Discover OpenStack: The Networking component Neutron

John Rhoton

December 04, 2013

This article presents the OpenStack Networking (Neutron) project, explains how it fits into the overall architecture, and shows how it operates. It illustrates Neutron with insight into what it takes to install, configure, and use the components.

This article describes OpenStack Networking, which manages the connectivity between the other OpenStack projects.

It would be possible to develop an elastically scalable workload management system without including any network-specific functionality. Certainly, the compute nodes would need connectivity between them and access to the outside world, but it would be possible to leverage the existing networking infrastructure to allocate IP addresses and relay data between nodes. The biggest problem with such an approach in a multitenant environment is that the network-management system in place would not be able to isolate traffic between users efficiently and securely — a big concern for organizations building both public and private clouds.

One way to address this problem would be for OpenStack to build an elaborate network-management stack that handles all network-related requests. The challenge with this approach is that every implementation is likely to have a unique set of requirements that include integration with a diverse set of other tools and software.

OpenStack has therefore taken the path of creating an abstraction layer, called OpenStack Networking which can accommodate a wide range of plug-ins that handle the integration with other networking services. It provides an application programming interface (API) to cloud tenants with which they can configure flexible policies and build sophisticated networking topologies — for example, to support multitier web applications.

OpenStack Networking enables third parties to write plug-ins that introduce advanced network capabilities, such as L2-in-L3 tunneling and end-to-end quality of service support. They can also create network services, such as load balancing, virtual private networks or firewalls that plug into OpenStack tenant networks.
Historically, the networking components of OpenStack were situated in the OpenStack Nova (Compute) project. Most of these were split into a separate project with the Folsom release. The new project was initially called Quantum but later renamed Neutron to avoid any trademark confusion with the company Quantum Corporation. So, don’t be surprised to see the names Nova, Quantum, and Neutron all appearing in references to OpenStack Networking.

**Model**

The OpenStack Networking API is based on a simple model of virtual network, subnet, and port abstractions to describe networking resources. Network is an isolated layer-2 segment, analogous to a virtual LAN (VLAN) in the physical networking world. More specifically, it is a broadcast domain reserved to the tenant that created it or explicitly configured as shared. The network is also the primary object for the Neutron API. In other words, ports and subnets are always assigned to a specific network.

Subnet is a block of IP version 4 or 6 addresses and their associated configurations. It is an address pool from which OpenStack can assign IP addresses to virtual machines (VMs). Each subnet is specified as a Classless Inter-Domain Routing range and must be associated with a network. Along with the subnet, the tenant can optionally specify a gateway, a list of Domain Name System (DNS) name servers, and a set of host routes. VM instances on this subnet then automatically inherit the configuration.

Port is a virtual switch connection point. A VM instance can attach its network adapter to a virtual network through this port. Upon creation, a port receives a fixed IP address from one of the designated subnets. API users can either request a specific address from the pool or Neutron allocates an available IP address. OpenStack can also define the media access control addresses that the interface should use. After the port has been de-allocated, any allocated IP addresses are released and returned to the address pool.

**Plug-ins**

The original OpenStack Compute network implementation assumed a basic model of performing all isolation through Linux® VLANs and IP tables. OpenStack Networking introduces the concept of a plug-in, which is a back-end implementation of the OpenStack Networking API. A plug-in can use a variety of technologies to implement the logical API requests.

Some OpenStack Networking plug-ins might use basic Linux VLANs and IP tables. These are typically sufficient for small and simple networks, but larger customers are likely to have more sophisticated requirements involving multitiered web applications and internal isolation between multiple private networks. They could require their own IP addressing scheme (which could overlap with addresses that other tenants use)—for example, to allow applications to be migrated to the cloud without changing IP addresses. In these cases, there may be a need for more advanced technologies, such as L2-in-L3 tunneling or OpenFlow.

The plug-in architecture offers a great deal of flexibility for the cloud administrator to customize the network's capabilities. Third parties can supply additional API capabilities through API extensions which may eventually become part of the core OpenStack Networking API.
The Neutron API exposes the virtual network service interface to users and other services, but the actual implementation of these network services resides in a plug-in which provides isolated virtual networks to the tenants and other services like address management. The API network should be reachable by anyone on the Internet and, in fact, may be a subnet of the external network. As I mentioned, the Neutron API exposes a model of network connectivity consisting of networks, subnets, and ports, but it doesn't actually perform the work. The Neutron plug-in is responsible for interacting with the underlying infrastructure so that traffic is routed in accordance with the logical model.

A large and growing number of plug-ins with different features and performance parameters are available. The list currently includes the following plug-ins:

- Open vSwitch
- Cisco UCS/Nexus
- Linux Bridge
- Nicira Network Virtualization Platform
- Ryu OpenFlow Controller
- NEC OpenFlow

The choice of plug-in is up to the cloud administrator, who can assess the options and align them to the specific installation requirements.

**Architecture**

`neutron-server` is the main process of the OpenStack Networking server. It is a Python daemon that relays user requests from the OpenStack Networking API to the configured plug-in. OpenStack Networking also includes three agents that interact with the main Neutron process through the message queue or the standard OpenStack Networking API:

- `neutron-dhcp-agent` provides Dynamic Host Configuration Protocol (DHCP) services to all tenant networks.
- `neutron-l3-agent` does L3/Network Address Translation forwarding to enable external network access for VMs on tenant networks.
- An optional plug-in-specific agent (`neutron-*-agent`) performs local virtual switch configuration on each hypervisor.

It is important to be aware of the interaction between OpenStack Networking and the other OpenStack components. As with other OpenStack projects, OpenStack Dashboard (Horizon) provides a graphical user interface for administrators and tenant users to access functionality — in this case, to create and manage network services. The services also defer to OpenStack Identity (Keystone) for the authentication and authorization of any API request.

The integration with OpenStack Compute (Nova) is more specific. When Nova launches a virtual instance, the service communicates with OpenStack Networking to plug each virtual network interface into a particular port.
Setting it up

The actual installation instructions vary greatly between distributions and OpenStack releases. Generally, they are available as part of the distribution. Nonetheless, the same basic tasks must be completed. This section gives you an idea of what's involved.

System requirements

OpenStack relies on a 64-bit x86 architecture; otherwise, it's designed for commodity hardware, so the minimal system requirements are modest. It is possible to run the entire suite of OpenStack projects on a single system with 8GB of RAM, but for any serious work, it's worthwhile to have a dedicated compute node with at least 8GB of RAM, two 2TB disks, and two Gbit network adapters. It is common to use a controller host to run centralized OpenStack Compute components. In this case, the OpenStack Networking server can run on that same host, but it is equally possible to deploy it on a separate server.

Installation

The installation instructions depend on the distribution and, more specifically, on the package-management utility you select. In many cases, it's necessary to declare the repository. So, for example, in the case of Zypper, you announce to libzypp with zypper ar:

```
# zypper ar -f http://download.opensuse.org/repositories/Cloud:/OpenStack:/Grizzly/SLE_11.SP3/
Cloud:OpenStack:Grizzly.repo
```

For purposes of illustration, below are the primary commands for Ubuntu, Red Hat (Red Hat Enterprise Linux, CentOS, Fedora), and openSUSE:

- **Ubuntu**: Install neutron-server and the client for accessing the API:
  
  ```
  $sudo apt-get install neutron-server python-neutronclient
  ```

  Install the plug-in:
  
  ```
  $sudo apt-get install neutron-plugin-<plugin-name>
  ```

  For example:
  
  ```
  $sudo apt-get install neutron-plugin-openvswitch-agent
  ```

- **Red Hat**: Similar to Ubuntu, you must install both the Neutron server and the plug-in — for example:
  
  ```
  $sudo yum install openstack-neutron
  $sudo yum install openstack-neutron-openvswitch
  ```

- **openSUSE**: Use the following commands:
  
  ```
  $sudo zypper install openstack-neutron
  $sudo zypper install openstack-neutron-openvswitch-agent
  ```

Configuration

Most plug-ins require a database. The Fedora packaging for OpenStack Networking includes server-setup utility scripts that take care of the full installation and configuration of the database:

```
$sudo neutron-server-setup --plugin openvswitch
```
But it is also possible to configure these databases manually. For example, on Ubuntu, you can install the database with the following command:

```
$ sudo apt-get install mysql-server python-mysqldb python-sqlalchemy
```

If a database has already been installed for other OpenStack services, you only need to create a Neutron database:

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

You must specify the database in the plug-in's configuration file. To do so, find the plug-in configuration file in `/etc/neutron/plugins/plugin-name` (for example, `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini`), and set the connection string:

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

### Usage scenarios

A typical OpenStack Networking setup can be complex, with up to four distinct physical networks. A management network is used for internal communication between OpenStack components. A data network handles data communication between instances. The API network exposes all the OpenStack APIs to tenants. In addition, you often need an external network that grants Internet access to the VMs.

On top of these physical networks, there are many ways to configure the virtual networks that the tenants require. The simplest scenario is a single flat network. There may also be multiple flat networks, private per-tenant networks, and a combination of provider and per-tenant routers to manage the traffic between the networks.

To get an idea of how OpenStack Networking might be used in practice, let's go through a simple scenario in which a tenant creates a network, defines a router to forward traffic from the private network, associates a subnet with the network, and launches an instance that is to be associated with the network.

1. Log in to the OpenStack Dashboard as a user with a Member role. In the navigation pane, beneath Manage Network, click **Networks**, and then click **Create Network**.
2. Fill in the network name as well as the first subnet.

**Figure 2. Create a network**

By *subnet*, I mean the network address range (for example, 10.2.0.0/16) and the default gateway.
**Figure 3. Create the subnet**

![Create Subnet Form]

3. You can also configure DHCP and DNS.

**Figure 4. Subnet details**

![Subnet Details Form]

4. As an optional step, create a router by clicking **Routers** beneath Manage Network, and then clicking **Create Router**. You can then connect the interfaces of the router to define how the traffic should flow.
5. The IP address port assignment occurs when you launch an image. Nova contacts Neutron to create a port on the subnet. Every virtual instance automatically receives a private IP address. You can optionally assign public IP addresses to instances by using the OpenStack concept of floating IP addresses.

Figure 6. Manage floating IP associations

6. After the project requests a floating IP address from the pool, it owns the address and is free to disassociate it from that instance and attach it to another.
Figure 7. Access and security

![Access & Security Interface](image)

Conclusion

I hope this short tour gave you a glimpse of the options that OpenStack provides for networking. Keep in mind that OpenStack doesn't actually provide many networking functions: The routing, switching, and name resolution, for example, are handled by the underlying network infrastructure. Rather, the role of OpenStack is to tie the management of these components together and connect them to the compute workloads.

This is the same approach that OpenStack uses for most of its projects, including storage, which is covered in the next article of this series.
Related topics

- Read more OpenStack articles in this series.
- Check out the OpenStack documentation.
- Keep up with OpenStack on Twitter.
- Read about IBM's open cloud architecture.
- Try OpenStack for yourself.
- developerWorks Labs: Experiment with new directions in software development.

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)