NSX Multi-Hypervisor - OpenStack Integration

NSX Multi-Hypervisor Neutron Advanced Features Reference Guide

Document Version: Juno-2014.2

September 2015

Table of Contents

Overview

This document contains recommendations for constructing advanced networking topologies for tenants in cloud installations built on OpenStack Neutron and VMware NSX for Multi-Hypervisor ("NSX-mh"). The intended audience is cloud administrators deploying OpenStack with NSX-mh network virtualization.

Related Documentation

This document assumes the reader is already familiar with NSX-mh and OpenStack concepts. To learn about these topics, please read:

- *NSX User Guide*, version 4.2, including the "NSX Technical Overview" (explains NSX basics and API concepts) and the "NSX Architecture Guide" (describes NSX Controller Cluster, NSX API, and data plane processing done by OVS devices to implement logical networks).
- OpenStack Documentation - http://docs.openstack.org/
- OpenStack networking guide: http://docs.openstack.org/networking-guide/
- OpenStack Neutron Administrator Guide – http://docs.openstack.org/admin-guide-cloud/networking.html
- Openstack Neutron API reference - http://docs.openstack.org/api/openstack-network/2.0/content/index.html
- OpenStack Neutron Plugin Specific Extensions - http://docs.openstack.org/admin-guide-cloud/networking_adv-features.html

Prerequisites

This document assumes the reader is already familiar with certain NSX and OpenStack concepts. To learn more about these topics, please read the *Related Documentation Section*, which provides links to additional information covering the basics of NSX and OpenStack integration.

Supported Hypervisors

The OpenStack Neutron NSX-mh plugin supports the following hypervisors:

- XenServer hypervisor
- KVM based Linux hypervisor
- ESXi 5.5 or later hypervisor

This guide assumes that all hypervisors have been configured according to the NSX setup instructions and that Transport Node objects have been created on the NSX Controller Cluster to represent these hypervisors. This can be done using either the NSX Manager (browser-based user interface) or the NSX API.

Configuration

NSX integration in OpenStack Neutron is achieved by providing a Neutron plugin implementation for NSX-mh. The plugin implementation adheres to the Neutron plugin interface and realized operations via REST API calls to NSX-mh. As such, the plugin driver is configured similar to all other Neutron plugins; via one or more configuration files in the `ini` format.

Enabling the Plugin

To enable the plugin, the `neutron.conf` must be edited to specify the NSX-mh plugin for the `core_plugin` property. The `neutron.conf` typically resides in `/etc/neutron/neutron.conf`, but may reside elsewhere depending on your installation.

For example, edit `neutron.conf` to define the `core_plugin` property as follows to enable the NSX-mh neutron plugin:

```
[DEFAULT]
core_plugin = vmware
```

Note that Neutron has default per-tenant quotas on most resources. To change them, edit the following section of the `neutron.conf` file if necessary:

```
quota_driver = neutron.db.quota_db.DbQuotaDriver
```

After updating the `neutron.conf`, the NSX plugin specific properties should be configured for your NSX installation environment as outlined in subsequent sections.

Configuring the Plugin

NSX plugin specific properties are configured via the plugin's `nsx.ini` file. This file resides under the neutron configuration directory `NEUTRON_CONF_DIR/plugins/vmware/` where `NEUTRON_CONF_DIR` is the configuration directory for OpenStack Neutron (for example `/etc/neutron/plugins/vmware/nsx.ini`).

A complete reference of the plugin's configuration options can be found on

the OpenStack documentation site:
http://docs.openstack.org/juno/config-reference/content/networking-plugin-vmware.html

In addition, the `nsx.ini` file includes comments to describe each of the properties available.

To get started with the plugin you need to minimally provide a default NSX transport zone UUID via the default_tz_uuid property in `nsx.ini`. We recommend that users create a dedicated transport zone for Openstack. From the NSX Network Manager console, select Network Components, Transport Layer, Transport Zones from the dropdown list.

The value for `default_tz_uuid` is from NSX, and can be retrieved using NSX API or NSX Manager. This flag represents the NSX Transport Zone that will be used to create tenant-specific private networks.

The value for default_l3_gw_service_uuid denotes an NSX L3 Gateway Service that should be used for creating Logical L3 routers. Note that the NSX Gateway must use the same transport connector type as specified by the default_transport_type setting in the `nsx.ini` file. The default transport type is `stt`, other available options are `gre`, `ipsec_gre`, `ipsec_stt`, and `bridge`.

You'll also need to provide values for the `nsx_user`, `nsx_password` and `nsx_controllers` properties within the configuration file. These values define the NSX controller(s) and credentials the plugin should use when creating connections to NSX.

If you're editing this file while the Neutron service is already running, you must restart the Neutron service for the configuration changes to take effect (note this may cause service interruption for active neutron users).

For example on Ubuntu:

```
$ sudo service neutron-server restart
```

Verifying the Configuration

The NSX Neutron plugin includes the command `neutron-check-nsx-config` which allows you to verify the NSX configuration for Neutron.

For example:

```
$ neutron-check-nsx-config /etc/neutron/plugins/vmware/nsx.ini
```

All configuration check steps should indicate `PASS` to indicate the integrity of

basic Neutron configuration.

Neutron DHCP Agent Requirements

If you are planning to run the Neutron DHCP agent to provide DHCP services to your virtual machines, ensure that Open vSwitch is installed on every system which is running the DHCP agent. Each host running the DHCP agent (and Open vSwitch) should also be added to NSX as a transport node.

A sample DHCP agent configuration snippet is shown below. Note that the DHCP agent's configuration file is typically located at `NEUTRON_CONF_DIR/dhcp_agent.ini` where `NEUTRON_CONF_DIR` the configuration directory for neutron. This is typically `/etc/neutron/` on many installations.

Sample `dhcp_agent.ini` snippet:

```
[DEFAULT]
enable_metadata_network = True
enable_isolated_metadata = True
ovs_use_veth = True
use_namespaces = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
```

Neutron Metadata Agent Requirements

A sample metadata agent configuration snippet is shown below. Note that the metadata agent's configuration file is usually located at `NEUTRON_CONF_DIR/metadata_agent.ini` where `NEUTRON_CONF_DIR` is the configuration directory for neutron. This is typically `/etc/neutron/` on many installations.

```
[DEFAULT]
auth_url = http://<keystone-host>:35357/v2.0
auth_region = <region>
admin_tenant_name = <tenant_name>
admin_user = <user>
admin_password = <password>
# IP address used by Nova metadata server
nova_metadata_ip = <ip of nova-metadata-server>

# TCP Port used by Nova metadata server
nova_metadata_port = 8775
```

OpenStack Nova Requirements

OpenStack nova must be configured to leverage Neutron from a networking perspective. In particular the `nova.conf` must be updated on each node that runs any nova service (e.g. `nova-api`, `nova-scheduler`, `nova-compute`, etc.). The default location for nova.conf is `NOVA_CONF_DIR/nova.conf`, typically `/etc/nova/nova.conf` on many installations.

The `nova.conf` snippet below illustrates how to setup neutron as the networking service for nova:

```
[DEFAULT]
network_api_class=nova.network.neutronv2.api.API
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron

[neutron]
admin_username=<username>
admin_password=<password>
admin_auth_url=http://<keystone host>:35357/v2.0
auth_strategy=keystone
admin_tenant_name=<tenant-name>
url=http://<neutron-server>:9696
region_name=<region-name>
service_metadata_proxy = True


[libvirt]
# the below only required if using libvirt hypervisor
vif_driver=nova.virt.libvirt.vif.LibvirtGenericVIFDriver
```

After updating `nova.conf` you'll need to restart the nova services for changes to take effect.

OpenStack Nova Requirements for XenServer

If you are using XenServer as your Hypervisor, ensure the `nova.conf` on each node running the nova-compute service has the following properties set:

```
xenapi_ovs_integration_bridge=xapi1
xenapi_vif_driver=nova.virt.xenapi.vif.XenAPIOpenVswitchDriver
compute_driver=xenapi.XenAPIDriver
xenapi_connection_url=https://<Xen Server IP>
xenapi_connection_username=root
xenapi_connection_password=<XenServer Root Password>
```

Note: As part of configuring your XenServer for use with NSX, you will have created an "NSX Integration Bridge". The OVS bridge name (by default, it's `br-int`) of this bridge must be specified using the `--xenapi_ovs_integration_bridge` flag. The above flag file defaults this value to `xapi1`, but this value can be different depending on your deployment configuration. You can run the command `xe network-list` to retrieve the list of networks and associated bridges.

OpenStack Nova Requirements for KVM

If you are using KVM as your Hypervisor, ensure the `nova.conf` on each node running the nova-compute service has the following properties set:

```
[libvirt]
ovs_bridge=<integration bridge used by NSX>
virt_type=<libvirt type, e.g. xen>
vif_driver=nova.virt.libvirt.vif.LibvirtGenericVIFDriver
```

Note: As part of configuring your KVM Server for use with NSX, you will have created an "NSX Integration Bridge". The OVS bridge name of this bridge must be specified using the `--libvirt_ovs_bridge` flag. This value can be different depending on your deployment configuration. This value defaults to `br-int` and therefore the `--if-not-exists` flag can be used to create the bridge only if needed.

OpenStack Nova Requirements for VMware ESXi 5.5

If you are using VMware ESXi 5.5 as your Hypervisor, the following prerequisites must be met:

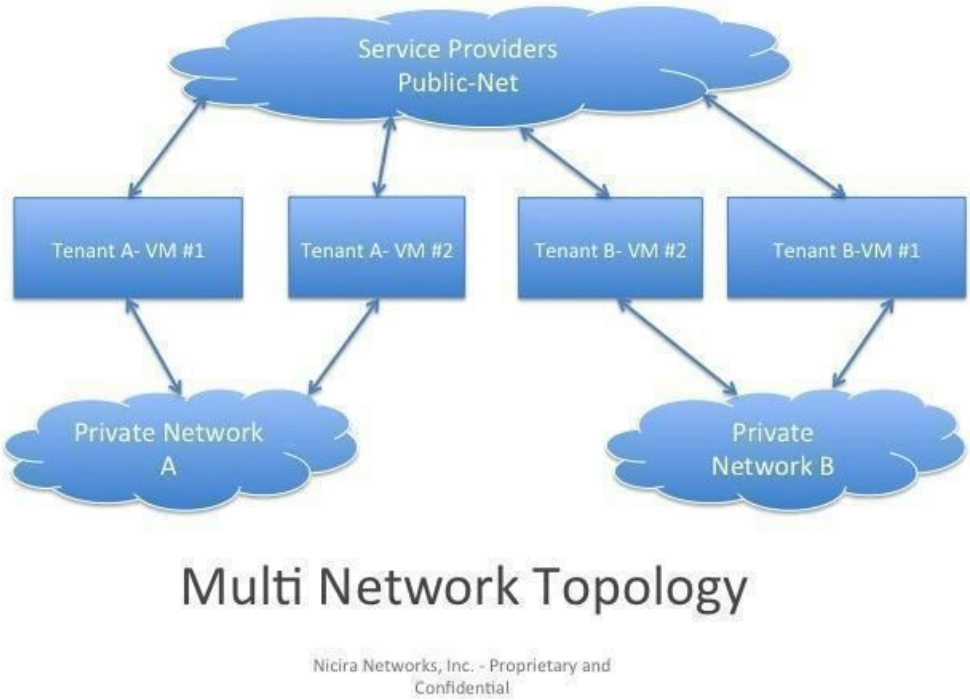
- NSX vSwitch must be installed and configured on each VMware ESXi 5.5 hypervisor node. (For details on installing and configuring NSX vSwitch for ESXi 5.5, please refer to the *NSX User Guide*.)
- On each ESXi host, enable the `allow-custom-vifs` mode in NSX vSwitch to ensure the NSX Controller will be given the OpenStack-generated UUIDs (not the vSphere-generated UUIDs) of all the VMs and VIFs it handles. You must do this before you power-on the VMs. At the ESXi host command prompt, type `nsxcli --allow-custom-vifs`. See the *NSX User Guide* for details.
- Configure the Nova Compute Service to run the VMware vCenter Driver (also known as the "VC Driver") if using ESXi.

Once NSX vSwitch has been installed and configured on all ESXi hypervisors, there is no further configuration required on each ESXi hypervisor / vSphere Cluster or Nova Compute node because the logic used to bind virtual machine interfaces to the NSX vSwitch is part of the native VC Driver.

Configuring VM Network Connectivity

This section focuses on configuring multiple networks and multiple virtual network Interfaces (vNICs) in OpenStack Nova Compute. We will create a network topology where each VM has two NICs: one on a "public-net" shared by all tenants and one on a private network specific to that tenant.

Multi-network topology with Logical Switches



In the above sample diagram, there is global network named "public-net" that is shared by all tenants. Public-net is implemented by a (non-external) Neutron network, which has been pre-provisioned by the service provider using either the Neutron CLI or API. The topology also contains a private network for every tenant.

If there were two projects (`tenantA` and `tenantB`), we could accomplish the above topology by creating three Neutron networks as follows:

```
$ neutron net-create public-net --shared
--provider:network_type=vlan
--provider:physical_network=af93bd8a-0786-4b3c-a037-c98bcc4c7827
--provider:segmentation_id=123
$ neutron subnet-create public-net 11.0.0.0/22
$ neutron net-create tenantA-private --tenant-id=<tenantA_id>
$ neutron subnet-create tenantA-private 10.0.1.0/24
$ neutron net-create tenantB-private --tenant-id=<tenantB_id>
$ neutron subnet-create tenantB-private 10.0.1.0/24
```

Note that we are assuming that these commands have been executed with administrator credentials. Otherwise, Neutron will allow for creating networks on behalf of another tenant. When `--tenant-id` is not specified, the tenant to be associated with the network is derived from the

authorization token sent in the Neutron request (`X-Auth-Token`)header.

The commands described above create the three networks that are illustrated in the above topology diagram and a subnet on each of them. The Neutron NSX plugin will use STT overlay networks for `private-tenantA` and `private-tenantB` networks, whereas the `public-net` network will be implemented on the transport zone `af93bd8a-0786-4b3c-a037-c98bcc4c7827` using a bridged connector mapped to VLAN `123`.

Verifying your Network Topology

At this point, if everything is set up correctly, spawning a new instance for each tenant should result in two virtual machines with 2 virtual interfaces each. They must be attached to 'public-net' and a tenant specific private network. This can be verified in the following ways:

On Neutron:

```
$ neutron port-list --<network_id> --device_id=<vm_id>
```

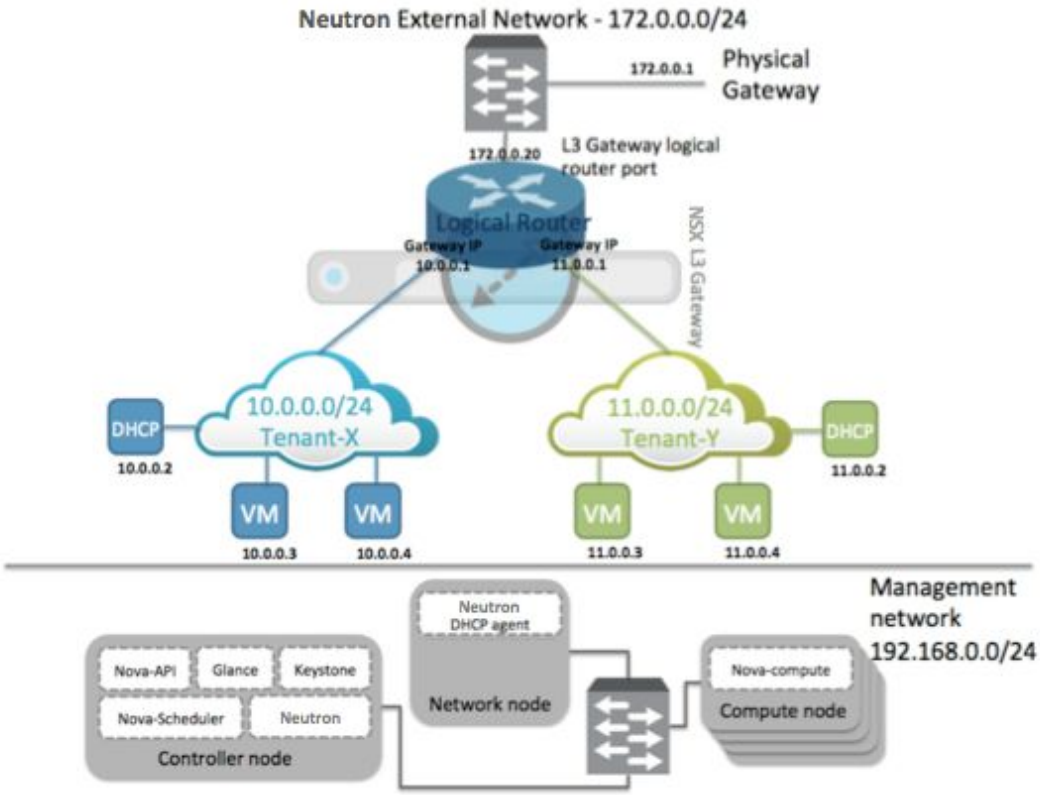
The above command should return two ports. For one port, the network ID should correspond to the public network, whereas for the other port, it should correspond to the private network.

On NSX:

Log in to the NSX Manager and check the logical network configuration. It should show the following:

1. A "Logical Switch" for each Neutron Network, with a tag that represents the OpenStack tenant identifier; the scope of this tag is `q_id`.
2. The transport connector for the private logical switches should map to the default transport zone specified in `nsx.ini`, and the mapping should be `STT`.
3. The transport connector for the public logical switch should map to the transport zone specified when creating the network, and the mapping should be of type `BRIDGE`.
4. On each private network logical switch there should be two ports; one for the DHCP server distributing addresses to the subnet, and one for the private interface of the VM instance.
5. You can further verify that port security and security profile settings on the port are consistent with the default settings; the default security profile should be associated with the port and port security should be enforced on both MAC and IP addresses (except the DHCP port).
6. On the public network logical switch, there should be three ports: one for the DHCP server, and two for the VMs that were created.

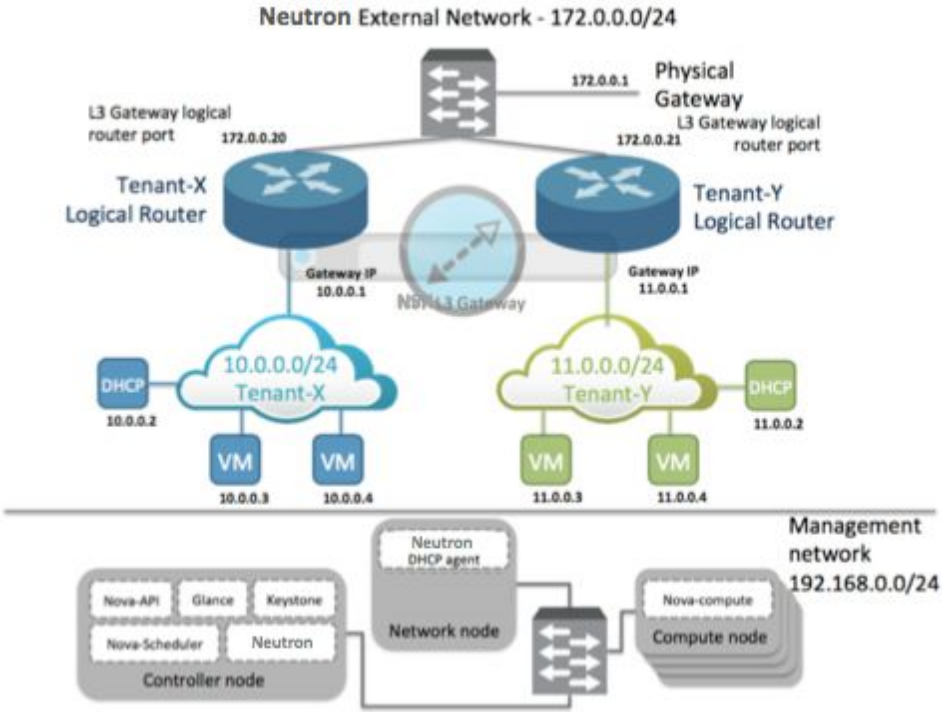
Topology with provider-supplied logical router



NSX L3 gateway integration with a centralized logical router and metadata services

As shown in the diagram above, the Neutron logical router leverages the NSX L3 Gateway integration, and provides connectivity to internal networks via the 'x.x.x.1' gateway address that is assigned to Logical router port attached to a global Logical router. In order to implement this topology, or any other topology where a logical router is shared across tenants, the logical router and its interfaces should be created with admin credentials.

Topology with per-tenant logical router



NSX L3 Gateway integration with per-tenant router and MetaData Services

In the diagram above, multiple Neutron logical routers are using the same NSX L3 Gateway service. In this case, logical routers are owned by the tenant themselves. One or more tenant networks can be attached to these logical routers.

Both tenant logical routers are connected to the same Neutron external network. According to the default policy configuration, external networks can be created by admin users only; however they're visible by every user and can used for setting external gateways and allocating floating IPs.

Creating NSX Bridged Networks with Neutron

NSX supports the creation of logical networks that are directly mapped to a physical network. This section explains how to use NSX bridged logical networks with OpenStack.

Bridged Networks with Neutron

By default, all Neutron networks are `STT` tunneled overlay logical networks. The provider networks API extension can be used for creating a bridged logical network. The provider networks extension augments the network resource by adding the following parameters:

- `provider:network_type`
- `provider:physical_network`
- `provider:segmentation_id`

In order to create a bridged logical network, `network_type` should be set to either `flat` or `vlan`. `physical_network` should be set to the uuid of an existing transport zone in NSX; finally, `segmentation_id` should either be set to a valid vlan identifier, or omitted if network_type is `flat`.

Note that the transport zone uuid specified in the `physical_network` parameter is not validated. If an invalid uuid is supplied the operation will fail with a 500 HTTP error.

For further information on the provider networks extension, please refer to the Neutron API reference guide [1] and the Neutron admin guide[2].

The following examples will create a bridged logical network with flat and VLAN network types:

```
$ neutron net-create flat_net --provider:network_type=flat
--provider:physical_network=d7b177dc-5310-4652-b1bb-49dc5f7fca4e

$ neutron net-create vlan_net --provider:network_type=vlan
--provider:physical_network=d7b177dc-5310-4652-b1bb-49dc5f7fca4e
--provider:segmentation_id=123
```

Controlling how bridged networks are managed in the NSX backend

When the network is deleted, Neutron will remove all the logical switches created for that network.

The maximum number of ports for each bridged logical switch is controlled by the `max_lp_per_bridged_ls` configuration parameter.

NSX L2 Gateways with Neutron

NSX Gateways allow for extending external L2 Networks into logical networks managed by Neutron. A special use case of an NSX Gateway is enabling Data Center Interconnect across 2 disjoint NSX-managed OpenStack deployments, residing in the same or separate data centers. The Neutron API network gateway extension allows for managing network gateways backed by NSX L2 Gateway nodes, as well as connecting and disconnecting Neutron

[1] http://docs.openstack.org/api/openstack-network/2.0/content/
[2] http://docs.openstack.org/grizzly/openstack-network/admin/content/

networks to or from them.

Tenant-provided network gateway

For this use case, we assume that the tenant will create their own network gateway devices (transport nodes) and all needed elements to connect VMs to the external network. Once that's done, the following high level steps implement the Neutron logic for the L2 connection.

1. Create Neutron network gateway devices.
2. Create Neutron network gateway service, and specify the gateway devices it should use.
3. Create a Neutron network (logical switch).
4. Connect the gateway to the network. Optionally, a VLAN can be specified to bind the Neutron network to a specific physical network on the external side of the network gateway.

If your cloud administrator allows you to create the gateway device transport nodes in NSX, then you can create one now and establish the management connections between the controller and the gateway node. Note that the connection will appear as down if the gateway node has not been connected to the right controllers using its own CLI.

Create the network gateway devices (transport nodes)

VMs connect to the physical network via gateway devices. Here we register an already installed NSX Gateway as a gateway device for use in the Neutron installation.

```
$ neutron gateway-device-create --connector-type <connector-type>
--connector-ip <connector-ip> --client-certificate-file
<path-to-cert> <gw-name>
```

This command creates an NSX gateway node, and populates it with information concerning authentication and transport network connection. The operation returns a "gateway device" resource. It is important to note that this is a Neutron construct, not an NSX one. Therefore, its identifier is not the identifier of the gateway node in the NSX backend.

The CLI command allows for specifying:

- `gw-name` - A descriptive name of the resource
- `connector-type` - Specifies the transport type the gateway node should use to connect to the overlay network. The default is `stt`. Other available options are `gre`, `ipsecgre`, `ipsecstt`, and `bridge`.
- `connector-ip` - Represents the IP address the gateway device will use To connect to the overlay network. This should be the IP address of one of the gateway's physical interfaces.
- `client-certificate-file` - The `.pem` file the gateway will use to

authenticate itself with the NSX controller.

Note that the CLI also allows passing in the client certificate as a string directly on the command. This option is not shown here, but can be found in the CLI help text and online OpenStack documentation.

Create the network gateway

A gateway service allows VMs to be connected at Layer 2 (L2) to an external network, even if the hypervisor running the VM is not physically connected to the L2 network.

```
$ neutron net-gateway-create <gateway_name>  --device
id=<gateway_device_id>,interface_name=<nsx_gateway_interface_name>
NAME
```

The `--device` option identifies the gateway device created in the previous step. It is important to note that, due to the implementation of the Neutron CLI, you must specify the `--device` option before the name of network gateway to be created.

Note that when an L2 gateway service is created, the user can specify one or more devices to be used for that gateway. The `net-gateway-create` operation will create a gateway service using the nodes specified on the command line. It is not possible to reuse the same node for multiple instances of Neutron network gateways.

The `interface_name` parameter optionally allows for specifying an interface that will be used for bridging traffic. For instance, in this case `breth2` will be the 'outside' looking interface of the gateway node, and traffic will be bridged from it to the STT overlay network where the gateway node is connected via its 10.0.0.1 interface. The default value for the `interface_name` parameter is `breth0` and can be specified in the `nsx.ini` file.

The `--device` option allows for specifying:

- UUID of a Neutron gateway device.
- The name of an interface on the transport node to use (e.g.: `breth0`, `breth1`, …). If no interface is supplied the value of the `default_interface_name` configuration flag will be used. The default value for this flag is `breth0`; the flag can be specified in `nsx.ini`. The name specified here will be the name of the gateway service on the NSX backend.

Note that multiple devices can be specified by repeating the `--device` option, in order to be able to create gateway services with multiple devices for redundancy.

A Neutron network gateway is backed by a NSX Layer-2 gateway service,

which is built with the NSX gateway transport nodes backing the gateway devices composing the Neutron network gateway. While it is possible to specify an empty list of devices, the resulting network gateway won't be functional.

Create a network, this creates a logical switch in NSX

A logical switch provides a standard L2 Ethernet service-model and contains logical switch ports that can be configured to implement various security and QoS policies and exposes port counters for metering or debugging.

```
$ neutron net-create <net-name>
```

Connect the network to the NSX L2 gateway

Now, connect an NSX L2 Gateway to a specific network.

```
$ neutron net-gateway-connect <gateway_id> <network_id>
--segmentation-type=vlan --segmentation-id=<vlan_id>
```

The `neutron net-gateway-connect` option allows for specifying:

- A Neutron network gateway UUID
- A Neutron network UUID
- *`--segmentation-type=vlan --segmentation-id=<vlan_id>`*, tells the controller that you want this l2-gateway to terminate on a VLAN and the VLAN ID.

The current release of the Neutron NSX plugin does not allow for updating the set of devices associated with a neutron network gateway. The `net-gateway-update` operation can update the name of the network gateway only.

When a Neutron network is connected to a gateway, a new port is added to the network itself. As this port represents a layer-2 connection, no IP address is allocated for it. In order to distinguish this port from regular Neutron ports, the `device_id` and `device_owner` attributes are set to the following values:

- `device_id: <network gateway id>`
- `device_owner: 'network:gateway_interface'`

This port is automatically removed when the network is disconnected from the gateway. This is achieved with the following command:

```
$ neutron net-gateway-disconnect <gateway_id> <network_id>
--segmentation-type={flat|vlan} --segmentation-id=<vlan_id>
```

Attempts to delete a network gateway which still has an active network

connection will result in a failure.

Gateways supplied by the service provider

For this workflow, we assume that a default layer-2 gateway service has been configured in `nsx.ini` (see example in the appendix). In order to enable a default gateway service, the `default_l2_gw_service_uuid` parameter should be specified in the `[DEFAULT]` section. This should correspond to the uuid of a L2 gateway service on the NSX platform.

Note: The plugin does not validate the l2 gateway service UUID specified in `nsx.ini`. If an invalid value is specified, a runtime error will be thrown when connecting/disconnecting networks.

Connecting a network to the default gateway:

1. User creates a Neutron network. Alternatively, the service provider can create the network on behalf of the user using the user's tenant identifier.
2. User retrieves the identifier of the network gateway to use through either Neutron API or CLI; the default gateway has a predefined name, '`default L2 gateway service`' which can be used in neutron CLI commands.
3. Service provider connects the gateway to the network using the Neutron API or CLI. Optionally, a VLAN can be specified to bind the Neutron network to a specific physical network on the external side of the network gateway.

Example:

```
$ neutron net-create test-net
$ neutron net-gateway-connect <default_gateway_id> <network_id>
--segmentation-type=vlan --segmentation-id=<vlan_id>
```

Alternatively, `segmentation-type=flat` could be used; in that case the `segmentation-id` attribute should not be specified

It is worth noting that the default gateway service is owned by the service provider, and therefore regular tenants cannot see it or connect networks to it. Indeed, in this case it is the service provider that must connect the user's network to the gateway, as specified previously in this section.

Enabling NSX QoS with Neutron

Neutron provides cloud operators with a tool for managing QoS parameters for logical ports and logical switches within their networks. In the following section, we outline some of the basic concepts of Neutron QoS, as well as how to configure it.

Concepts

- QoS Pool: This represents a base guaranteed rate profile, represented in kbps, that can be associated with a particular logical network. Once a QoS Pool is associated with a logical network, every Logical Port on that particular logical network is guaranteed the QoS rate as specified by the QoS pool.
- rxtx_factor (integer value): This is a multiplying factor associated with a particular logical port. The 'rxtx_factor' is also associated with a Nova 'flavor'. When Nova launches a VM of a particular flavor that has a particular rxtx_factor configured, the multiplier is used to size the QoS parameters to be associated with the Neutron port(s) created for that VM.
- Net rate: The resulting QoS rate that an OpenStack VM receives is base rate (represented by QoS Pool) times 'rxtx_factor' for a particular VM.

Example:

1. Network A is configured with a QoS Pool allowing 1024 kbps per port.
2. Port A on Network A contains a vNIC that is of a particular flavor with 'rxtx_factor' of 2.
3. The net QoS rate associated with Port A equals 1024 x 2 = 2048 kbps.

Configuring QoS with Neutron and Nova

Neutron Configuration

1. Create a QoS Pool using Neutron CLI:

```
$ neutron queue-create <queue_name> --min <min_rate> --max
<max_rate> --qos_marking <{trusted|untrusted}> --dscp <dscp_value>
--default <{True|False}>
```

Example:

```
$ neutron queue-create net1queue --min 0 --max 1024
```

2. Associate a QoS Pool with a Network:

```
$ neutron net-create net1 --queue_id <queue_id>
```

Or if the network already exists:

```
$ neutron net-update <net_id> --queue_id <queue_id>
```

Nova Configuration

1. Create a Nova flavor that contains `rxtx_factor` associated with it:

```
$ nova-manage flavor create --name=NewFlavor –memory=2048 –cpu=2
–local_gb=5 –rxtx_factor=2 --swap=1024 –flavor=8
```

When booting an instance is associated with the newly created flavor, the logical port that the VM's vNIC is attached to will have an associated `rxtx_factor` of `2`. If this particular logical network contains a qos_pool of 512 kbps, then the effective QoS rate associated with this new port will be 512 x 2 = 1024 kbps.

The QoS pool (or queue) associated with the network should be considered as a "template" for pools to be associated to the network's ports. Therefore:

- Changing the QoS queue settings on the network won't affect any existing port, but will apply to every port created afterwards;
- Neutron will have a distinct queue for each port, in addition to the queue associated with the network.

Configuring Default QoS with Neutron

The default QoS Pool feature allows you to create a QoS Pool that will be associated with every logical port that will be created using the Neutron API. This process removes the necessity to manually associate a QoS pool with a Neutron network. It's important to note that if a neutron network is associated with a QoS pool and a default queue exists the QoS pool associated with the network takes precedence.

Note: Only users with administrative right are allowed to manipulate the default queue.

Neutron Configuration

1. Create a default QoS Pool using Neutron CLI:

```
$ neutron queue-create <queuename> --min <min_rate> --max
<max_rate> --qos_marking <{trusted|untrusted}> --dscp <dscp_value>
--default True
```

For example:

```
$ neutron queue-create default_queue --min 0 --max 1024  --default
True
```

2. Delete a default QoS Pool with a network:

```
$ neutron queue-delete <queue_id>
```

Port Security

The Port Security mechanism prevents a VM from spoofing its MAC or IP address. Port security is enabled at the port level via an extended attribute of the Port resource in Neutron. When a port is created, by default, port security is always enabled[3].

In order to disable port security one needs to pass `--port_security_enabled=False` on the CLI when creating the port. Note that when `port_security_enabled=True` (the default setting) only packets that match the mac and ip addresses associated the port are allowed to traverse it.

Note: a prerequisite of enabling `port_security_enabled` is that there must be an IP address associated with the port in neutron, otherwise the API will return a 409 Conflict HTTP error.

Port security can also be enabled/disabled via the neutron CLI shown below.

Create a port with port security enabled:

```
$ neutron port-create <net_id> --port_security_enabled=True
```

Add port security to a port:

```
$ neutron port-update <port_id> --port_security_enabled=True
```

Remove port security from a port:

```
$ neutron port-update <port_id> --port_security_enabled=False
```

Security Groups using Neutron API and NSX

The NSX Plugin allows for configuring security groups either through Neutron and/or Nova APIs.

From a functional perspective, the only difference between the two modes is that with the Neutron API for security groups, one can also control egress filtering from the VM. We recommend using Neutron security groups directly.

However, because of integration with existing tools, such as Horizon, it might be necessary or preferable managing security groups through the Nova API.

[3] Unless the port is created via Nova. In that case port security is always enabled as it is required for enforcing the default security group.

The following setting must be applied to `nova.conf` (usually in `/etc/nova`) on all of your hosts running any Nova service:

```
security_group_api = neutron
```

For examples of how to use Nova security groups, see the Nova security group documentation at the following address:
http://docs.openstack.org/openstack-ops/content/security_groups.html

Managing security groups and rules

Create a security group for "web servers"

```
$ neutron security-group-create webservers --description "security
group for webservers"
```

Viewing Security Groups:

```
$ neutron security-group-list
```

Creating Security Group Rule to allow port 80 ingress

```
$ neutron security-group-rule-create --direction ingress --protocol
tcp --port_ra-ge_mi- 80 --port_ra-ge_ma- 80 <security_group_uuid>
```

List Security Group Rules

```
$ neutron security-group-rule-list
```

Delete security group rule

```
$ neutron security-group-rule-delete <security_group_rule_uuid>
```

Delete security group

```
$ neutron security-group-delete  <security_group_uuid>
```

Working with ports and security groups

Create a port associated with security group

```
$ neutron port-create <network_id> --security_groups list=true
<security_group1_id> <security_group2_id> --port_security=mac_ip
```

Remove security groups from a port

```
$ neutron port-update <port_id> --security_groups=None
```

Boot VM on "webservers" security group:

```
$ nova boot --image <image> --flavor <flavor> --nic
net-id=<network-uuid> MyVM --security_groups webservers
```

Launching a VM without a security group

You can launch a VM without a security group by following these steps:

First, create a port in Neutron without a security group:

```
$ neutron port-create --no-security-groups <network>
```

Then pass that port to Nova:

```
$ nova boot --nic port-id=<port-id> ...
```

Please note that pre-Havana versions of Nova would automatically add a security group to ports passed in via the API, but since the Havana release, Nova no longer does this.

Openstack L3 APIs using NSX L3 Gateway

Neutron provides an API extension for managing Layer 3 forwarding, source NAT, as well as Floating IPs.

VMware NSX users can use these APIs in order to leverage NSX L3 Gateway functionality for enabling a fault-tolerant and scalable Logical L3 model through the VMware NSX-mh Neutron Plugin.

The VMware NSX Plugin for Neutron supports all the features of the Neutron L3 API extension.

This section discusses how to use the L3 extension with the NSX plugin. For more information concerning the API specification, please refer to Neutron API reference and administrator guides.

Prerequisites

1. Add an NSX Gateway transport node and create a Gateway Service using NSX Manager. The UUID of this gateway service must be stored in `nsx.ini` via the `default_l3_gw_service_uuid` property. Please note that the NSX plugin does not validate the correctness of the specified id against the NSX controller. Setting an invalid value will result in an error while performing L3 operations.
2. Configure physical network infrastructure that we will be NATing private networks to and gather the CIDR block and next-hop IP.

3. Gather information around the CIDR block that will be used to create the floating IP "pool" out of which virtual machines will be assigned external network IP addresses.

Managing Routers and SNAT

Creating a router

```
$ neutron router-create <router_name>
```

This operation results in the creation of a logical router in the NSX platform. Although the router has a logical port with a L3 Gateway Attachment, no Source NAT rules are configured, and no IP address is configured on the gateway port.

By default the Neutron API creates a "centralized" router. It is however also possible to create "distributed" routers, thus leveraging NSX's distributed logical router capabilities.

To this aim, it is sufficient to set the `--distributed` flag to true when creating a logical router in Neutron:

```
$ neutron router-create <router_name> --distributed True
```

Note that the default router type can be specified in `neutron.conf` via the `router_distributed` property. See the neutron configuration reference for additional details.

However, Neutron ships with a restriction which allows only admin users to create distributed routers. When using the NSX plugin there is no reason to keep this restriction, that can be lifted amending the following line in `/etc/neutron/policy.json`:

```
"create_router:distributed": "rule:admin_only"
```

into:

```
"create_router:distributed": "rule:admin_or_owner"
```

The VMware neutron plugins package also provides json policy files specialized for NSX features. These files also have policies for NSX distributed logical routers. These policies are installed by default in `/etc/neutron/plugins/vmware/policy`.

However, the router policies contained in routers.json won't override the default ones in `policy.json`, which should be manually removed.

Attaching a Neutron subnet to a router

```
$ neutron router-interface-add <router_name|router_id>
<subnet_name|subnet_id>
```

This operation connects a neutron subnet to a router. The subnet's gateway IP (by default the first address in the CIDR for the subnet), will be assigned to the router interface. With this operation, a port is also added to the neutron network.

In order to distinguish this port from regular neutron ports, the `device_id` and `device_owner` flags are set to the following values:

- `device_id: <router id>`
- `device_owner: 'network:router_interface'`

On the NSX side, this operation adds a logical port both on the logical router and on the logical switch corresponding to the subnet's network. A patch attachment will connect these two logical ports. The subnet's gateway ip address will be configured on the router's logical port.

If an external gateway is already configured for the router (see below), a Source NAT rule will be added for the subnet's CIDR. The order of such rule is determined as follows:

- 255 - CIDR_PREFIX_LEN. For instance the SNAT rule order for the 10.0.0.0/24 subnet would be 231 (255 - 24)

This operation also generates a "Source No Nat" rule in the NSX backend for the subnet being connected. This rule will prevent east-west traffic[4] from being NATted.

Detaching a subnet from a router

```
$ neutron router-interface-delete <router_name|router_id>
<subnet_name|subnet_id>
```

This operation removes the association between a Neutron subnet and a router. The corresponding logical ports and NAT rules on the NSX platform are removed as well.

Configuring an external gateway on a router (SNAT)

```
$ neutron network-create <name> --router:external=True
```

[4] Traffic directed between private Neutron networks connected to the same router

```
$ neutron subnet-create <ext_net_id|ext_net_name> <external_cidr>
--enable_dhcp=False
$ neutron router-gateway-set <router_id|router_name> <ext_net_id>
```

This operation enables source NAT on the logical router. SNAT rules will be created for every subnet connected to the router.

The 'external network' abstracts the physical network beyond the NSX logical router. For this reason:

- The gateway ip address for the subnet defined on the external network should correspond to the ip address of the physical network's gateway.
- The `allocation_pools` for this subnet should be configured appropriately as floating IPs will be selected from this pools. By default all the IPs in the CIDR, with the exception of the gateway IP and the DHCP server (if enabled), are available in the allocation pool. For more information about configuration of allocation pools please refer to the Neutron API reference.
- It is advisable to keep dhcp disabled for external networks.

Note: Unlike regular Neutron networks, external networks are not backed by a NSX logical switch. When an external network is created via the Neutron API no corresponding configuration is performed on the NSX cluster. As a consequence, the scenario in which instances are deployed directly on public facing networks (ie: without requiring a floating IP for external access), should be implemented using the provider networks extension (see Section 4 in this document for more information)

On the NSX platform, the `router_gateway_set` operation will:

- Update the router's gateway address to the external subnet's `gateway_ip` attribute
- Allocate an address from the external subnet's allocation pool and assign it to the NSX logical router's L3 GW port.

Managing Floating IPs

Prerequisites

As stated previously in this section, floating IPs are allocated from an external network. It is important that:

- The external network is configured as an external gateway for a router.
- The router has an interface on each internal logical network where the ports to be give external access through Floating IPs are located.

Create and associate a Floating IP

```
$ neutron floatingip-create <ext_net_id|ext_net_name>
```

This operation allocates a floating IP from the allocation pool for the external network. When a floating IP is created in this way, it is not associated and no configuration is performed on NSX.

```
$ neutron floatingip-create <ext_net_id|ext_net_name> --port_id
<port_id>
```

Note: the parameter `--port_id` uses an underscore here. Unlike many Neutron parameters, the syntax with the the dash (`--port-id`) won't work in this case.

This operation instead creates a floating IP and associates it to a Neutron port. It is important that the port should belong to a network connected to the router for which ext_net represents the external gateway network.

The association implies that on the NSX platform:

- The floating IP address is added to the L3 gateway router logical port.
- Source and Destination NAT rules for forwarding traffic to and from the internal neutron port are added to the NSX logical router.

Associate a previously unassociated floating IP

```
$ neutron floatingip-associate <floatingip_id> <port_id>
```

This operation performs the association of an already existing floating IP to a Neutron port.

Disassociate a floating IP

```
$ neutron floatingip-disassociate <floatingip_id>
```

This operation removes the floating IP address from the logical router port configuration, and removes source and destination NAT rules for the floating IP.

Sample workflow

We can test out a sample workflow to create a VM and assign it a floating IP using the workflow described below. Please note that we are assuming that:

1. A layer-3 gateway service has been created in NSX, and its uuid specified in `nsx.ini`
2. A neutron router has been configured and associated with an external network and at least one internal network.

Boot a VM with specified image id, flavor and name.

```
$ nova boot --image <img_id> --flavor 1 <instance_name>
```

Allocate a Floating IP. This command will create a DNAT rule on NSX's Logical router allowing the VM external network connectivity.

```
$ neutron floatingip-create <external_network_id>
```

Query Neutron to get the id of the port to be associated with the floating IP.

```
$ neutron port-list --device_id=<vm_id>
```

Associate an allocated floating IP with a particular VM on port `<port_id>`.

```
$ neutron floatingip-associate <floating_ip_id> <port_id>
```

Detach a Floating IP from a VM.

```
$ neutron floatingip-disassociate <floatingip_id>
```

Delete DNAT rule on the NSX Logical router, which allows external network connectivity.

```
$ neutron floatingip-delete <floatingip_id>
```

Managing multiple NSX Gateway appliances

By default, logical routers are implemented by the default layer-3 gateway service specified in `nsx.ini`. It is possible to use several gateway services, and allow tenants to choose which gateway service should be used by their routers.

This is achieved using the *provider networks* extension, associating external networks with L3 gateway service. To do so the following parameters should be specified:

- `provider:network_type: l3_ext`
- `provider:physical_network: <l3_gw_service_uuid>`
- `provider:segmentation_id: <vlan_id>`

The second parameter identifies the gateway service associated with the external network, whereas the third parameter is optional and allows to specify a particular VLAN tag to be used on the external interface of the gateway.

For instance the following command will create an external network mapped to a non-default gateway service:

```
$ neutron net-create external-2 --router:external
--provider:network_type=l3_ext
--provider:physical_network=58e1e906-39c1-4599-8b87-e00274c249ab
```

Note: The `--router:external` parameter is required when creating the `l3_ext` net.

This network is then used as any other network for setting the external gateway for a router:

```
$ neutron router-gateway-set router-2 external-2
```

In the underlying NSX platform the L3 Gateway Attachment for the router's gateway port will specify the `58e1e906-39c1-4599-8b87-e00274c249ab` layer-3 gateway service.

Note: Routers handling exclusively east-west traffic, i.e.: without an external gateway, are always implemented using the default layer-3 gateway service.

Metadata Support

VM Instances often use the nova metadata server to retrieve information such as keys for ssh access. Instances will send HTTP requests to a pre-defined address 169.254.169.254 in order to download this information.

The Neutron NSX plugin offers two approaches for allowing instances to reach the metadata server:

- **Metadata Access Network:** This mode allow for accessing the metadata server through a Neutron router. Metadata traffic is routed to a "metadata access network", which is connected to the router, but invisible to the user. This mode requires IP namespaces.
- **Metadata Host Route:** This is the only mode supported when the infrastructure does not support IP namespaces. If IP namespaces are supported, one of the previous two mode should be selected. This mode requires instances to support DHCP option 121 (classless static route). This mode works in a similar way as the isolated metadata proxy.

The following table provides a quick reference for knowing whether a particular metadata mode should be used according to requirements.

	IP Namespaces Supported	**IP Namespaces NOT supported**
Option 121 supported	- Metadata Access Network	- Metadata Host Route

For configuring the metadata mode, it might be necessary to edit both the `nsx.ini` and `dhcp_agent.ini` configuration files. For each mode, the following table report the configuration settings for both files.

Mode	nsx.ini	dhcp_agent.ini

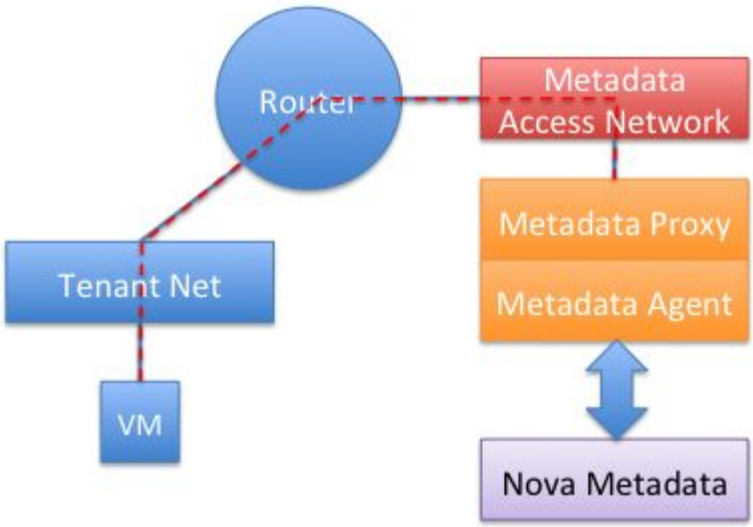
Metadata access network	`[nsx]`
	`metadata_mode = access_network`
		`[DEFAULT]`
		`enable_metadata_network = True`
		`enable_isolated_metadata = True`

Metadata host route	`[nsx]`
	`metadata_mode = dhcp_host_route`
		`[DEFAULT]`
		`enable_metadata_network = False`
		`enable_isolated_metadata = False`

In the remainder of this section, we'll discuss the supported metadata access modes in detail.

Metadata Access Network

This metadata access mode uses the Neutron logical router to forward metadata traffic to a special-purpose "metadata access" network. This network will the host a metadata access proxy[5] which forwards traffic to the metadata agent and then to Nova Metadata server, as depicted in the following diagram.



[5] This instance of the metadata proxy will handle requests for all networks attached to the router associated with the metadata access network.

In order to enable this mode:

1. The NSX plugin should be configured to automatically attach the metadata access network when a router interface is added. This is achieved by setting `metadata_mode` option to `access_network` in `nsx.ini`
2. The DHCP agent should be configured to handle metadata access networks. To this aim, also the isolated metadata proxy should be enabled. Therefore both the `enable_metadata_network` and `enable_isolated_metadata` parameters should be set to `True`

Metadata Host Route

As stated earlier in this section, this approach leverages the DHCP agent to inject a host route in VM instances in order to redirect metadata traffic through the dhcp agent itself.

Further configuration to be performed on the DHCP Agent:

1. Configure IP 169.254.169.254 on lo for the node where the dhcp agent is running, and add a NAT PREROUTING rule to redirect all traffic to this address on port 80 to port 8775 (or whatever port the nova metadata server is using).

   If the metadata server is running on a host different from the one where the dhcp agent is running, the iptables NAT table should be configured to redirect traffic to 169.254.169.254 to `<metadata_ip>:<metadata_port>`

This can be done using the following command:

```
$ iptables -t nat -A PREROUTING -d 169.254.169.254 -p tcp -m tcp
--dport 80 -j DNAT --to-destination 172.16.110.201:8775
```

2. Ensure the `metadata_mode` option in `nsx.ini` (`[nsx]` section) is set to `dhcp_host_route`. If not, please bear in mind that the Neutron service should be restarted after altering `nsx.ini`.

3. Ensure your images are able to handle DHCP option 121

This would result in a route to 169.254.169.254/32 being added to the VM's routing table. The next hop would be the DHCP agent itself.

The route will also be updated if the address of the DHCP server should change; however this would not be reflected on the VM's route table until the DHCP lease expires (by default, expiration time is 120 seconds).

Known Issues

When using the metadata access network, occasional failures might be experienced when removing the last interface from a router. Should that happen, the following workaround might be considered:

- Re-run the command for removing the router interface:
  `neutron router-interface-delete <router> <subnet>`
  Do not worry if this operation fails because the interface is not found.
- Remove the router: `neutron router-delete <router>`
- Re-create the router: `neutron router-create <router>`

User-configurable connectivity timeouts

The NSX-mh plugin allows one to configure custom timeout for the API connectivity between the plugin and NSX API Server.

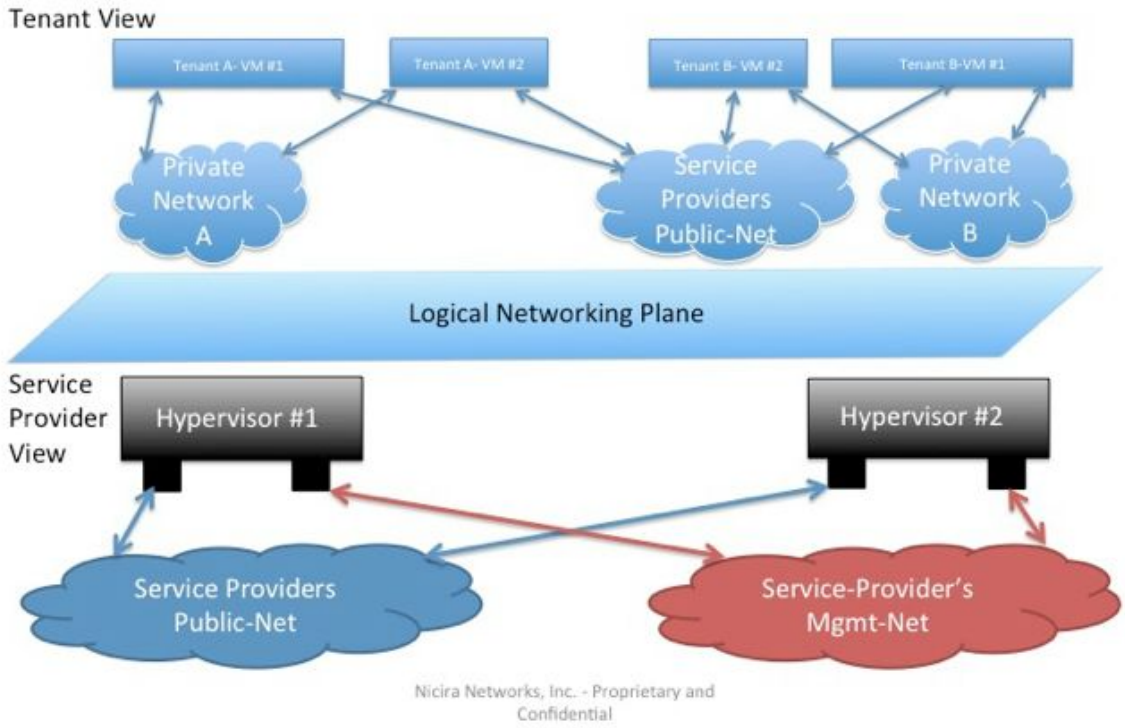
The following parameters are user configurable (through `nsx.ini`):

- `http_timeout`: How long to wait before aborting an unresponsive controller (and allow for retries to another controller in the cluster).
- `retries`: The maximum number of times to retry a particular request.
- `redirects`: The maximum number of times to follow a REDIRECT response from the server.

The default values for these will be as follows:

- `http_timeout`: 30 (seconds)
- `retries`: 2
- `redirects`: 2

Appendix A: Reference Network Architecture



- In logical space, each virtual machine is connected to a shared "Public" network.
- In logical space, each virtual machine also has access to a tenant-specific, isolated private network.
- In transport space, each hypervisor is connected to the "Public" network on one particular NIC.
- In transport space, each hypervisor has a NIC connected to a "Management" network. The management network is leveraged for inter-hypervisor communication as well as for creating "tunneled" L2 over L3 networks, which are represented as "logical" networks in the "Tenant View" of the diagram.

Appendix B: Suggested upgrade workflow

The instructions provided here are intended to supplement your OpenStack vendor's upgrade instructions and outline upgrade of the Neutron NSX-mh plugin only. Information contained herein is intended for reference.

1. Backup your existing neutron configuration. Typically this is located under `/etc/neutron/`
2. Install or update the neutron based packages using your OpenStack vendors instructions.
3. Using your backed up configuration as a reference, copy the applicable values into the new neutron configuration files. This includes `neutron.conf`, `dhcp_agent.ini`, `metadata_agent.ini` and `nsx.ini`.

4. Mark the current database as Icehouse:

```
$ neutron-db-manage --config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugins/vmware/nsx.ini stamp icehouse
```

5. Run schema and data migration:

```
$ neutron-db-manage --config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugins/vmware/nsx.ini upgrade head
```

6. Restart neutron server.