers that provide routing between Neutron networks and the outside world. Concepts that will be covered include network namespaces, Source Network Address Translation (SNAT), and floating IPs.

The basics of routing in Neutron

If you recall from the previous chapter, users can create and manage networks known as tenant networks within their respective project without any knowledge of the underlying infrastructure. By default, instances connected to tenant networks are isolated from other networks and are unable to access external resources such as the Internet. Neutron provides connectivity to instances in tenant networks by way of virtual routers.

Network namespaces

In a reference implementation, virtual routers created in Neutron are implemented as network namespaces that reside on nodes running the Neutron L3 agent service. Network namespaces are similar in function to Virtual Routing and Forwarding (VRF) domains, where multiple instances of a routing table exist to provide complete network segregation in a single device. The use of network namespaces allows Neutron to support overlapping subnets across networks. In Linux, network namespaces can be managed using the `ip netns` command as the root user or a user with `sudo` privileges. Router namespaces follow the `qrouter-<router_id>` naming convention.
In addition to providing dedicated routing tables to each namespace, Linux allows processes like dnsmasq to be run and contained within namespaces. Neutron uses network namespaces to isolate DHCP services between networks. DHCP namespaces follow the qdhcp-<network_id> naming convention.

The Neutron L3 agent service usually runs on controller nodes or dedicated network nodes, but as we’ll see later, the service can also run on compute nodes to help provide a smaller failure zone when using distributed virtual routers.

**Connectivity through a router**

At its most basic level, a Neutron router acts as a default gateway for one or more connected tenant networks, as shown in the following diagram:

Instances in **Tenant Network A** can use the Neutron router as their default gateway to communicate with instances in **Tenant Network B**, and vice versa. When an external provider network is attached to the Neutron router, the router can route traffic upstream to its respective gateway device. In most cases, a physical routing device will be the gateway for a Neutron router. The following diagram demonstrates a physical gateway and the Neutron router connected to a common provider network:
In the diagram, a Neutron router is connected to multiple tenant networks and serves as the default gateway for those networks. The Neutron router is also connected to a provider network that provides access to external networks, including the Internet.
Outbound connectivity

By default, Neutron routers will apply Source Network Address Translation (SNAT) to all outbound traffic from connected tenant networks. This means that, as traffic exits the virtual router and heads upstream, the router modifies each packet and changes the source IP address to that of its own external interface. This ensures that return traffic gets directed back to the virtual router, where the destination IP address is modified from the router's address back to the original client. The following diagram demonstrates a router performing SNAT for a virtual machine instance:

In the diagram, the outbound traffic from VM 1 is modified as it traverses the Neutron router towards its destination. As each packet leaves the router, the source address is modified. As the inbound response traffic enters and traverses the router, the destination address is changed from that of the router to that of the virtual machine.

Inbound connectivity

In a SNAT scenario, all traffic leaving the router appears to come from the same address. In addition, inbound connections cannot be made directly to a SNAT address, which means that that address cannot be used to reach the instance directly.
A floating IP is an address that is used to provide a 1:1 static NAT mapping to a single fixed IP. In Nova/Neutron-speak, a fixed IP is an IP address associated with an instance and, by definition, a Neutron port. Floating IPs provide a unique outbound and inbound address; this allows clients to reach individual virtual machine instances and other devices. The following diagram demonstrates a router performing an address translation using a floating IP:

In the diagram, the outbound traffic from VM 1 is modified as it traverses the Neutron router towards its destination. As each packet leaves the router, the source address is modified to that of the floating IP. As the inbound response traffic enters and traverses the router, the destination address is changed from that of the router to that of the virtual machine. This is similar in operation to the earlier SNAT example. Rather than using a shared address, however, the floating IP is dedicated to traffic associated with the fixed IP of the instance. Inbound connections to the floating IP from an external network are translated to the respective fixed IP and directed to the appropriate resource or instance.
Types of routers
Neutron routers act as the default gateway for connected tenant networks and provide outbound and inbound connectivity to the instances they service. Neutron provides three types of routers to users:

- Standalone
- Highly available
- Distributed

Routers can be created in both the Horizon dashboard and via the Neutron CLI. As an ordinary user, the type of router that is created via the API is predetermined, based on a combination of settings found in the Neutron server and L3 agent configuration files. Users with the admin role are free to define the type of router to be created using the `router-create` command, using the following flags:

```
--distributed {true | false}
--ha {true | false}
```

Neutron does not expose the router type to users via the API, even with the `router-show` command. Users with the admin role, however, can see those details. The Horizon dashboard limits all users, including administrative users, to the default router type specified in the configuration. This behavior may change in future releases.

Standalone routers
A standalone router is a single logical router that is implemented as a single network namespace on a host running the Neutron L3 agent. Most often, the L3 agent runs on dedicated network nodes or the controller nodes themselves. By its very nature, a standalone router is a single point of failure for directly connected networks. If the node hosting the network namespace experiences issues, connectivity through the namespace can become limited or completely unavailable. Needless to say, the failure of a standalone router can result in an unhappy user experience.

Standalone routers have been the default router type since the Folsom release of OpenStack and are supported by both the Open vSwitch and LinuxBridge mechanism drivers and agents.
Highly available routers

A **Highly Available (HA)** router is a single logical router that is implemented as two or more network namespaces on hosts running the Neutron L3 agent. Like its standalone counterpart, an HA router is likely to be spread across dedicated network or controller nodes. An HA router utilizes the `keepalived` service and the **Virtual Routing Redundancy Protocol (VRRP)** between network namespaces to provide high availability. Only one of the network namespaces acts as a master virtual router at any given time while the others remain in a backup state awaiting a failover event. If the active router fails, a backup router will take over quickly. While HA routers provide redundancy not found with standalone routers, pushing all traffic through a subset of nodes may still be seen as a bottleneck that can result in poor network performance.

Highly available routers have been available since the Juno release of OpenStack and are supported by both the Open vSwitch and LinuxBridge mechanism drivers and agents.

Distributed virtual routers

A **Distributed Virtual Router (DVR)** is a single logical router that is implemented as multiple network namespaces on network and compute nodes. The model of distributing virtual routers across compute nodes is similar to the multihost functionality of **Nova Network**. It offers high availability of networking by limiting single points of failure to individual compute nodes rather than network nodes.

Distributed virtual routers have been available since the Juno release of OpenStack and, as of the Liberty release, are supported only by the Open vSwitch mechanism driver and agent.

Managing routers in the dashboard
Like networks, virtual routers can be created and managed within the Horizon dashboard and by using the Neutron command-line client.

Creating routers within a project
To create a router, follow these steps:

1. Navigate to the Project | Network | Routers section of the Horizon dashboard and click on the Create Router button in the upper right-hand corner of the screen, as shown here:

2. A single-step router creation wizard will appear. Name the router in the Router Name field, as shown here:

3. Click on the Create Router button to complete the wizard and return to the Routers screen, as shown in the following screenshot:
Congratulations, you just created a virtual router! Behind the scenes, Neutron has determined the type of router to create and may have implemented one or more network namespaces as a result. In this environment, the default router type is standalone, which means a single network namespace will be created that will serve as the virtual router. In Chapter 7, Building Networks and Routers, we'll build some common network topologies and observe traffic flow through a Neutron router.

**Viewing the network topology**

Now that we have created a router, let's view the resulting network topology. Here are the steps:

1. From the **Project | Network** menu, choose **Network Topology**.
2. Right now, the topology consists of a single network object and a single router object. Notice that the router and network are not connected. To connect the router to the network, click on the Add Interface button, shown here:

3. An interface wizard will appear. From the Subnet menu, select the subnet to attach the router to:
4. When adding an interface to a router, the router will take on the IP address defined in the `gateway_ip` attribute of the selected subnet. Instead, you can specify a different IP in the `IP Address` field. Click on the **Add interface** button to complete the wizard and return to the network topology screen pictured here:

The router is now connected to the network! This means that an instance in the `MySimpleNetwork` network can use the router as its default gateway.

Additional interfaces can be added to the router, but only from other networks. A Neutron router should not be connected to the same network more than once.
Managing routers as an administrator

From the Admin | System | Routers window, administrators can view all routers in the cloud as well as edit and delete selected routers, as shown here:

Here, we can see the previously created router, MySimpleRouter. Clicking on the router name provides a limited subset of the actions that are available through the Project | Network | Routers pane:

Administrators can only delete routers, mark their administrative state UP or DOWN, view static routes, and add or delete interfaces. All other router management functions must be done from within the respective Project panel.
Managing routers with the Neutron client

The Neutron command-line client provides additional functionality not found in the Horizon dashboard, including the ability to specify the type of router to create.

Creating and listing routers

Listing networks with the Neutron client is as easy as using the `neutron router-list` command, shown here:

```bash
[root@allinone ~(keystone_admin)]# neutron router-list
+----+--------+-----------------+-----+---+
| id | name   | external_gateway_info | distributed | ha |
+----+--------+-----------------+-----+---+
| d2a55148-8edd-44fe-8e82-d1260b29bce | MySimpleRouter | null | False | False |
```

Running the `router-list` command as an administrator will return all routers known to Neutron, while running the command as an ordinary user will only return routers associated with the user's tenant or project. As an example, let's authenticate as the demo user and run the same command to see what is returned:

```bash
[root@allinone ~(keystone_demo)]# neutron router-list
```

As expected, no routers were returned. Routers, like other OpenStack resources, are associated with a single tenant or project and can only be viewed or managed by the respective users of those projects or by administrators.

Creating a router

Creating a router with the Neutron client can be accomplished with the `neutron router-create` command. In this example, the demo user is logged in and the router to be created is named MyDemoRouter:

```bash
[root@allinone ~(keystone_demo)]# neutron router-create MyDemoRouter
```
The operation returns a response that can be seen in the following output:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin_state_up</td>
<td>True</td>
</tr>
<tr>
<td>external_gateway_info</td>
<td>dfa617ad-3ded-4962-9484-a5e9ce138172</td>
</tr>
<tr>
<td>id</td>
<td>MyDemoRouter</td>
</tr>
<tr>
<td>name</td>
<td>dfa617ad-3ded-4962-9484-a5e9ce138172</td>
</tr>
<tr>
<td>routes</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>status</td>
<td>b8e0562dab644c87aa693abf48d3040d</td>
</tr>
</tbody>
</table>

Notice that the output did not return information regarding the router type. As an ordinary user, that information is not exposed by Neutron and is only available to users with the admin role.

Copy the id and name values from the output. You will use the router ID when you add an interface to the router or perform certain other network activities. In some cases, the router name can be used in lieu of the ID, but only when the name is unique. In this example, the ID is dfa617ad-3ded-4962-9484-a5e9ce138172, but the value will be unique in your response.

**Adding an interface**

Adding an interface to a router with the Neutron client can be accomplished with the `neutron router-interface-add` command. To add an interface, you must specify the router and subnet name or ID.

Users who consume all available addresses in a subnet may find it necessary to add additional subnets to a network. Neutron allows multiple subnets to be associated with a single network, and routers should be connected to each subnet using the `router-interface-add` command.
If you recall from Chapter 4, Interfacing with Neutron, we created a network and subnet as the demo user, named MyDemoNetwork and MyDemoSubnet, respectively. As the demo user, perform a neutron net-list command to retrieve a list of networks and associated subnets, like this:

```
[root@allinone ~]# neutron net-list
<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>subnets</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyDemoNetwork</td>
<td>0000-00-00-00-00-00-00</td>
<td>192.168.0.0/24</td>
</tr>
</tbody>
</table>
```

Using the neutron router-interface-add command, add an interface to the router and attach it to the MyDemoSubnet subnet:

```
[root@allinone ~]# neutron router-interface-add MyDemoRouter MyDemoSubnet
Added interface 9fffe733-ff0d-4e34-8b55-faf97e2b6e to router MyDemoRouter.
```

That's it! As a result of adding an interface to the router via the API, Neutron created a logical Neutron port for the virtual network interface used by the router, created the virtual interface inside the respective network namespace, attached the interface to the virtual switch or bridge, and configured an IP address on the virtual interface that corresponds with the address defined by the gateway_ip attribute of the subnet. Thanks, Neutron!

**Listing router interfaces**

To obtain a list of the Neutron ports associated with a router, use the neutron router-port-list command, seen here:

```
[root@allinone ~]# neutron router-port-list MyDemoRouter
<table>
<thead>
<tr>
<th>name</th>
<th>mac_address</th>
<th>fixed_ip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000-00-00-00-00-00-00</td>
<td></td>
</tr>
</tbody>
</table>
```

All ports currently associated with the router will be listed in the output. As expected, the fixed IP of the port corresponds to the gateway_ip of the MyDemoSubnet subnet.
Examining the routers

Virtual routers are implemented as network namespaces on one or more nodes. In our single-node environment, the two routers that have been created so far can both be found on the same node. Using the `ip netns` command as `root` or a user with `sudo` privileges, you can list all network namespaces on a host, like so:

```
[root@allinode ~(keystone_admin)]# ip netns
qrouter-dfa617ad-3ded-4962-9484-a5e9ce13b172
qrouter-d2a55148-0edd-44fe-8e82-d12690c29bce
qdhcp-460b2688-02c4-42db-b25c-7e9ba749d368
qdhcp-c8cde907-9a30-4e86-8c31-11d156cb2c
```

The two `qrouter` namespaces in the list correspond to the two routers we've created so far:

```
[root@allinode ~(keystone_admin)]# neutron router-list
+----------+----------+-----------------+----------+-----------+
| id       | name     | external_gateway_info | distributed | ha        |
+----------+----------+-----------------+----------+-----------+
| d2a55148-0edd-44fe-8e82-d12690c29bce | MySimpleRouter | null | False | False |
| dfa617ad-3ded-4962-9484-a5e9ce13b172 | MyDemoRouter | null | False | False |
+----------+----------+-----------------+----------+-----------+
```

Using the `ip netns exec <namespace>` command, you can specify a command to execute within the specified namespace. Useful commands such as `ip`, `netstat`, `ps`, and `iptables` provide details within the scope of the namespace they're executed in.

A quick look at the `MySimpleRouter` network namespace shows the virtual interface created by Neutron when we attached the router to the `MySimpleSubnet` subnet earlier in this chapter:

```
[root@allinode ~(keystone_admin)]# ip netns exec qrouter-d2a55148-0edd-44fe-8e82-d12690c29bce ip a
1: lo <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
14: qr-2e525501-5f: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:05:65:db brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-2e525501-5f
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe25:6d8/64 scope link
        valid_lft forever preferred_lft forever
```

Using the `ovs-vsctl show` command, we can see the `qr-2e525501-5f` interface attached to the integration bridge in a local VLAN mapped to `MySimpleNetwork`:
Additional router interfaces will be connected to the same integration bridge, but they will be tagged with different local VLAN tags and names based on the network, subnet, and port the interfaces are associated with.

Summary

In this chapter, we learned that Neutron routers can route between directly connected tenant networks and external networks using network address translation. Neutron routers can be configured in a redundant or distributed manner, and they trade simplicity in their implementation for high availability. Like the virtual switching infrastructure covered in the previous chapter, users without access to the underlying infrastructure will be unable to observe how Neutron implements virtual routers and their respective features. The logical network diagram provided within the Horizon dashboard, coupled with an understanding of the concepts outlined in this book, will help you understand what is happening behind the scenes.

In the next chapter, we will take a look at some common virtual network architectures that can be built by users and will showcase the traffic flow from client workstations to virtual machine instances using floating IPs.
Where to buy this book
You can buy OpenStack Networking Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.