Networking Midonet Documentation

Release 11.1.0.dev3

OpenStack Foundation

Sep 26, 2020

CONTENTS

CHAPTER

ONE

SUPPORTED FEATURES

1.1 Neutron extensions supported by MidoNet

MidoNet provides the following Neutron API extensions. (The list doesnt include extensions implemented by Neutron in a mostly backend-agnostic way, like `subnet_allocation` and `standard-attr-revisions`.)

Category	Extension Alias	Required MidoNet version
Core	extra_dhcp_opt	>=5.2.1
	port-security	>=5.0
	allowed-address-pairs	>=5.0
	external-net	>=5.0
	provider	>=5.0
	security-group	>=5.0
L3	router	>=5.0
	extraroute	>=5.0
	ext-gw-mode	>=5.0
	router-interface-fip	>=5.1.1
	fip64	>=5.4
QoS	qos	>=5.4
L2Gateway	gateway-device	>=5.1
	l2-gateway	>=5.1
	l2-gateway-connection	>=5.1
BGP	bgp	>=5.2
	bgp-speaker-router-insertion	>=5.2
TaaS	taas	>=5.2
VPNaaS	vpnaas	>=5.1
	vpn-endpoint-groups	>=5.1

1.2 FAQ

- MidoNet doesnt support IPv6 in general. An exception is the `fip64` extension.
- While the IPAM part of Neutron `address-scope` extension does work with networking-midonet, the routing decision part of it is not implemented.
- MidoNet doesnt support Neutron `dvr` extension because L3 routing is always distributed in MidoNet.
- MidoNet doesnt support Neutron `l3-ha` extension. In MidoNet, Its common to use multiple router ports with BGP to provide a redundancy.[1]

[1] https://docs.midonet.org/docs/latest-en/operations-guide/content/configuring_uplinks.html

CHAPTER

TWO

INSTALLATION AND CONFIGURATION

2.1 Supported MidoNet versions

The current set of supported versions of MidoNet are:

- v5.x

NOTE: MidoNet changed its versioning scheme. v5.0 is what used to be called v2015.09.

2.2 How to Install

For productional deployments, we recommend to use a package for your distribution if available:

http://builds.midonet.org/

You can install the plugin from the source code by running the following command:

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

The plugin requires python-midonetclient package, which is usually available along with other midonet packages. Its recommended to use the same version of python-midonetclient and midonet-cluster. Alternatively, you can install python-midonetclient from source:

```
$ sudo pip install -e 'git://github.com/midonet/midonet.git@master
↪#egg=midonetclient&subdirectory=python-midonetclient'
```

2.3 MidoNet API configuration

You need to configure the Neutron server how to talk with your MidoNet cluster.

- Prepare a Keystone credential, with which the Neutron server will send requests to the MidoNet cluster.
- Configure Keystone authentication in MidoNet. See Using the Keystone authentication service in MidoNet documentation for details.
- Configure the following options in the Neutron server configuration file.

  midonet.midonet_uri *midonet.username* *midonet.password* *midonet.project_id*

2.4 ML2 mechanism and type drivers

networking-midonet is compatible with ML2 plugin. ML2 mechanism driver and type drivers for MidoNet are available:

```
[DEFAULT]
core_plugin = ml2

[ml2]
tenant_network_types = midonet
type_drivers = midonet,uplink
mechanism_drivers = midonet
```

2.5 L3 service plugin

networking-midonet uses its own L3 service plugin:

```
[DEFAULT]
service_plugins = midonet_l3
```

2.5.1 L3 extensions

In addition to the standard ones, networking-midonet L3 service plugin provides the following extensions. No additional configurations are necessary to enable them. They are enabled unconditionally.

- router-interface-fip (MidoNet 5.1.1 and later)
- fip64 (MidoNet 5.4 and later)

2.6 Interaction with Neutron agents

No Neutron agents are necessary for networking-midonet.

You can configure networking-midonet work with Neutron DHCP and Metadata agents. But it isnt recommended anymore.

For details, please refer to MidoNet documentation:

https://docs.midonet.org

2.6.1 Interface driver

Neutron agents use *interface driver* to connect themselves into the datapath. In case of MidoNet, they should be configured with the MidoNet interface driver.:

```
[DEFAULT]
interface_driver = midonet
```

2.7 VPNaaS

Starting v5.1, MidoNet implements Neutron VPNaaS extension API.

MidoNet plugin implements VPNaaS as a service driver. To configure it, add the following entries in the Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
service_plugins = vpnaas

[service_providers]
service_provider=VPN:Midonet:midonet.neutron.services.vpn.service_drivers.
↪midonet_ipsec.MidonetIPsecVPNDriver:default
```

NOTE: This plugin does not use Neutron VPNaaS agent.

2.8 Gateway Device Service

Starting v5.1, MidoNet implements Gateway Device Service vendor extension API.

To configure it, add the following service plugin to the *service_plugins* list in the DEFAULT group of */etc/neutron/neutron.conf*:

```
[DEFAULT]
service_plugins = midonet_gwdevice
```

2.9 L2 Gateway Service

Starting v5.1, MidoNet implements Neutron L2 Gateway Service extension API. The implementation differs slightly from upstream. Please check the spec to see the differences:

https://docs.openstack.org/networking-midonet/latest/specs/mitaka/border_gw.html

MidoNet plugin implements L2 Gateway Service as a service driver. To configure it, add the following service plugin to the *service_plugins* list in the DEFAULT group of */etc/neutron/neutron.conf*:

```
[DEFAULT]
service_plugins = midonet_l2gw
```

In addition, configure the service provider in the service_providers group of L2 Gateway plugin configuration file */etc/neutron/l2gw_plugin.ini*:

```
[service_providers]
service_provider = L2GW:Midonet:midonet.neutron.services.l2gateway.service_
↪drivers.l2gw_midonet.MidonetL2gwDriver:default
```

2.10 BGP dynamic routing service

Starting v5.2, MidoNet implements Neutron BGP dynamic routing service extension API. The implementation differs from upstream as follows:

- Router that is treated as bgp-speaker can be specified explicitly.
- Bgp-peer can relate to only one bgp-speaker.
- Binding network to bgp-speaker must be done before associating peers.
- Removing network from bgp-speaker must be done after all peers are disassociated from the bgp-speaker.
- Only one network can be associated with a bgp-speaker.
- Advertise_floating_ip_host_routes and advertise_tenant_networks are ignored.
- Attached network to the router and destination network in extra routes on the router are showed as advertised routes.

To configure it, add the following service plugin to the *service_plugins* list in the DEFAULT group of */etc/neutron/neutron.conf*:

```
[DEFAULT]
service_plugins = midonet_bgp
```

2.11 Tap-as-a-Service

Starting v5.2, MidoNet implements Tap-as-a-Service extension API.

MidoNet plugin implements TaaS as a service driver. To configure it, add the following entries in the Neutron configuration file */etc/neutron/neutron.conf*:

```
[DEFAULT]
service_plugins = taas
```

In addition, configure the service provider in the service_providers group of TaaS plugin configuration file */etc/neutron/taas_plugin.ini*:

```
[service_providers]
service_provider = TAAS:Midonet:midonet.neutron.services.taas.service_
↪drivers.taas_midonet.MidonetTaasDriver:default
```

2.12 QoS

Starting v5.4, MidoNet implements Neutron QoS extension API. Its automatically enabled when QoS service plugin is configured. No MidoNet-specific configuration is necessary.

2.12.1 QoS service plugin

QoS service plugin can be configured in the Neutron server configuration file */etc/neutron/neutron.conf*:

```
[DEFAULT]
service_plugins = qos
```

2.12.2 QoS core resource extension for ML2

QoS core resource extension for ML2 plugin can be configured in the Neutron server configuration file */etc/neutron/neutron.conf*:

```
[ml2]
extension_drivers = qos
```

2.13 Horizon

Starting with Newton, Horizon has built-in support for MidoNet network types.

To enable it, add the following configuration to the *OPENSTACK_NEUTRON_NETWORK* dict in *local_settings.py*:

```
'supported_provider_types': ['midonet', 'uplink'],
```

2.14 Magnum

Starting v5.2, MidoNet can be used for Magnum deployment.

CHAPTER

THREE

RELEASE NOTES

See https://docs.openstack.org/releasenotes/networking-midonet/.

CHAPTER

FOUR

MIGRATION FROM MONOLITHIC V2 PLUGIN TO ML2 PLUGIN

4.1 Overview

MidoNet monolithic v2 plugin (*midonet_v2*) is not supported anymore. When upgrading to Pike, a deployer needs to switch to ML2 with MidoNet mechanism driver. This document outlines the migration procedure.

Note: The procedure documented here is appropriate only when upgrading to Pike.

4.2 How to migrate

0. Take a backup. (not strictly necessary but strongly recommended)
1. Upgrade to Pike as usual.

   This step includes the usual DB migration via neutron-db-manage.
2. Update neutron configuration to use ML2 plugin.

   See the following section for examples.
3. Start neutron server.

   On the first startup, MidoNet mechanism driver automatically migrates the data in the Neutron DB from the form what MidoNet monolithic plugin recognises. This is a one-way migration. On successful migration, the message DB Migration from MidoNet v2 to ML2 completed successfully will be logged in the neutron server log at *INFO* level.

Note: The Step 3 assumes its the only neutron server process using the Neutron DB. If your deployment has multiple neutron servers, make sure to shut them down prior to Step 3. After verifying that the migration succeeded, you can start them. Also, make sure that they are also configured to use ML2 and midonet mechanism driver.

4.2.1 Neutron server configuration

Basically, you need to:

- Change *core_plugin* to *ml2*
- Add the *ml2* group.

The `midonet` group is common for both of the monolithic plugin and the ML2 driver. It doesnt need any changes.

Example before migration

```
[DEFAULT]
core_plugin = midonet_v2

# Alternatively you might have non-alias version.
#core_plugin = midonet.neutron.plugin_v2.MidonetPluginV2

[MIDONET]
cluster_port = 8088
cluster_ip = 192.168.137.129
client = midonet.neutron.client.api.MidonetApiClient
project_id = service
password = midonet_service_password
username = neutron
midonet_uri = http://192.168.137.129:8181/midonet-api
```

Example after migration

```
[DEFAULT]
core_plugin = ml2

[MIDONET]
cluster_port = 8088
cluster_ip = 192.168.137.129
client = midonet.neutron.client.api.MidonetApiClient
project_id = service
password = midonet_service_password
username = neutron
midonet_uri = http://192.168.137.129:8181/midonet-api

[ml2]
mechanism_drivers = midonet
type_drivers = midonet,uplink
tenant_network_types = midonet
external_network_type = midonet
extension_drivers = port_security,qos
# Or, in case you don't have qos service plugin configured.
#extension_drivers = port_security
```

CHAPTER

FIVE

HISTORICAL UPGRADE NOTES

Note: For releases newer than Mitaka, please look at Release Notes.

This section describes changes which might impact upgrades from the previous releases.

5.1 From Liberty to Mitaka

- Neutron MidoNet interface driver has been moved out of Neutron tree. If your deployment uses Neutron DHCP agent and its configuration doesnt use the stevedore alias (midonet), you should update it:

  Before:

```
interface_driver = neutron.agent.linux.interface.
↪MidonetInterfaceDriver
```

  After:

```
interface_driver = midonet
```

- The following sub-commands were removed from *midonet-db-manage* command:

```
current             Display the current revision for a database.
history             List changeset scripts in chronological order.
branches            Show current branch points
check_migration     Show current branch points and validate head file
upgrade             Upgrade to a later version.
downgrade           (No longer supported)
stamp               'stamp' the revision table with the given␣
↪revision;
revision            Create a new revision file.
```

  You can use *neutron-db-manage subproject networking-midonet* instead.

  For example,:

```
$ neutron-db-manage --subproject networking-midonet upgrade head
```

- At the start of the Mitaka development cycle (immediately after liberty db milestone), our sub-project db migration chain was separated into two branches, expand and contract, to allow a shorter

downtime as Neutron does. See the blueprint[1] for details.

5.2 From Kilo to Liberty

- v2 plugin was separated into two plugins, core plugin and L3 service plugin. You need to configure L3 service plugin in addition to the core plugin:

```
core_plugin = midonet.neutron.plugin_v2.MidonetPluginV2
service_plugins = midonet.neutron.services.l3.l3_midonet.
↪MidonetL3ServicePlugin
```

- Plugin entry point for v1 plugin (the older plugin which is compatible with MidoNet v2015.03 and v2015.06) has been moved out of Neutron tree:

  Before:

```
core_plugin = neutron.plugins.midonet.plugin.MidonetPluginV2
```

  After:

```
core_plugin = midonet.neutron.plugin_v1.MidonetPluginV2
```

- *midonet-db-manage* command is now obsolete. While its still provided for backward compatibility, we plan to remove it in a feature release. You can use *neutron-db-manage subproject networking-midonet* instead.

  For example,:

```
$ neutron-db-manage --subproject networking-midonet upgrade head
```

5.3 From Juno to Kilo

- A separate plugin (v2 plugin) which is compatible with MidoNet v5.0 (previously called v2015.09) was introduced:

```
core_plugin = midonet.neutron.plugin_v2.MidonetPluginV2
```

[1] http://specs.openstack.org/openstack/neutron-specs/specs/liberty/online-schema-migrations.html

CHAPTER

SIX

NETWORKING-MIDONET CONFIGURATION GUIDE

This section provides a list of all possible options for each configuration file.

6.1 Configuration

Networking-midonet uses the following configuration options in the Neutron server configuration, which is typically */etc/neutron/neutron.conf*.

6.1.1 midonet

midonet_uri

Type string

Default `http://localhost:8080/midonet-api`

MidoNet API server URI. Note that, for historical reasons, the port number in the default value (8080) does not match the default of the MidoNet API in MidoNet 5.0 and later, which is 8181. Even if you configured the MidoNet API to use port 8080, we recommend to configure this option explicitly because the default value may change in the future release of networking-midonet.

username

Type string

Default `admin`

MidoNet admin username.

password

Type string

Default `passw0rd`

MidoNet admin password.

project_id

Type string

Default `77777777-7777-7777-7777-777777777777`

ID of the project that MidoNet admin user belongs to.

tunnel_protocol

Type string

Default `vxlan`

Tunnel protocol used by Midonet. Currently unused.

`cluster_ip`

Type string

Default `localhost`

IP that the cluster service can be reached on. Currently unused.

`cluster_port`

Type string

Default `8088`

Port that the cluster service can be reached on. Currently unused.

`client`

Type string

Default `midonet.neutron.client.api.MidonetApiClient`

MidoNet client used to access MidoNet data storage. Do not change unless you want to try the experimental Task-based API.

6.2 Policy

Networking-MidoNet, like most OpenStack projects, uses a policy language to restrict permissions on REST API actions.

6.2.1 networking-midonet policies

The following is an overview of all available policies in networking-midonet. For a sample configuration file, refer to *Sample Networking-MidoNet Policy File*.

networking-midonet

`create_bgp_speaker:logical_router`

Default `rule:admin_only`

Operations

- **POST** `/bgp-speakers`

Create a BGP speaker with `logical_router` attribute

`create_gateway_device`

Default `rule:admin_or_owner`

Operations

- **POST** `/gw/gateway-devices`

Create a gateway device

update_gateway_device

Default `rule:admin_or_owner`

Operations

- **PUT** `/gw/gateway-devices/{id}`

Update a gateway device

delete_gateway_device

Default `rule:admin_or_owner`

Operations

- **DELETE** `/gw/gateway-devices/{id}`

Delete a gateway device

get_gateway_device

Default `rule:admin_or_owner`

Operations

- **GET** `/gw/gateway-devices`
- **GET** `/gw/gateway-devices/{id}`

Get gateway devices

6.2.2 Sample Networking-MidoNet Policy File

The following is a sample networking-midonet policy file for adaptation and use.

The sample policy can also be viewed in `file form`.

Important: The sample policy file is auto-generated from networking-midonet when this documentation is built. You must ensure your version of networking-midonet matches the version of this documentation.

```
# Create a BGP speaker with ``logical_router`` attribute
# POST  /bgp-speakers
#"create_bgp_speaker:logical_router": "rule:admin_only"

# Create a gateway device
# POST  /gw/gateway-devices
#"create_gateway_device": "rule:admin_or_owner"

# Update a gateway device
# PUT  /gw/gateway-devices/{id}
#"update_gateway_device": "rule:admin_or_owner"

# Delete a gateway device
# DELETE  /gw/gateway-devices/{id}
#"delete_gateway_device": "rule:admin_or_owner"
```

(continues on next page)

(continued from previous page)

```
# Get gateway devices
# GET  /gw/gateway-devices
# GET  /gw/gateway-devices/{id}
#"get_gateway_device": "rule:admin_or_owner"
```

CHAPTER

SEVEN

CONTRIBUTOR GUIDE

7.1 DevStack plugin

7.1.1 DevStack external plugin

networking-midonet has its devstack plugin. The following `local.conf` snippet would enable it:

```
enable_plugin networking-midonet https://opendev.org/openstack/networking-
↪midonet
```

local.conf examples

ML2 Plugin with MidoNet drivers

You can find an example at devstack/ml2/local.conf.sample in the source tree.

```
[[local|localrc]]

# This configuration file is intended to be used for CI and local␣
↪develeopment
# where you may only want networking related services to be running.

# Load the devstack plugin for ml2
Q_PLUGIN=ml2
Q_AGENT=none  # dummy value to avoid pulling functions from openvswitch_
↪agent
Q_ML2_PLUGIN_MECHANISM_DRIVERS=midonet
Q_ML2_PLUGIN_TYPE_DRIVERS=midonet,uplink
Q_ML2_TENANT_NETWORK_TYPE=midonet
ML2_L3_PLUGIN=midonet_l3

enable_plugin networking-midonet https://opendev.org/openstack/networking-
↪midonet

# Set all the passwords
DATABASE_PASSWORD=midonet
RABBIT_PASSWORD=midonet
SERVICE_TOKEN=midonet
SERVICE_PASSWORD=midonet
ADMIN_PASSWORD=midonet
```

(continues on next page)

(continued from previous page)

```
# Enable keystone, nova, glance and neutron
# Also enable tempest since it's useful for CI and local dev
ENABLED_SERVICES=rabbit,mysql,key
ENABLED_SERVICES+=,n-api,n-crt,n-obj,n-cpu,n-cond,n-sch,placement-api
ENABLED_SERVICES+=,g-api,g-reg
ENABLED_SERVICES+=,q-svc,neutron
ENABLED_SERVICES+=,tempest
ENABLED_SERVICES+=,horizon
ENABLED_SERVICES+=,n-novnc,n-cauth

NOVA_VNC_ENABLED=True

# If you want to use Gateway Device Management Service,
# uncomment the following lines.
# Q_SERVICE_PLUGIN_CLASSES+=,midonet_gwdevice

# The following section contains environment variable settings that would
# further minimalize the environment.  For example, in some cases you may␣
↪just
# want to run the Neutron API with no agents, or you may want to disable
# authentication.

# If you want to also disable auth for Neutron, uncomment this
#
# Q_AUTH_STRATEGY=noauth

# If you don't want the host to be configured with fake uplink uncomment␣
↪this
#
# MIDONET_CREATE_FAKE_UPLINK=False

# If you don't want devstack to create default networks, uncomment this
#
# NEUTRON_CREATE_INITIAL_NETWORKS=False

# If you want to disable Neutron agents so that only the API runs,␣
↪uncomment
# these
#
# disable_service q-dhcp
# disable_service q-meta

# If you want to use the embedded metadata proxy, uncomment these:
#
# MIDONET_USE_METADATA=True
# Q_METADATA_ENABLED=True
# disable_service q-dhcp
# disable_service q-meta

# If you want to use L2 Gateway Management Service,
# uncomment the following lines.
# enable_plugin networking-l2gw https://opendev.org/x/networking-l2gw
# enable_service l2gw-plugin
# L2GW_PLUGIN="midonet_l2gw"
# NETWORKING_L2GW_SERVICE_DRIVER="L2GW:Midonet:midonet.neutron.services.
↪l2gateway.service_drivers.l2gw_midonet.MidonetL2gwDriver:default"
```

(continues on next page)

(continued from previous page)

```
# If you want to use VPNaaS, uncomment the following lines and manually␣
↪install
# ipsec package "libreswan".
#
# enable_plugin neutron-vpnaas https://opendev.org/openstack/neutron-vpnaas
# enable_service neutron-vpnaas
# NEUTRON_VPNAAS_SERVICE_PROVIDER="VPN:Midonet:midonet.neutron.services.
↪vpn.service_drivers.midonet_ipsec.MidonetIPsecVPNDriver:default"

# If you want to use BGP dynamic routing service,
# uncomment the following lines.
# Q_SERVICE_PLUGIN_CLASSES+=,midonet_bgp

# If you want to use Logging Resource Service, uncomment the following␣
↪lines.
# Q_SERVICE_PLUGIN_CLASSES+=,midonet_logging_resource

# If you want to use Tap as a Service, uncomment the following lines.
# enable_plugin tap-as-a-service https://opendev.org/x/tap-as-a-service
# enable_service taas
# TAAS_SERVICE_DRIVER="TAAS:Midonet:midonet.neutron.services.taas.service_
↪drivers.taas_midonet.MidonetTaasDriver:default"
```

MidoNet backend communication

MidoNet exposes two ways to communicate to its service:

1. REST (synchronous)
2. Tasks DB (asynchronous - experimental)

By default, the plugin is configured to use the REST API service. The REST API client is specified as:

```
MIDONET_CLIENT=midonet.neutron.client.api.MidonetApiClient
```

If you want to use the experimental Tasks based API, set the following:

```
MIDONET_CLIENT=midonet.neutron.client.cluster.MidonetClusterClient
```

VPNaaS

Starting v5.1, MidoNet implements Neutron VPNaaS extension API. To configure MidoNet as the VPNaaS driver when running devstack, make sure the following is defined in `local.conf`:

```
enable_plugin neutron-vpnaas https://opendev.org/openstack/neutron-vpnaas
enable_service neutron-vpnaas
NEUTRON_VPNAAS_SERVICE_PROVIDER="VPN:Midonet:midonet.neutron.services.vpn.
↪service_drivers.midonet_ipsec.MidonetIPsecVPNDriver:default"
```

NOTE: Currently, this devstack plugin doesnt install ipsec package libreswan. Please install it manually.

Gateway Device Management Service

Starting v5.1, MidoNet implements Neutron Gateway Device Management Service extension API. To configure MidoNet including Gateway Device Management Service when running devstack, make sure the following is defined in `local.conf`:

```
Q_SERVICE_PLUGIN_CLASSES=midonet_gwdevice
```

L2 Gateway Management Service

Starting v5.1, MidoNet implements Neutron L2 Gateway Management Service extension API. To configure MidoNet including L2 Gateway Management Service when running devstack, make sure the following is defined in `local.conf`:

```
enable_plugin networking-l2gw https://opendev.org/x/networking-l2gw
enable_service l2gw-plugin
Q_PLUGIN_EXTRA_CONF_PATH=/etc/neutron
Q_PLUGIN_EXTRA_CONF_FILES=(l2gw_plugin.ini)
L2GW_PLUGIN="midonet_l2gw"
NETWORKING_L2GW_SERVICE_DRIVER="L2GW:Midonet:midonet.neutron.services.
↪l2gateway.service_drivers.l2gw_midonet.MidonetL2gwDriver:default"
```

BGP dynamic routing service

Starting v5.2, MidoNet implements Neutron BGP dynamic routing service extension API. The implementation differs slightly from upstream. In MidoNet, router treated as bgp-speaker must be specified.

To configure MidoNet including BGP dynamic routing service when running devstack, make sure the following is defined in `local.conf`:

```
enable_plugin neutron-dynamic-routing https://opendev.org/openstack/
↪neutron-dynamic-routing
DR_MODE=dr_plugin
BGP_PLUGIN=midonet_bgp
enable_service q-dr
```

QoS

The following `local.conf` snippet would enable QoS extension with MidoNet driver:

```
enable_plugin neutron https://opendev.org/openstack/neutron
enable_service q-qos
```

Tap as a service

The following `local.conf` snippet would enable Tap-as-a-service support:

```
enable_plugin tap-as-a-service https://opendev.org/openstack/tap-as-a-
↪service
enable_service taas
TAAS_SERVICE_DRIVER="TAAS:Midonet:midonet.neutron.services.taas.service_
↪drivers.taas_midonet.MidonetTaasDriver:default"
```

7.2 Policies

7.2.1 Policies

This page explains development policies for *networking-midonet* project.

See also:

- Neutron Policies
- OpenStack Developers Guide

Review and merge patches

- How to find patches to review

  Unlike some of other gerrit-using communities, (e.g. midonet project on gerrithub) a submitter of patches usually doesnt add reviewers to their patches explicitly. (Nor recommended to do so) We consider its reviewers responsibility to find patches to review. There are a few tools available to help the process.

  - Gerrit dashboards[1]
  - Email notifications from gerrit[2]
  - Gerrit notifications on Freenode IRC channels, #openstack-neutron[3] and #midonet[4]

- We require two +2 votes before merging a patch

  When you merge a patch without two +2 votes, please leave a message to explain why. E.g. This is a trivial fix for a problem blocking other projects.

  Usually the reviewer who voted the second +2 also make it Workflow +1. It makes zuul run the gate jobs for the patch and merge it if tests succeeded. Of course, its also ok for the reviewer to choose not to put Workflow +1. E.g. When he thinks more reviews are desirable. E.g. When the gate jobs are known to be broken. (In that case, running them would just waste the infra resources.)

- Do not ignore the result of non-voting jobs

[1] https://docs.openstack.org/networking-midonet/latest/contributor/dashboards.html#gerrit-dashboards
[2] https://review.opendev.org/#/settings/projects
[3] http://eavesdrop.openstack.org/irclogs/%23openstack-neutron/latest.log.html
[4] http://eavesdrop.openstack.org/irclogs/%23midonet/latest.log.html

When you merge a patch with non-voting jobs failing, please leave a message to explain why. Please make sure that theres a bug filed for the symptom. E.g. Test failures are unrelated to this patch. bug xxxxxx.

- Document recheck reasons

  Writing a comment starting with recheck[5] on the gerrit, you can re-trigger test jobs for the patch. Please try to examine the failure and explain why a recheck was necessary in the comment. A bug reference is the most appropriate. E.g. recheck bug xxxxxxx E.g. recheck builds.midonet.org connection timeout

- Check the rendered htmls when reviewing document changes

  Test jobs `build-openstack-sphinx-docs` and `build-openstack-releasenotes` provide the rendered results for the change.

7.3 Programming HowTos and Tutorials

7.3.1 Alembic Migrations

Script Auto-generation

Please refer to the Neutron documentation Script Auto-generation for the instruction to auto-generate migration scripts.

You need to specify *subproject networking-midonet* option to the *neutron-db-manage* command to generate a migration script for this project.

Depending on *aaS and other sub-projects set up in your environment, you might need to edit the generated script. Typically, you need to remove extra drop table ops for tables which dont belong to our project.

7.3.2 Tests

You can run the unit tests with the following command:

```
$ tox -e py27
```

It installs its requirements to `.tox/py27` on the initial run.

7.4 Dashboards

7.4.1 Dashboards

Gerrit Dashboards

Networking-MidoNet Review Inbox

[5] https://opendev.org/opendev/project-config/src/commit/cef92ba2d9abc000fb5a35b85904e8dc2bf6be1a/zuul.d/pipelines.yaml#L16-L17

Grafana Dashboards

Networking-MidoNet Failure Rate

CHAPTER

EIGHT

SPECIFICATIONS

8.1 Kilo specs

8.1.1 Agent API

In a Neutron-MidoNet deployment, where numerous agents are running on various hosts to provide services, it is important that the operators have a way to view these agents to check their existence and health. Neutron already provides this feature with its agent extension API [1]. This document describes the design of the agent extension API implementation by the MidoNet Neutron plugin that provides this feature also for the MidoNet agents.

Problem Description

There is currently no way to display the information about the MidoNet agents that are deployed using the Neutron API. Such information is useful for operators to see their liveness and the location that they are deployed on.

Additionally, the operators need to find out the IDs of the agents, as well as the IP addresses of the hosts that the agents are running on, to be able to execute the agent_membership API, where both of these values are needed in the input. Without executing the agent_membership API, the MidoNet agents cannot create tunnels among themselves, and VMs running on remote hosts would not be able to connect to each other.

Proposed Change

Implement the existing agent Neutron extenion API in the MidoNet Neutron plugin that provide (at the very least) the following:

- Display all the MidoNet agents deployed, with the id field indicating the globally unique identifier of each agent that can be used in the agent membership API
- Show the IP addresses of the host that the agent is running on
- Show the aliveness of the agents

The decision to provide these using the existing agent Neutron extension as opposed to creating a new vendor extension is that there are significant overlaps between the two and the agent extension provides integration with Horizon and neutron CLI.

The following fields exist in the Neutron agent extension that MidoNet can provide:

- id: Unique identifier of the agent.

- agent_type: Represents the type of the agent. Midonet agent is the type for the MidoNet agents.
- binary: Represents the package name. For MidoNet, it is midolman.
- alive: Represents the liveness of the agent.
- description: The description of the agent. This is the only updatable field of the API.
- configurations: A dictionary that includes configurations specific to the agent. For MidoNet, this dictionary contains the IP addresses and the interfaces of the host:

```
{
    "interfaces":
        [
            {"name": INTERFACE_NAME, "ip_addresses": [IP_ADDRESSES]}
        ]
}
```

Since the agent API is designed with OpenStack agents in mind, however, there are fields in the API that MidoNet cannot provide.

The following fields are not supported by the MidoNet plugin at the time of this proposal:

- host: The host name where the agent is running. While this information is useful, it is not currently supported by MidoNet.
- topic: AMQP message topic to communicate with the agent. MidoNet agents do not support this.
- admin_state_up: Sets the administrative status of the agent. MidoNet does not support this. An attempt to update gets an unsupported error response.
- heartbeat_timestamp: The heartbeat for aliveness. MidoNet agents do not provide this data.

MidoNet agents, when they spawn, report their existence to the MidoNet Network State Database (NSDB). NSDB is also notified when the agent goes down. The MidoNet Cluster, through its RPC service, exposes an API to provide this data to the Neutron plugin. The RPC API provides all the information the plugin needs to populate the supported fields, and it also contains the most up-to-date information of the agents health. Because all the required agent information can be retrieved via the Cluster API on each Neutrons agent API, Neutrons agents DB table does not need to be populated for the MidoNet agents. The plugin is responsible for merging the OpenStack agent data and the MidoNet agent data.

Lastly, deletion of agents, supported by Neutron API, is not supported by the MidoNet plugin. Unsupported exception is thrown when a user attempts to delete an agent.

REST API

No change except that some fields will be unsupported as explained above.

DB Model

No change, and since all the MidoNet agent data are provided by MidoNe cluster, the agents database table in Neutron will be unused.

Security

No impact

Client

No code change is made, but neutron agent-delete‘ command is not supported since the MidoNet plugin does not allow agent deletion.

See the Neutron CLI documentation for more details on the agent commands[2].

References

Because of the lack of API documentation available, the Neutron agent extension API reference is its source code:

[1] https://github.com/openstack/neutron/blob/master/neutron/extensions/agent.py [2] http://docs.openstack.org/cli-reference/content/neutronclient_commands.html

8.1.2 Agent Membership API

In MidoNet, each MidoNet agent must be activated in order to join the MidoNet deployment. This step ensures that no rogue MidoNet agent automatically joins the MidoNet deployment. This document describes the agent-membership Neutron extension API that provides this feature.

Problem Description

In the previous MidoNet API, the authorization step to allow a MidoNet agent to be activated in the deployment was to add it to a tunnel zone.

This was undesirable because it required explicit tunnel zone coniguration using the API, and in an OpenStack-MidoNet deployment, there was no use case known or supported that requires more than one tunnel zone to exist. By forcing users to create a tunnel zone and adding individual hosts to them, it was creating unnecessary potential failure points without adding any value.

Proposed Change

Maintain a singleton default Tunnel Zone, with the name, DEFAULT, in the system. This tunnel zone is created automatically by the MidoNet cluster. The Neutron plugin signals the cluster to do create it when it starts up by submitting a new task type, CONFIG.

CONFIG task contains all the global configuration values settable in Neutron that MidoNet would find useful. The handling of the case in which the cluster fails to process this task is outside the scope of this proposal, and it is assumed that CONFIG task is treated the same as any other tasks.

For this particular change, only one new field is introduced in CONFIG, which is tunnel_protocol, that indicates the global tunneling protocol that MidoNet should use. This value is used by MidoNet to create the singleton Tunnel Zone. The default tunneling protocol used is vxlan, but you can override it by specifying the following in neutron.conf:

[MIDONET] tunnel_protocol=gre # Could be vxlan or gre

With this approach, the concept of Tunnel Zone is completely hidden from the user as well as from the neutron implementation.

To authorize an agent to be added to the deployment, agent-membership Neutron extension API described below is defined.

REST API

AgentMembership

Attribute Name	Type	POST/ PUT	Required	Description
id	string (UUID)	POST	generated	ID of the MidoNet agent, which maps to hostId in cluster
ip_address	string	POST	Yes	IP address to use for tunneling

Only POST and DELETE operations are permitted, and only admin can execute them.

Only IPv4 address is supported for ip_address.

id field is the ID of the MidoNet host object, which you can retrieve using the agent API extension of Neutron (not implemented yet). The agents and the MidoNet hosts map one-to-one. Likewise, the agent API will also include the host interfaces and their IP addresses, useful to populate the ip_address field for the agent membership API.

DB Model

midonet_agent_membership

Name	Type	Description
id	String	ID of the agent (same as host in the cluster)
ip_address	String	IP address to use for tunneling

Only IPv4 address is supported for ip_address.

New task types are:

- CONFIG: Represents global Neutron configurations
- AGENTMEMBERSHIP: Represents AgentMembershp resource

Security

Only admins are allowed to execute the agent-membership API. This explicit step to add each agent as a member provides an extra layer of security to prevent unwanted agents to join automatically.

Client

The following command lists all the memberships:

::

    neutron agent-membership-list [-h] [-P SIZE] [sort-key FIELD] [sort-dir {asc, desc}]

-h, help:: show the help message

-P SIZE, page-size SIZE:: Specify retrieve unit of each request

sort-key FIELD:: Sorts the list by the specified fields

sort-dir {asc,desc}:: Sorts the list in the specified direction

The following command adds an agent to the MidoNet deployment membership:

::

    neutron agent-membership-create [-h] [agent-id AGENT] [ip-address IP_ADDRESS]

-h, help:: show the help message

-a, agent-id:: Specify the ID of the agent to add to membership

    -a, --ip-address Set IP address to use for tunneling

The following command removes an agent from the MidoNet deployment membership:

:: neutron agent-membership-delete [-h] AGENT_MEMBERSHIP

AGENT_MEMBERSHIP:: ID of the agent membership to remove

8.1.3 Data Sync

In MidoNet 2.0, Neutron is the data store of network configurations and Cluster is the backend data store that contains MidoNet-specific data objects translated from Neutron data models.

Also, MidoNet 2.0 includes data syncing feature between Neutron and Cluster, which is internally implemented by importing the Neutron data into Cluster through the tasks table. The data sync feature is needed for the following use cases:

- Operator sets up MidoNet for the first time and wants to initialize the Cluster data
- Operator using a non-MidoNet Neutron plugin wishes to migrate over to MidoNet
- Operator wants to upgrade to the new version of MidoNet and the MidoNet data schema has changed.

- Modifying either the Neutron DB or MidoNet data store directly for some operational purpose led to severe data mismatch between Neutron and MidoNet, and wishes to re-sync.

Each data imported is versioned. This document describes the data version management feature of midonet-db-manage tool that includes syncing Neutron and Cluster data, and rolling back to one of the previous data versions.

The Cluster design of data import, and the upgrade process, are outside the scope of this document.

Problem Description

While MidoNet 2.0 is designed to provide the data syncing feature, currently there is no tool made available in Neutron to facilitate it. Without such a tool, the data sync feature does not get exposed to the operators, making the upgrade process and data re-syncing between Neutron and MidoNet challenging.

Proposed Change

midonet-db-manage, which manages the tasks table in Neutron DB among other DB related features, is enhanced with the following capabilities:

- Toggle the write access of the tasks table between read-only and read-write modes, which also toggles the API between read-only and read-write correspondingly
- Through the tasks table, signal to Cluster that data sync is about to start and execute the data import, and signal that the import has completed to activate the imported data
- Maintain all the past data sync events and their summaries

Deletion of the imported Cluster data is not included in this proposal, but it is planned to be included in one of the future releases.

Tasks Write Access

A new column is added to midonet_data_state table which indicates the write access to the tasks table. The default is read-write. The data sync operation is only allowed in the read-only mode, and the operator cannot switch back to read-write while the data sync is taking place. The plugin will throw 503 (Service Unavailable) on all the POST/PUT/DELETE Neutron API requests if the tasks table is in a read-only mode.

Data Version

Each data sync event is summarized and saved in the midonet_data_versions table as a new data version. A data version includes fields that are updated only by midonet-db-manage and fields updated only by Cluster.

The fields updated by midonet-db-manage are:

- id: Globally unique identifier of the data version
- sync_started_at: Time the sync started
- sync_tasks_status: Status of sync tasks insertion
- stale: Flag indicating that this data set is out of date

The field updated by Cluster is

- sync_finished_at: Time all the tasks were processed by Cluster

Both midonet-db-manage and Cluster update:

- **sync_status: Status of the sync. midonet_db_manager updates it when it starts,** and Cluster updates when it finishes processing all the tasks

How these fields get set are described in Tasks Data Sync section below.

The data version summary does not include the number of tasks processed but such information would be printed and logged from the midonet-db-manage command, and may be added to the table if there are clear use cases for it.

When the midonet-db-manage sync command is issued, id, sync_started_at and sync_tasks_status are initialized. sync_tasks_status is set to STARTED.

When the midonet-db-manage finishes adding all the tasks, sync_tasks_status is updated to COMPLETED.

Tasks Data Sync

The data sync operation is allowed only when the data write access is set to read-only.

The data sync operation is implemented as follows:

1. midonet-db-manage truncates the tasks table and inserts a DATA_VERSION_SYNC task in the first row. This task instructs Cluster that a new data sync has started, and it needs to prepare a new storage for the sync. midonet-db-manage sets sync_started_at to the current time, and sync_tasks_status and sync_status to STARTED. Syncing is disallowed in the following cases:
    a. Sync is already being executed
    b. There are unprocessed tasks (so that truncate does not delete unprocessed tasks)
2. Immediately following the DATA_VERSION_SYNC task insertion, midonet-db-manage queries Neutron DB and generates CREATE tasks for all the existing resources. Cluster processes them as usual and creates these resources in the backend.
3. Once all the tasks to re-create the Neutron objects are inserted into the tasks table, DATA_VERSION_ACTIVATE task is added to indicate that the import has finished. midonet-db-manage updates the sync_task_status to COMPLETED

midonet-db-manage sync command exits immediately after DATA_VERSION_ACTIVE task has been added to the tasks table, and does not know whether Cluster has successfully processed all the tasks. sync_status and sync_tasks_status exist to differentiate the statuses of the Cluster processing and the midonet-db-manage command.

Cluster, after processing DATA_VERSION_ACTIVATE task, updates sync_status to COMPLETED. If Cluster encounters an error, it updates sync_status to ERROR. In both cases, sync_finished_at is updated.

If there was an error while inserting sync tasks, midonet-db-manager updates sync_tasks_status to ERROR. If the sync command was forcefully terminated (SIGINT) by the user, then sync_tasks_status is set to ABORTED. In both cases, the command terminates immediately, and adds DATA_VERSION_ACTIVE task with the version ID set to the currently active data version (not the one being synced).

Data Version Activation

An active data version means that the data originated from this data sync event is what the MidoNet agents are currently using for packet simulation. At any time, exactly one data version may be active. When a data sync process completes, the newly imported data set is automatically activated.

In addition, midonet-db-manage offers a command to rollback to the previously active data version. A rollback could only happen during one read-only session. Once the operator sets the API to read-write, none of the previously synced data could be chosen for a rollback. You can only rollback to the data sync that was completed in the same read-only session. The operator is expected to do all the necessary verifications of the completed data sync before the data access is set back to read-write. When the data is set back to read-write, midonet-db-manage sets the stale field of all the non-active data versions to true.

When a data activation command is issued, midonet-db-manage sets the sync_status and task_status to STARTED. When the command completes, it sets the task_status to COMPLETED. Cluster, when it finishes the activation process, updates sync_status to COMPLETED, and active_data_version field of the midonet-data-state table to the activated version.

You can not go back to the read-write mode if either task_status or sync_status field is set to STARTED.

REST API

None

DB Model

midonet_data_versions

Name	Type	Description
id	Int	The version of the data
sync_started_at	DateTime	Time the data sync started
sync_finished_at	DateTime	Time the data sync finished
sync_status	String	Status of the sync operation
sync_tasks_status	String	Status of the sync tasks insertion
stale	Boolean	True if the date version is stale

The sync_status column could contain one of the following values:

- STARTED
- COMPLETED
- ERROR

The sync_tasks_status column could contain one of the following values:

- STARTED
- COMPLETED
- ABORTED
- ERROR

midonet_data_state

Rename midonet_task_state to midonet_data_state.

Add a new column to store the write access to the tasks table.

Name	Type	Description
active_version	Int	Active data version
readonly	Boolean	If true, tasks table is readonly

FLUSH task type is deleted, and new resource types, DATA_VERSION_SYNC and DATA_VERSION_ACTIVATE are created.

To start the data sync process, this is added in row 1 of the tasks table:

:: task_type: DATA_VERSION_SYNC resource_type: resource_id: <DATA_VERSION> data: {}

To activate a data version, this is added to the tasks table:

:: task_type: DATA_VERSION_ACTIVATE resource_type: resource_id: <DATA_VERSION> data: {}

Security

Similar to neutron-db-manage, only the admins are expected to run midonet-db-manage. While there is no special authentication mechanism implemented for this tool, the only way to run this script is if you have access to the management hosts in the cloud, and preventing unauthorized users from gaining such access is out of this documents scope.

Client

The following command displays the global information about the data, including the write access and the last processed task:

:: midonet-db-manage data-show

The following command sets the Neutron data to be read-only:

:: midonet-db-manage data-readonly

The following command sets the Neutron data to be read-write:

:: midonet-db-manage data-readwrite

The following command displays all the data versions:

:: midonet-db-manage data-version-list

The following command starts data sync to create a new version:

:: midonet-db-manage data-version-sync

The following command activates the specified version. It could be used for the rollback:

:: midonet-db-manage data-version-activate <VERSION_ID>

Documentation

In the Deployment Guide, the following section is added:

- How to initialize the Cluster data when Setting up MidoNet for the first time
- How to initialize the Cluster data when migration from a different Neutron plugin
- Within the upgrade section, how to sync the data from Neutron to Cluster, including how the rollback is accomplished

In the Operational Guide, the following section is added:

- How to sync data between Neutron and Cluster when the data between them become inconsistent due to some operational errors

8.1.4 Gateway Device Management API

MidoNet provides a Neutron extension API called Gateway Device Management to provide device-level gateway management service to the operators. This API is required in order to propagate device connectivity details to enable Midonet to manage VTEP Logical Switch configuration upon Logical Gateway definition. Gateway Device Management API is required for management IP and Port settings. Gateway device should be identified by user driven name in order to correlate it with Logical Gateway entity.

VTEP status, VTEP configuration, such as Tunnel IP are out of the scope of current version of this API. MidoNet currently does not support secure connection settings.

Proposed Change

REST API

GatewayDevice

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the Gateway Device
name	string	CRU	No	User defined device name
management_ip	string (ip addr)	CR	Yes	Manangement IP of device
management_port	int	CR	Yes	Management port of device

Currently, only the VTEP device is supported.

GatewayDevicePeer

To support Active Active Hardware VTEP, MidoNet has an API in place to set peers of VTEP gateway devices.

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the device peering
name	string	CRU	No	User defined peering name
device1_id	string (UUID)	CR	Yes	ID of the first device
device2_id	string (UUID)	CR	Yes	ID of the second device

DB Model

midonet_gateway_devices

Name	Type	Description
id	String	ID of the gateway device
name	String	Name of the gateway device
management_ip	String	Management IP address of the gateway device
management_port	int	Management port of the gateway device

midonet_gateway_device_peers

Name	Type	Description
id	String	ID of the gateway peering
name	String	Name of the gateway peering
device1_id	String	ID of the first gateway device
device2_id	String	ID of the second gateway device

Client

The following command creates a gateway device:

::

neutron gateway-device-create [name NAME] [ip MGMT_IP] [port MGMT_PORT]

The following command updates a gateway device:

::

neutron gateway-device-update DEVICE_ID [name NAME] [ip MGMT_IP] [port MGMT_PORT]

The following command views a gateway device:

:: neutron gateway-device-show DEVICE_ID

The following command deletes a gateway device:

:: neutron gateway-device-delete DEVICE_ID

The following command creates a gateway device peering:

::

neutron gateway-device-peering-create [name NAME] [device1 DEV1] [device2 DEV2]

The following command views a gateway device peering:

:: neutron gateway-device-peering-show DEVICE_PEER_ID

The following command deletes (tears down) a gateway device peering:

:: neutron gateway-device-peering-delete DEVICE_PEER_ID

Alternative Proposal

Instead of managing Gateway devices using REST API, do so using configuration files, which is the approach more familiar to those coming from Neutron background. The REST API approach was chosen to simplify and possibly automate the gateway device management.

8.1.5 Dynamic Routing Service

The MidoNet Neutron plugin, through Neutrons advanced service framework, provides the dynamic routing feature on the provider router.

MidoNet currently only supports BGP but it will also support OSPF in the future. The API design attempts to abstract away the underlying routing protocol.

Problem Description

With the dynamic routing service, an OpenStack cloud would run a routing protocol (for example, BGP) against at least one router in each uplink network provider. By announcing external network hosting floating IP prefixes to those peers, the Neutron network would be reachable by the rest of the internet via both paths. If the link to an uplink provider broke, the failure information would propagate to routers further up the stream, keeping the cloud reachable through the remaining healthy link. Likewise, in such a case, Neutron would eliminate the routes learned through the faulty link from its forwarding table, redirecting all cloud-originated traffic through the healthy link.

Without dynamic routing, the scenario described above would not be possible.

Proposed Change

Three new models are introduced.

RoutingInstance is the top level object that abstracts a dynamic routing service (such as BGP, OSFP). When configured, the dynamic routing service is enabled on the router that it is associated with.

RoutingPeer is the peering configuration applied on the router port that you want to start the peering session from. Since RoutingPeers are associated with ports, there would be multiple RoutingPeers for a given RoutingInstance.

AdvertiseRoute is the route advertised with dynamic routing. In Neutron, Floating IP could be advertised to the outside of OpenStack cloud by creating an AdvertiseRoute object for that CIDR.

In MidoNet, routes learned from the peer are inserted into the routing table of the router, and this proposal does not affect this mechanism.

REST API

RoutingInstance

Attribute Name	Type	POST/ PUT	Re-quired	Description
id	string (UUID)	POST	gener-ated	ID of the routing instance
router_id	string (UUID)	POST	Yes	Router that the routing service is attached to
local_as	int	POST	Yes	Local AS number used in BGP
protocol	string		No	Routing protocol to use. Only BGP supported, so cannot be updated.

Deleting the routing instance deletes all the advertise routes and routing peers. loopback address feature is not included in this spec, but will be added in the future. Also, while the models proposed are meant to abstract away all the dynamic routing protocols, because MidoNet only handles BGP right now, they only include BGP-specific fields.

A router that has a routing instance associated cannot be deleted, and you must delete the routing instance first.

A router could have only one routing instance associated.

RoutingPeer

Attribute Name	Type	POST/ PUT	Re-quired	Description
id	string (UUID)	POST	gener-ated	ID of the routing peer
rout-ing_instance_id	string (UUID)	POST	Yes	Routing instance it is associated with
port_id	string (UUID)	POST	Yes	Port used to connect to the peer
peer_as	int	POST	Yes	Peer AS number used in BGP
peer_address	string	POST	Yes	Peer IP address

Only IPv4 is supported for peer_address. In this proposal, the support for establishing connections with peers that do not have an IP address is not included.

Deleting a routing instance deletes the associated routing peers.

AdvertiseRoute

Attribute Name	Type	POST/ PUT	Re- quired	Desciption
id	string (UUID)	POST	gen- er- ated	Unique Identifier for route configuration
rout- ing_instance_id	string (UUID)	POST	Yes	ID of the routing instance the route is associated with
destination	string	POST	No	Value to compare with the destination IP address of the flow being forwarded Default: 0.0.0.0/32

Only IPv4 is supported for *destination*.

Deleting a routing instance deletes the associated advertise routes.

DB Model

midonet_routing_instances

Name	Type	Description
id	String	ID of the routing instance
router_id	String	ID of the router the routing instance is attached to
local_as	Int	Local AS number
protocol	String	Routing protocol

The only supported value for protocol is BGP, but OSPF will be added in the future.

midonet_routing_peers

Name	Type	Description
id	String	ID of the routing peer
routing_instance_id	String	ID of the routing instance associated
port_id	String	ID of the port for the peer connection
peer_as	Int	Peer AS number used for BGP
peer_address	String	Peer IP address

midonet_advertise_route

Name	Type	Description
id	String	ID of the route
routing_instance_id	String	ID of the routing instance associated
destination	String	destination CIDR to match on

midonet_tasks

New task data types are introduced:

- ROUTING_INSTANCE
- ROUTING_PEER
- ADVERTISE_ROUTE

Security

For this proposal, dynamic routing configuration is limited to admins only.

Client

The following command creates a routing instance:

::

neutron routing-instance-create [router-id ROUTER_ID] [local-as LOCAL_AS]

router-id ROUTER_ID:: ID of the router to associate with

local-as LOCAL_AS:: The local AS number

The following command gets a routing instance:

:: neutron routing-instance-show ROUTING_INSTANCE_ID

ROUTING_INSTANCE_ID:: ID of the routing instance to look up

The following command lists all the routing instances of a tenant:

:: neutron routing-instance-list

The following command associates a routing instance to a router:

::

neutron routing-instance-associate [router-id ROUTER_ID] ROUTING_INSTANCE_ID

ROUTING_INSTANCE_ID:: ID of the routing instance to look up

router-id ROUTER_ID:: ID of the router to associate with

The following command disassociates a routing instance from a router:

:: neutron routing-instance-disassociate ROUTING_INSTANCE_ID

ROUTING_INSTANCE_ID:: ID of the routing instance to look up

The following command deletes a routing instance:

:: neutron routing-instance-delete ROUTING_INSTANCE_ID

ROUTING_INSTANCE_ID:: ID of the routing instance to look up

The following command creates a routing peer:

::

neutron routing-peer-create [routing-instance-id ROUTING_INSTANCE_ID] [port-id PORT_ID] [peer-as PEER_AS] [peer-address PEER_ADDRESS]

routing_instance_id ROUTING_INSTANCE_ID:: ID of the routing instance to create the routing peer for

port-id PORT_ID:: ID of the port to connect to peer from

peer-as PEER_AS:: Peer AS number for BGP

peer-address PEER_ADDRESS:: Peer IP address

The following command deletes a routing peer:

:: neutron routing-peer-delete ROUTING_PEER_ID

ROUTING_PEER_ID:: ID of the routing peer to delete

The following command gets a routing peer:

:: neutron routing-peer-get ROUTING_PEER_ID

ROUTING_PEER_ID:: ID of the routing peer to look up

The following command lists all the routing peers of a tenant:

:: neutron routing-peer-list

The following command creates an advertise route:

::

neutron advertise-route-create [routing-instance-id ROUTING_INSTANCE_ID] [destination DESTINATION]

routing_instance_id ROUTING_INSTANCE_ID:: ID of the routing instance to create the advertse route for

destination DESTINATION:: destination CIDR of the route

The following command delets an advertise route:

:: neutron advertise-route-delete ADVERTISE_ROUTE_ID

ADVERTISE_ROUTE_ID:: ID of the advertise route to delete

The following command gets an advertise route:

:: neutron advertise-route-get ADVERTISE_ROUTE_ID

ADVERTISE_ROUTE_ID:: ID of the advertise route to look up

The following command lists all the advertise routes of a tenant:

:: neutron advertise-route-list

8.1.6 Extra Routes API

This document describes MidoNets implementation of extra routes Neutron extension API.

Problem Description

MidoNet plugin has not implemented extra routes extension API, and without it, MidoNets routing table management feature could not be exposed.

Proposed Change

MidoNet plugin implements the extra routes extension. Current design of extra routes, however, only contains destination and nexthop fields, representing the destination CIDR to match on the packet and the next hop gateway IP address. MidoNet plugin extends the current extra route API to add more fields in the route model to provide more detailed management of the routing table.

Plugin

Add extraroute in the supported_extension_aliases list.

Extend extraroute extension and add a source field, and validate source the same way destination is validated.

REST API

Router

Extra routes extension adds the routes field in the router requests and responses, which is a list of route objects where each route consists of:

Attribute Name	Type	POST/ PUT	Re- quired	Description
destination	string (CIDR)	PUT	Yes	CIDR to match on the packets destination ip Default: 0.0.0.0/0
source	string (CIDR)	PUT	No	CIDR to match on the packets source ip Default: 0.0.0.0/0
nexthop	string (CIDR)	PUT	Yes	IP of the next hop gateway

DB Model

AdvancedExtraRoute

table name: midonet_router_routes

router_routes table in Neutron is used to store the extra routes. In addition, to store the midonet-specific field, source, midonet_router_routes table is introduced:

Name	Type	Description
source	String(64)	Source CIDR to match on
router_id	String(36)	ID of the router the route belongs to

router_id has a foreign key constraint defined for id column of the routers table.

Client

The CLI command to update a router accepts the following new argument:

::

neutron router-update ROUTER_ID routes type=dict list=true [source SOURCE]

source SOURCE: source CIDR of the route

Documentation

Operational Guide must be updated to explain the source field added in the extra route extension.

8.1.7 L2GW API

The L2GW extension API exposes a flexible way to allow implementers to map logical gateways to the physical ones the way they see fit. This API is about logical gateways definition, intentionally leaving out the physical device management.

L2GW API exposes the abstraction of L2 gateway with its interface(s). L2 gateway can expand over several devices with number of interfaces, and each interface can be defined with different list of segmentation ids. Each device is identified by meaningful name, and its possible to add/remove interfaces.

L2GW Binding API allows binding of logical gateway to an overlay network. In the future logical gateway can be bound to the list of virtual networks. Optionally, its possible to specify the default segmentation-id that will be applied to the interfaces for which segmentation id was not specified in l2-gateway-create command.

L2GW API allows to define logical GW that contains group of devices. In single L2GW instance the administrator will define all VTEP devices that should be used as single logical gateway either in Active-Passive or Active-Active mode.

Please refer to the upstream Stackforge project, network-l2gw[1] for the API DB and the client design.

Proposed Change

Plugin

Add l2-gateway extension alias in the supported extension aliases list.

MidoNet plugin should extend L2GatewayMixin class, and implement the CRUD methods for l2 gateway and l2 gateway connection objects.

New tasks representing the L2GW and L2GW Connection are inserted into the tasks table in the appropriate CRUD operations.

REST API

The upstream network-l2gw API is re-used.

DB Model

The upstream network-l2gw DB tables are re-used.

New task types, L2GW and L2GWCONNECTION, are introduced.

Client

The client commmands are the same as those defined in the network-l2gw project.

References

[1] https://github.com/stackforge/networking-l2gw

8.1.8 Port Binding API

In the MidoNet integration with OpenStack, there are two ways a port could be bound:

- When a VM is launched, orchestrated by nova
- When an operator binds a virtual port to a specific host interface

This document describes the design of port binding in MidoNet that implements these use cases.

In the case of operator-specified port binding, there is a special case in which a port is bound to an interface connected to the uplink physical network, but that is not covered in this document.

Problem Description

MidoNet does not currently provide a way to bind ports in Neutron using standard Neutron API.

Proposed Change

Use the existing binding extension API in Neutron to implement port binding in MidoNet.

To bind a port to a specific host interface, an operator makes a create_port API request to Neutron and provide the following binding details:

- binding:host_id: ID of the host to bind the port on
- binding:profile[interface_name]: Name of the interface to bind the port

The host ID is stored in the portbindings table, and the interface name is stored in the midonet_portbindings table. When the host ID and the interface name are supplied in the port creation request, the MidoNet executes the binding within the same API request.

Updating a port with a different binding effectively unbinds the port and re-binds it to the new host interface.

In the case of VM port binding, the workflow is as follows:

1. Nova API makes a create_port request to Neutron API specifying the ID of the host(host_id) where the VM is going to be placed. host_id is stored in Neutrons portbindings table.
2. Neutron generates the tap interface name the same way Nova does (tap + portID up to 14 chars), and stores it in the midonet_portbindings table.
3. On the compute host, mm-ctl script is executed to do the actual binding. mm-ctl adds a port binding task to signal to MidoNet that the binding should occur. This step may change in the future.

For each scenario in which a port binding occurs, the plugin inserts a PORTBINDING task with resource_id set to the ID of the port getting bound.

The actual mechanism in which the binding takes place inside MidoNet is outside the scope of this document.

REST API

An example of a port binding attributes in the request to create a port is:

```
{
  "binding:host_id": "HOST_ID",
  "binding:profile": {"interface_name": "eth0"}
}
```

DB Model

midonet_portbindings

Name	Type	Description
port_id	UUID	ID of the port
interface_name	String	Name of the interface to bind the port

port_id is the primary key and has a foreign key constraint to the id column of the ports table.

Client

The following command creates a port with port binding attributes:

```
neutron port-create [--binding:host_id HOST_ID]
                    [--binding:profile if_name=IF_NAME]
```

binding:host_id HOST_ID: ID of the host to bind the port on

binding:profile if_name=IF_NAME: Name of the interface to bind the port to

8.1.9 Uplink Network API

MidoNet plugin implements the provider network Neutron extension API, and makes slight modification to the port binding extension API implementation to simplify the uplink port configuration of the edge routers.

Problem Description

The edge routers are virtual routers that interface with the uplink routers outside of the cloud. Thus, the ports on these virtual routers must be bound to the interfaces on the edge hosts of the MidoNet deployment. The MidoNet Neutron Plugin, however, does not offer any API to create a port on the edge router. Furthermore, Neutron requires that all ports are created on a network, so the concept of having a port directly on a router must be emulated using the Neutron constructs.

Proposed Change

Because Neutron does not allow creation of a port directly on a router, a network must be created for any port to exist. This network must be flagged as a special network to indicate that it is an uplink network so that MidoNet would know that the ports on this network need be treated as router ports bound directly on the physical hosts (as opposed to normal network ports). Normal networks cannot be used for this purpose because in the future MidoNet will support binding of ports on any network, which must be differentiated from binding ports on the edge router.

To mark these networks as uplink networks, implement the provider network Neutron extension API to map virtual networks to the uplink networks in the underlay. Once a virtual network is mapped to a physical network, you could create a port on this network with binding details to control exactly which host interface that the port should be bound to.

The provider network extension lets you map a physical network to a virtual network by associating the virtual network with attributes describing the physical network. One of the attributes is called network_type which could be LOCAL, FLAT, VXLAN, VLAN and GRE. For the purpose of uplink network mapping, only the LOCAL network type is accepted by the plugin because it requires the least amount of extra configurations. To create an uplink network, only network_type needs to be specified.

MidoNet currently does not support binding a vlan tagged interface, so the VLAN type cannot be used, but it will be supported in the future. Likewise, GRE and VXLAN are not currently supported for binding.

The uplink set up workflow is as follows:

1. The operator creates a Neutron network mapped to each uplink physical network. The following field should be set for the network to create:
   - **provider:network_type => Type of the uplink network. Only LOCAL** type is accepted.
2. On these networks, create a port for each interface on the host nodes that are connected to the uplink networks. Supply these additional binding details for each port created as follows:
   - binding:host_id => ID of the host to bind the port on
   - binding:profile[if_name] => Name of the interface to bind to

   The host IDs and interface names can be fetched from the agents extension API.

3. For each uplink network port created, execute router interface add standard Neutron API to connect the uplink network and the edge router. This triggers the binding of the port to the physical interface. The detail of how the binding is achieved is outside of this documents scope.

There are no changes required in the REST API, DB models or client since only the standard Neutron extension API is invoked. Please refer to the provider network extension API document[1] for more details on the workflow.

Database

The only visible change is that the network and port tasks contain new fields:

Network:

:: provider:network_type: LOCAL

Port:

:: binding:host_id: HOST_ID, binding:profile: {if_name: eth0}

Plugin

The provider extension alias is added to the MidoNet plugin. When looking up a network, the provider extension attributes must be added to the network API request and response.

Also, although the plugin already supported binding extension API, the binding does not include, binding:host_id and binding:profile, that are added to the port API request and response.

Documentation

The setup of the uplink networks must be described in the Deployment Guide and/or the Operational Guide.

Alternative Proposal

Instead of using the provider network extension to specify the binding information for the edge hosts, store the binding information in the MidoNet agent configuration. The reason for choosing the proposed approach is that there was no concrete design agreed by all stakeholders on the agent configuration approach.

References

[1] http://docs.openstack.org/admin-guide-cloud/content/provider_api_workflow.html

8.1.10 Task Management

Neutron and the MidoNet cluster, which is the distributed network configuration storage service of MidoNet, communicate via the tasks database table. A task represents a single Neutron API operation that the cluster translates into lower level MidoNet concepts. This table stores as tasks all the API write requests as well as Neutrons global configurations specified in neutron.conf during the initialization stage. They are processed by the cluster in the order inserted. This document describes new commands of midonet-db-manage tool that provide tasks table management functionalities.

Problem Description

While the tasks table provide a reliable communication channel between Neutron and MidoNet, it lacks the following features:

- Ability to view/filter the processed and unprocessed tasks for debugging
- Ability to clean up the processed tasks. You should be able to delete all the processed tasks.

Proposed Change

New commands are added to midonet-db-manage tool to provide better visibility into the tasks table as well as a way to safely clean up the processed tasks. They do not belong in the Neutron API because they have very little to do with network management. Note that only the manual clean up of the tasks is described in this proposal, and there will be a separate proposal to address the automatic clean-up.

To implement the commands, the last processed task ID, which is currently maintained by the cluster, needs to be also stored in the Neutron DB so that midonet-db-manage could use this value to differentiate the processed and unprocessed tasks. The task IDs are auto-incremented integer field, and the last processed task ID indicates the latest task that was processed by the cluster.

The cluster processes a transaction consisting of one or more tasks atomically, and in the same transaction, the ID of the last processed task is stored. The cluster then stores this task ID in Neutron DBs midonet_task_state table. If the Neutron DB update fails due to a temporary resource issue, such as network disruption, the cluster will re-sync in the next successful task processing. It is guaranteed that the last processed task ID in the cluster never trails that of Neutron because the cluster always updates its last processed task ID before it updates the Neutron DBs table. This means that there may be tasks that are not yet marked as processed by the cluster in the Neutron DB that have actually already been processed. However, the reverse cannot be true.

Once the last processed task ID is made available to Neutron, midonet-db-manage could easily separate the processed and unprocessed tasks by querying the tasks table filtered by this value. While it may be useful to filter the tasks based on other criteria, such as tenant ID, resource ID, and resource type, but such feature will be addressed in a different proposal to not over-complicate.

REST API

None

DB Model

midonet_task_state

Name	Type	Description
id	Int	The primary key useful to idenfity the single row (Set to 1)
last_processed_id	Int	The last processed task ID (default NULL)
updated_at	DateTime	Time of the last update (default NULL)

midonet_task_state is a singe-row table representing the current state of the tasks table. It is created and data initialized by the alembic migration script. The single row is created during the alembic migration with default values. The cluster updates this table when it completes processing a particular task.

last_processed_id has foreign key reference to the tasks tables id column.

id is used by the cluster to identify the single row. Also, sqlalchemy requires that there is a primary key column. The id of the single row is set to 1.

Security

Similar to neutron-db-manage, only the admins are expected to run midonet-db-manage. While there is no special authentication mechanism implemented for this tool, the only way to run this script is if you have access to the management hosts in the cloud, and preventing unauthorized users from gaining such access is out of this documents scope.

Client

The following command lists the tasks:

:: midonet-db-manage task-list [-u]

-u, unprocessed:: Show only the processed tasks

The following command deletes the processed tasks:

:: midonet-db-manage task-clean

When the processed tasks are deleted, the last_processed_id is reset to NULL. Note that there is no command to delete unprocessed tasks because such command is dangerous, and will be addressed separately when the upgrade/import feature is designed. If that must be done, then the operator must do so directly from the sql client.

The following command displays the state of the resources based on the tasks so that you can see which ones should (or will) exist. This is implemented with best effort since the tasks table may not contain the entire history:

:: midonet-db-manage task-resource [-p]

-p, processed:: Calculate based on only the processed tasks

8.2 Mitaka specs

8.2.1 BGP Speaker Insertion Model on Routers

This spec describes an extension to associate a BGP speaker to a router.

For detailed explanations of the BGP implementation of networking-midonet, refer to the BGP Operational Guide [1].

Problem Description

MidoNet BGP model does not match the bgp extension model of Neutron in some critical ways. Namely, in MidoNet, BGP must be configured on a router whereas in Neutron, BGP is configured independently. There is no way to associate a BGP speaker to a router in the Neutrons bgp extension model.

Proposed Change

In order to have MidoNet implement bgp extension API of Neutron, bgp-speaker-router-insertion vendor extension API is defined to track the associations between BGP speakers and routers.

For each BGP speaker, exactly one router is associated. The IP address used by the BGP speaker is the IP address on the router port that is on the same subnet as the BGP peer IP.

With this mode, the following operations no longer become applicable:

- add_gateway_network
- delete_gateway_network

Invoking these operations on a BGP speaker that has a router associated results in an error.

Also, the following fields no longer become applicable:

- advertise_floating_ip_host_routes
- advertise_tenant_networks

The values set in these fields are ignored.

Since bgp-speaker-router-insertion is a vendor extension, it works only with networking-midonet as the plugin. However, if this extension becomes part of neutron-dynamic-routing project, it will be expected to work with the reference implementation, dr-agents. There should be nothing in this design that should interfere with the current implementation of dr-agents, including its HA capabilities.

Data Model Impact

bgp_speaker_router_associations table is created:

Attribute name	Type	Description
bgp_speaker_id	uuid	BGP speaker id
router_id	uuid	Associated router

bgp_speaker_id is the primary key of the table. Both bgp_speaker_id and router_id have foreign key constraints set to bgp_speakers and routers tables, respectively.

REST API Impact

```
RESOURCE_ATTRIBUTE_MAP = {
  'bgp-speakers': {
      'logical_router': {'allow_post': True, 'allow_put': False,
                         'validate': {'type:uuid': None},
                         'is_visible': True, 'default': None},
  }
}
```

Security Impact

None

Other End User Impact

Neutron CLI provides support for logical_router field as follows:

```
neutron bgp-speaker-create [--tenant-id TENANT_ID] --local-as LOCAL_AS
                           [--ip-version {4,6}]
                           [--logical-router ROUTER]
                           NAME

--logical-router ROUTER
    Router ID or name to associate BGP speaker with.
```

Performance Impact

None

IPv6 Impact

None

Other Deployer Impact

None

Developer Impact

None

Documentation Impact

MidoNet Operational Guide will be updated to include the new attribute added to the BGP speaker model.

REFERENCES

[1] https://docs.google.com/document/d/1cNIkY6zC9djKo2laFKN8JvAD8oHK87uNZMvt_81dQ7E/

8.2.2 Border GW API

https://blueprints.launchpad.net/networking-midonet/+spec/border-gw-api-for-midonet

Border GW is term for enhanced L2GW that should enable inter-site connectivity.

Current L2GW API

The L2GW extension API defined in networking-l2gw[1] exposes the abstraction of L2 gateway with its interface(s). L2 gateway can expand over several devices with number of interfaces, and each interface can be defined with different list of segmentation ids. Each device is identified by meaningful name, and its possible to add/remove interfaces. L2GW Binding API allows binding of logical gateway to an overlay network. L2GW API allows to define logical GW that contains group of devices. In single L2GW instance the administrator will define all VTEP devices that should be used as single logical gateway either in Active-Passive or Active-Active mode.

Border GW API

Border GW funtionality is required for multi-site deployment. It should connect Tenants overlay networks across sites. It will use dedicated Tunnel that connects Tenant networks across sites. L2GW abstract model fits very well into Border GW case. Current L2GW API imposes limitation that should be released in order to support Border GW use case. It should be possible to create L2GW specifing device and segmentation_id, without specific interface name. Interface name is not necessary for the Border GW use case since scope of segmentation_id is global.

In order to support Border GW as well as L2GW abstraction API, the gateway device should be defined in the MidoNet model. Gateway Device should be defined via device-management API[2] in order to be confirmed by L2GW API at L2GW instance creation.

[1] https://github.com/openstack/networking-l2gw
[2] https://blueprints.launchpad.net/networking-midonet/+spec/gw-device-api

Proposed Change

Plugin

MidoNet l2gw (border gw) driver should be added to be loaded and used by l2gw service plugin to support l2-gateway extension in MidoNet.

MidoNet l2gw driver should provide its own L2Gateway API validation method to apply Border GW API validation for CRUD methods for l2 gateway and l2 gateway connection objects. Logical Gateway device will be defined by using device_id of the Gateway Device and segmentation_id.

REST API

The upstream networking-l2gw API is re-used.

To create logical border gateway, following format can be used:

JSON Request

```
POST /v2/l2-gateways
Content-Type: application/json
{"l2_gateway": {"name": "<gateway-name>",
                "devices": [{"device_id": "<device-id1>",
                             "segmentation-id": <seg-id1>}]
                            }}
```

Response:

```
{"l2_gateway": {"name": "<gateway-name>",
                "tenant_id": "7ea656c7c9b8447494f33b0bc741d9e6",
                "devices": [{"device_id": "<device-id1>",
                             "segmentation-id": <seg-id1>}],
                "id": "d3590f37-b072-4358-9719-71964d84a31c"}}
```

DB Model

The upstream networking-l2gw DB tables are re-used.

Client

Currenlty supports MidoNet l2gw creation CLI that is different from that of the upstream networking-l2gw.

The following command create a l2 gateway:

```
neutron midonet-l2-gateway-create GATEWAY-NAME [--device device_id=DEVICE_
↪ID,segmentaion_id=SEGMENTAION_ID]
```

Other Deployer Impact

If L2 gateway service is to be enabled, then it is required to configure the L2 gateway service plugin in neutron.conf.

/etc/neutron.conf: service_plugins=l2gw

Provider driver should be specified, service_provider=L2GW:l2gw:<driver>

References

8.2.3 Gateway Device Management API update for Router Peering

https://blueprints.launchpad.net/networking-midonet/+spec/gw-device-api

MidoNet provides a Neutron extension API called Gateway Device Management to provide device-level gateway management service to the operators. This API is required in order to propagate device connectivity details to enable Midonet to manage VTEP Logical Switch configuration upon Logical Gateway definition. In order to support Router Peering and Direct Connect use cases following definition in[2], Overlay VTEP Router device is supported by MidoNet.[3] While for the routing functionality this device is managed as traditional neutron Router, it should be possible for operator (or Orchestration Layer) to enable its VTEP functionality. While for HW VTEP Device this API is used for management IP and Port settings, for Overlay VTEP Router Device it is used to enable Router with VTEP Logical Switch management capability.

VTEP Tunnel IPs and Remote MAC Table management is currently supported for the router_vtep type of gateway device only.

Other VTEP configurations as well as VTEP device status are out of the scope of the current version of this API.

Gateway device should be identified by the user driven name in order to correlate it with Logical Gateway entity.

Proposed Change

The following section provides details of the enhanced version of the device management spec[1] with support for both HW VTEP and Overlay VTEP Router as gateway devices.

REST API

GatewayDevice

[2] https://docs.google.com/presentation/d/1b_lmDLF-i2rZlOGnZfYwZgim3W2BNf2rLWao3aULHC4/edit#slide=id.p
[3] https://docs.google.com/document/d/1QMcQ33L76c_igBomOAeH9yiiOJwJQ8QK7ZVV8-jrPVA/edit#
[1] https://raw.githubusercontent.com/openstack/networking-midonet/master/specs/kilo/device_management.rst

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the Gateway Device
name	string	CRU	No	User defined device name
tenant_id	string	CR	Yes	Tenant ID of gateway Device object owner
management_ip	string (ip addr)	CR	No	Management IP to the device. Defaults to None.
management_port	int	CR	No	Management port to the device. Defaults to None.
management_protocol	string	CR	No	Management protocol to manage the device: ovsdb or none. If management ip and port are specified, defaults to ovsdb. Otherwise to none.
type	string	CR	No	Type of the device: hw_vtep or router_vtep. Defaults to hw_vtep
resource_id	string (UUID)	CR	No	Resource UUID or None (for type router_vtep will be router UUID)
tunnel_ips	string (list of ip addrs)	CRU	No	IP addresses on which gateway device originates or terminates tunnels.
remote_mac_entries	list of entries	CR	No	Mapping of MAC addresses to the tunnel IP addresses of the corresponding VTEP

Currently, only the HW VTEP device and Router VTEP are supported.

Remote MAC Table entries are managed as sub-resource of the gateway_device.

RemoteMac

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the remote mac entry
mac_address	string	CR	Yes	MAC address
vtep_address	string	CR	Yes	Remote VTEP Tunnel IP to be used to reach this MAC address
segmentation_id	int	CR	Yes	VNI to be used to reach this MAC address

REST API Impact

Proposed attributes:

```
RESOURCE_ATTRIBUTE_MAP = {
    'gateway_devices': {
        'id': {'allow_post': False, 'allow_put': False,
               'validate': {'type:uuid': None},
               'is_visible': True, 'primary_key': True},
        'name': {'allow_post': True, 'allow_put': True,
```

(continues on next page)

(continued from previous page)

```
                'is_visible': True, 'default': '',
                'validate': {'type:string': None}},
        'tenant_id': {'allow_post': True, 'allow_put': False,
                      'required_by_policy': True,
                      'is_visible': True},
        'management_ip': {'allow_post': True, 'allow_put': False,
                'is_visible': True, 'default': ''},
        'management_port': {'allow_post': True, 'allow_put': False,
                'is_visible': True, 'default': ''}'
        'management_protocol': {'allow_post': True, 'allow_put': False,
                'is_visible': True, 'default': ''}'
        'type': {'allow_post': True, 'allow_put': False,
                'is_visible': True, 'default': 'hw_vtep'},
        'resource_id': {'allow_post': True, 'allow_put': False,
                'is_visible': True, 'default': None}'
        'tunnel_ips': {'allow_post': True, 'allow_put': True,
                'is_visible': True, 'default': ''},
        'remote_mac_entries': {'allow_post': False, 'allow_put': False,
↪'is_visible': True},
    },
}


SUB_RESOURCE_ATTRIBUTE_MAP = {
    'remote_mac_entries': {
        'parent': {'collection_name': 'gateway_devices',
                   'member_name': 'gateway_device'},
    'parameters': {
        'id': {
            'allow_post': False, 'allow_put': False,
            'validate': {'type:uuid': None},
            'is_visible': True}},
        'tenant_id': {'allow_post': True, 'allow_put': False,
                      'required_by_policy': True,
                      'is_visible': True},
        'vtep_address': {
            'allow_post': True, 'allow_put': False,
            'is_visible': True, 'default': None,
            'validate': {'type:ip_address': None}},
        'mac_address': {
            'allow_post': True, 'allow_put': False,
            'is_visible': True,
            'validate': {'type:mac_address':None}},
        'segmentation_id': {
            'allow_post': True, 'allow_put': False,
            'is_visible': True,
            'validate': {'type:non_negative': None}},
    }
}
```

Sample request/response:

Update Remote MAC Entry Request:

```
POST /v2.0/gw/gateway_devices/46ebaec0-0570-43ac-82f6-60d2b03168c4/remote_
↪mac_entries
```

(continues on next page)

(continued from previous page)

```
{
    "remote_mac_entry: {
        "mac_address": "10:20:30:40:50:60",
        "vtep_ip": "192.168.34.5",
        "segmentation_id": 304
    }
}


Response:
{
    "remote_mac_entry": {
        "id": "5f126d84-551a-4dcf-bb01-0e9c0df0c793",
        "mac_address": "10:20:30:40:50:60",
        "vtep_ip": "192.168.34.5",
        "segmentation_id": 304
    }
}
```

DB Model

midonet_gateway_devices

Name	Type	Description
id	String	ID of the gateway device
name	String	Name of the gateway device
type	String	Type of the gateway device (hw_vtep or router_vtep)

midonet_gateway_hw_vtep_devices

Name	Type	Description
device_id	String	ID of the gateway device
management_ip	String	Management IP address of the gateway device
management_port	int	Management port of the gateway device
management_protocol	String	Management protocol of the gateway device

midonet_gateway_overlay_router_devices

Name	Type	Description
device_id	String	ID of the gateway device
resource_id	String	Router UUID enabled as gateway device

midonet_gateway_tunnel_ips

Name	Type	Description
device_id	String	ID of the gateway device
tunnel_ip	String	Tunnel IP to originate/terminate traffic

midonet_gateway_remote_mac_table

Name	Type	Description
id	String	ID of the entry
device_id	String	ID of the gateway device
mac_address	String	MAC address to be reached
vtep_address	String	VTEP IP address to reach MAC address
segmentation_id	int	VNI to reach the MAC address

Client

The following command enables a gateway capabilities on the router device:

```
neutron gateway-device-create [--name NAME] [--type router_vtep] [--
↪resource-id UUID]
```

The following command creates a HW VTEP gateway device:

::

neutron gateway-device-create [name NAME] [type hw_vtep] [ip MGMT_IP] [port MGMT_PORT]

The following command updates a gateway device:

```
neutron gateway-device-update GW_DEVICE_ID [--name NAME]
```

The following command lists gateway devices:

```
neutron gateway-device-list
```

The following command views a gateway device:

```
neutron gateway-device-show GW_DEVICE_ID
```

The following command deletes a gateway device:

```
neutron gateway-device-delete GW_DEVICE_ID
```

References

8.2.4 Logging API for firewall-rules

This document describes MidoNets implementation of logging for firewall rules.

FWaaS v2.0[2] will be implemented in newton, However, first of all we implement MidoNet for mitaka with FWaaS v1.0 because development schedule between networking-midonet and neutron does not match.

Note that eventually we aim to unify this spec and spec for upstream[1]. Thus, this spec is written following spec for upstream. In addition, we should keep on watch spec for upstream because the spec have not completed yet.

[2] https://github.com/openstack/neutron-specs/blob/master/specs/newton/fwaas-api-2.0.rst

[1] http://docs-draft.openstack.org/09/203509/41/check/gate-neutron-specs-docs/34a11fa//doc/build/html/specs/newton/logging-API-for-security-group-rules.html

Problem Description

Operator wants to

- monitor network traffic in system to detect illegal traffic, or to solve unexpected communication error across L3.
- pass audit for system.

Tenant wants to

- monitor network traffic in tenant to detect illegal traffic, or to solve unexpected communication error across L3.
- pass audit for tenant

In Neutron, traffic accepted/denied on routers are managed by FWaaS. However, logging is currently a missing feature in FWaaS. Thus, requirements above cannot be satisfied.

Proposed Change

The scope of this spec:

- logging API for operator and tenant.
- logging format for operator and tenant.

How tenants can consume outputted logs are out of scope. Logging format that will be sent to tenants are out of scope. Where the logs are generated is also out of scope.

Plugin

Add logging-resource extension alias in the supported extension aliases list.

MidoNet plugin implement the CRUD methods for logging resource and firewall log objects.

Expected API behavior

The events related to firewall rules will collect:

(1) ACCEPT event

(2) DROP event

Operators and tenants can specify what kind of events they want to log. Note that only logging of firewall rules that are created explicitly are gathered.

(1) ACCEPT/DROP or ALL (collect all ACCEPT/DROP events of firewalls).

(2) firewall uuid.

REST API

In this spec, two resources are newly defined.

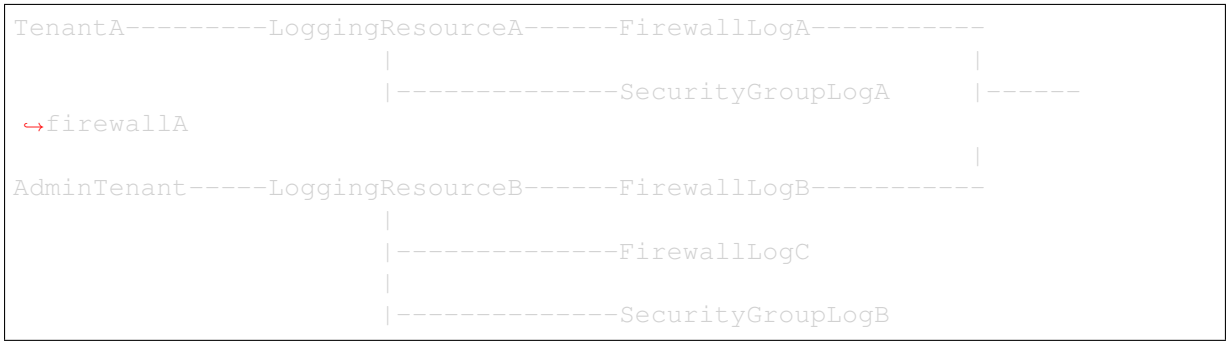
LoggingResource is root resource for logging. This model defines policy of logging. e.g. how can we see logs, where logs are going to be collected.

FirewallLog is logging resource for specific resource. Though this spec proposes only FWaaS, many logging resources for specific resource will be supported in the future. (e.g. security group)

In addition, firewall can be associated with multiple logging resource only when the tenant_ids are different to allow operators and tenants to specify same firewall as logging resource. This allows operators to gather log in all of system.

Note that there is no difference in outputted format between tenants logging resource and operators logging resource. If operator wants to operate description above, some validations are needed to solve why the log is outputted.

The rough sketch that contains future consideration is following;



LoggingResource

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the LoggingResource
name	string	CRU	No	Defined LoggingResource name
description	string	CRU	No	Description for the LoggingResource
tenant_id	string	CR	No	Tenant ID of LoggingResource object owner
enabled	string	CRU	No	Enable/disable for the LoggingResource. log is gathered only when this flag is enable.

FirewallLog

Attribute Name	Type	CRUD	Required	Description
id	string (UUID)	CR	generated	ID of the FirewallLog
description	string	CRU	No	Description for FirewallLog
tenant_id	string	CR	No	Tenant ID of FirewallLog object owner
fw_event	string	CRU	No	Event of firewall. ACCEPT/DROP/ALL can be specified. ALL is set as default.
firewall_id	string (UUID)	CR	Yes	ID of firewall instance

API list is as follows. Note that api path prefix logging may conflict with upstream. However, we keep this prefix to keep consitency with upstream. To avoid confusion, we should do following things.

- Separate our plugin configuration from upstream.
- Note to user that not to use both plugins together.

Object	URI	Type
logging-resource	/logging/logging_resources	POST
logging-resource	/logging/logging_resources	GET
logging-resource	/logging/logging_resources/{id}	GET
logging-resource	/logging/logging_resources/{id}	DELETE
logging-resource	/logging/logging_resources/{id}	PUT
firewall-log	/logging/logging_resources/{id}/firewall_logs	POST
firewall-log	/logging/logging_resources/{id}/firewall_logs	GET
firewall-log	/logging/logging_resources/{id}/firewall_logs/{id}	GET
firewall-log	/logging/logging_resources/{id}/firewall_logs/{id}	DELETE
firewall-log	/logging/logging_resources/{id}/firewall_logs/{id}	PUT

REST API Examples

To Create a LoggingResource to manage security event log, following API can be used:

JSON Request

```
POST /v2.0/logging/logging_resources
{
    "logging_resource": {
        "name": "firewall_log",
        "description": "Get traffic flow of firewall",
        "enabled": True
     }
}
```

Response:

```
Response:
{
   "logging_resource": {
       "id": "46ebaec0-0570-43ac-82f6-60d2b03168c4",
```

(continues on next page)

(continued from previous page)

```
        "tenant_id": "8d4c70a21fed4aeba121a1a429ba0d04",
        "name": "firewall_log",
        "description": "Get traffic flow of firewall",
        "enabled": True
    }
}
```

To Create a FirewallLog to collect security event of the firewall, following API can be used:

JSON Request

```
POST /v2.0/logging/logging_resources/46ebaec0-0570-43ac-82f6-60d2b03168c4/
↪firewall_logs
{
    "firewall_log": {
        "description": "Collecting all traffic passing the firewall",
        "fw_event": "ALL",
        "firewall_id: "21aeda2a-a52f-4e81-9e64-7edeb59fa25b"
    }
}
```

Response:

```
{
"firewall_log": {
    "id": "5f126d84-551a-4dcf-bb01-0e9c0df0c793",
    "tenant_id": "8d4c70a21fed4aeba121a1a429ba0d04",
    "logging_resource_id": "46ebaec0-0570-43ac-82f6-60d2b03168c4",
    "description": "Collecting all traffic passing the firewall",
    "fw_event": "ALL",
    "firewall_id": "21aeda2a-a52f-4e81-9e64-7edeb59fa25b"
    }
}
```

REST API Impact

The new resources:

```
LOGGING_PREFIX = '/logging'
FW_EVENT_ACCEPT = 'ACCEPT'
FW_EVENT_DROP = 'DROP'
FW_EVENT_ALL = 'ALL'
FW_EVENTS = [FW_EVENT_ACCEPT, FW_EVENT_DROP, FW_EVENT_ALL]
LOG_COMMON_FIELDS = {
    'id': {'allow_post': False, 'allow_put': False,
           'validate': {'type:uuid': None},
           'is_visible': True, 'primary_key': True},
    'tenant_id': {'allow_post': True, 'allow_put': False,
                  'required_by_policy': True, 'is_visible': True},
    'logging_resource_id': {'allow_post': False, 'allow_put': False,
                            'is_visible': True}
}

RESOURCE_ATTRIBUTE_MAP = {
```

(continues on next page)

(continued from previous page)

```
    'logging_resources': {
        'id': {'allow_post': False, 'allow_put': False,
               'validate': {'type:uuid': None}, 'is_visible': True,
               'primary_key': True},
        'tenant_id': {'allow_post': True, 'allow_put': False,
                      'required_by_policy': True, 'is_visible': True},
        'name': {'allow_post': True, 'allow_put': True,
                 'validate': {'type:string': attr.NAME_MAX_LEN},
                 'default': '', 'is_visible': True},
        'description': {'allow_post': True, 'allow_put': True,
                        'validate': {'type:string': attr.LONG_DESCRIPTION_
↪MAX_LEN},
                        'default': '', 'is_visible': True},
        'enabled': {'allow_post': True, 'allow_put': True,
                    'is_visible': True, 'default': False,
                    'convert_to': attr.convert_to_boolean},
        'firewall_logs': {'allow_post': False, 'allow_put': False,
                          'is_visible': True}
    }
}

SUB_RESOURCE_ATTRIBUTE_MAP = {
    'firewall_logs': {
        'parent': {'collection_name': 'logging_resources',
                   'member_name': 'logging_resource'},
        'parameters': dict((LOG_COMMON_FIELDS),
                           **{
                             'description': {
                                 'allow_post': True, 'allow_put': True,
                                 'validate': {'type:string': None},
                                 'default': None, 'is_visible': True},
                             'firewall_id': {
                                 'allow_post': True, 'allow_put': False,
                                 'is_visible': True,
                                 'validate': {'type:uuid': None}},
                             'fw_event': {
                                 'allow_post': True, 'allow_put': True,
                                 'is_visible': True,
                                 'validate': {'type:values': FW_EVENT},
                                 'default': 'ALL'}
                           })
    },
}
```

Logging format

Following items can be shown as follows. Eventually, we catch up neutron behavior that agent collects logs and sends logs to specified location from user. Therefore, outputted items should be unified with neutron after supporting function in neutron.

Item	Description
tenant_id	Tenant ID of targeted firewall
timestamp	Time of the event is happened The time is based on ISO8601, time zone is UTC
firewall UUID	UUID of neutron firewall
firewall rule UUID	UUID of neutron firewall rule
router UUID	UUID of neutron router
source IP address	Source IP address of the communication
destination IP address	Destination IP address of the communication
source L4 port	Source L4 port of the communication
destination L4 port	Destination L4 port of the communication
protocol	IANA protocol number
action	ACCEPT/DROP

Logging out location

Currently, operators can only access directly to file on host that midolman is running to consume log-data. File location has format: /var/log/midolman/logging/fw-<firewall-log-uuid>.log How generated log files are sent to tenants is up to the operator. Backend implementation and/or log collector are expected to handle log rotation. In the case with MidoNet, log rotation policy can be configured using its configuration tool.

DB Model impact

To avoid competition of table name with upstream, we add specific initial to head of table names. Note that upstream DB will be reused in newton or later and DB in networking-midonet will be deleted.

The LoggingResource model has the following attributes:

midonet_logging_resources

Attribute Name	Type	Access	Default Value	Validation/ Conversion	Description
id	uuid	RO	generated	uuid	Identity
tenant_id	uuid	RO	N/A	uuid	Id of tenant that created this LoggingResource
name	string	RW	N/A	none	LoggingResource name
description	string	RW	N/A	none	LoggingResource description
enabled	bool	RW	False	Boolean	Enable/disable log

The FirewallLog model would look like:

midonet_firewall_logs

Attribute Name	Type	Access	Default Value	Validation/ Conversion	Description
id	uuid	RO	generated	uuid	Identity
logging_resource_id	uuid	RO	N/A	uuid	LoggingResource UUID
tenant_id	uuid	RO	generated	uuid	Tenant creates logging
description	string	RW	N/A	none	FirewallLogging description
fw_event	enum	RW	N/A	enum	ACCEPT/DROP & ALL (collect all ACCEPT/DROP events)
firewall_id	uuid	RW(No update)	N/A	uuid	Firewalls UUID is enabled logging

Quota

Firewall log is managed by Quota. Default value of firewall log is 10 that is same number as firewall. Basically, both Quota value for firewall and firewall log should be aligned.

CLI Impact

Additional methods will be added to python-neutronclient to create, update, delete, list, get logging resource and firewall logging.

Checking support resource logging

For logging resource:

```
neutron logging-create --name <logging-resource-name>
                       [--enable <True/False>]
                       [--description <logging-resource-description>]
neutron logging-list
neutron logging-update <logging-resource-name-or-id>
                       [--name ...]
                       [--description ...]
                       [--enable <True/False>]
neutron logging-show <logging-resource-name-or-id>
neutron logging-delete <logging-resource-name-or-id>
```

For firewalls logging:

```
neutron logging-firewall-create <logging-resource-name-or-id> <firewall-id>
                                [--description <firewall-log description>]
                                [--fw-event <ACCEPT/DROP/ALL>]
neutron logging-firewall-list <logging-resource-name-or-id>
neutron logging-firewall-update <logging-resource-name-or-id> <firewall-
↪log-id>
                                [--description ...]
                                [--fw-event ...]
```

(continues on next page)

(continued from previous page)

```
neutron logging-firewall-show <logging-resource-name-or-id> <firewall-log-
↪id>
neutron logging-firewall-delete <logging-resource-name-or-id> <firewall-
↪log-id>
```

Other Deployer Impact

Set quota for firewall log in quotas section of neutron.conf.

quota_firewall_log = 10

References

8.2.5 Floating IP via router interfaces

This spec describes an extension to associate floating IPs via router interfaces, rather than the router gateway port.

Problem Description

For some use cases[1], it can be useful to make floating IP translation happens on non-gateway router interfaces.

Proposed Change

Introduce router-interface-fip extension, which allows users to associate floating IPs via router interfaces.

Consider the topology like the following diagram. This extension allows to associate floating-ip-B to fixed-ip-X.

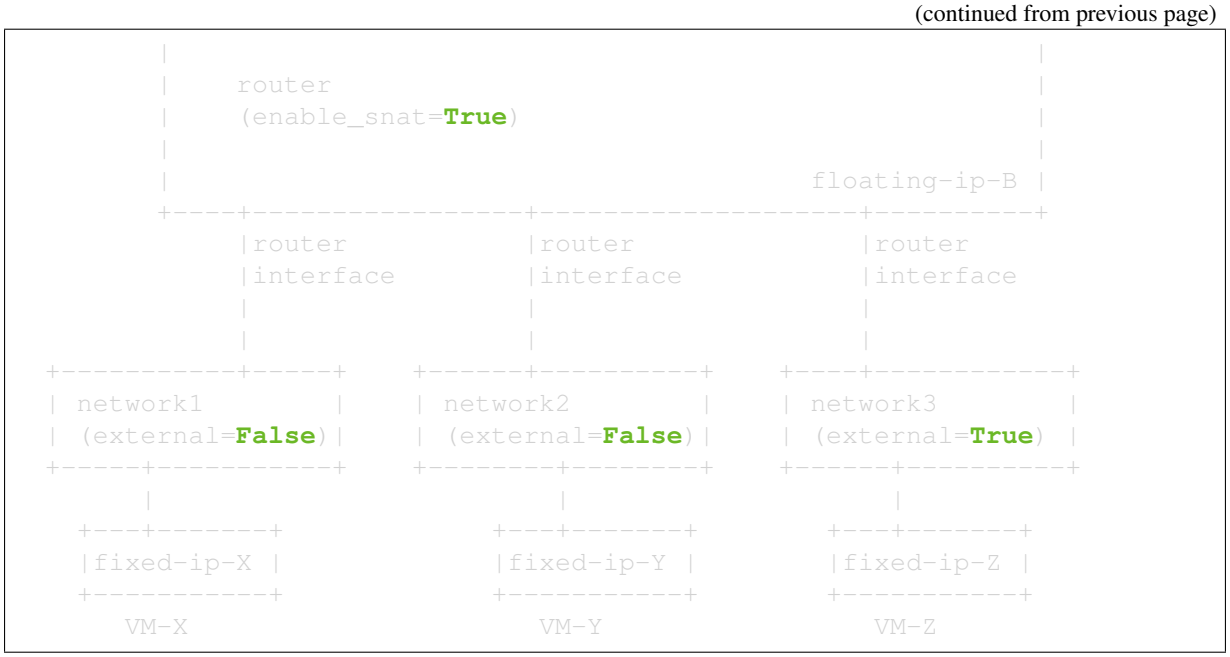
- floating-ip-A is created on network4.
- floating-ip-B is created on network3.
- Both of floating-ip-A and floating-ip-B are associated to fixed-ip-X.
- fixed-ip-Y and fixed-ip-Z dont have any floating ip associations.

```
+-----------------------+
|  network4             |
|  (external=True)      |
+-------------------+---+
                    |
                    |
                    |router gateway port
                    |(its primary address is gw-ip)
          +---------+-------------------------------------------+
          |      floating-ip-A                                  |
```

(continues on next page)

[1] https://docs.google.com/presentation/d/1-v-bCsaEphyS5HDnhUeI1KM5OssY-8P4WMpQZsOqSOA/edit#slide=id.g1232f85657_0_63

(continued from previous page)



In case multiple floating ip addresses are associated to a fixed ip address, a datapath should be careful which floating ip to use for SNAT:

- If theres a floating ip associated via the egress port, either the router gateway port or a router interface, it should be used. For example, in the case of the above diagram, if VM-X sends a packet fixed-ip-X -> fixed-ip-Z, floating-ip-B, rather than floating-ip-A, should be used.
- Otherwise, if theres a floating ip associated via the router gateway port, it should be used. For example, in the case of the above diagram, if VM-X sends fixed-ip-X -> fixed-ip-Y, floating-ip-A should be used.
- Otherwise, the datapath can choose arbitrary one.

A few interesting cases:

- If VM-Y sends a packet fixed-ip-Y -> floating-ip-A, its translated to gw-ip -> fixed-ip-X by the router and VM-X will receive it. This behaviour is not specific to this extension. See bug 1428887[3] for the reason of the SNAT.
- If VM-Y sends a packet fixed-ip-Y -> floating-ip-B, its translated to gw-ip -> fixed-ip-X by the router and VM-X will receive it. However, its return traffic fixed-ip-X -> gw-ip will be translated to floating-ip-A -> fixed-ip-Y and probably will not be recognized as a return traffic by VM-Zs network stack.
- If VM-Z sends a packet fixed-ip-Z -> floating-ip-B, its translated to fixed-ip-Z -> fixed-ip-X by the router and VM-X will receive it. While this case is very similar to the above cases, the SNAT should not be applied here. The datapath can distinguish these cases by the existence of the asssociation of a floating-ip via the router interface. (floating-ip-B) This behaviour is necessary for the primary use case.[1]
- If VM-Z sends a packet fixed-ip-Z -> floating-ip-A, its translated to fixed-ip-Z -> fixed-ip-X by the router and VM-X will receive it. However, its return traffic fixed-ip-X -> fixed-ip-Z will be translated to floating-ip-B -> fixed-ip-Z and probably will not be recognized as a return traffic by VM-Zs network stack.

[3] https://bugs.launchpad.net/neutron/+bug/1428887

API changes

For API, at least following changes are necessary:

- Add an extension router-interface-fip for feature discovery. The extension does not add any resources or attributes to the REST API.
- Allow floating IP association via a router interface.
- The existing RouterExternalGatewayInUseByFloatingIp check needs to be tweaked so that it doesnt count floating IPs associated via router interfaces.
- A check similar to RouterExternalGatewayInUseByFloatingIp but for router interfaces needs to be introduced.

Datapath support

The datapath needs to be updated to perform actual address translations.

In case of MidoNet, latest versions have the basic support already.[2]

The following is an example of a pseudo rules for logical routers PREROUTING/POSTROUTING processing:

```
PREROUTING

    // floating ip dnat
    [per FIP]
    (dst) matches (fip) -> float dnat, ACCEPT

    // rev-snat for the default snat
    [if default SNAT is enabled on the router]
    (dst) matches (gw port ip) -> rev-snat, ACCEPT

    // rev-snat for MidoNet-specific "same subnet" rules
    [per RIF]
    (inport, dst) matches (rif, rif ip) -> rev-snat, ACCEPT

POSTROUTING

    // floating ip snat
    // multiple rules in order to implement priority (which FIP to use)
    // Note: "fip port" below is a router port, either the router gateway
    // port or router interface, which owns the corresponding FIP
    // configured.
    [per FIP]
    (outport, src) matches (fip port, fip) -> float snat, ACCEPT

    ----- ordering barrier

    [per FIP]
    (src) matches (fip) -> float snat, ACCEPT  // gateway port

    ----- ordering barrier
```

(continues on next page)

[2] https://review.gerrithub.io/#/q/I37d22d43e4bf95bcce870679083aa3e129de8ea7

(continued from previous page)

```
    [per FIP]
    (src) matches (fip) -> float snat, ACCEPT  // non gateway port

    ----- ordering barrier

    // do not apply default snat if it came from external-like network
    // (router interfaces with FIPs, and the gateway port)
    // Note: iptables based implementations need to "emulate" inport
    // match (eg. using marks in PREROUTING) as it isn't available
    // in POSTROUTING.
    [per FIP port]
    (inport) matches (fip port) -> ACCEPT
    [if default SNAT is enabled on the router]
    inport == the gateway port -> ACCEPT

    ----- ordering barrier

    // apply the default snat for the gateway port
    [if default SNAT is enabled on the router]
    outport == the gateway port -> default snat, ACCEPT

    // for non-float -> float traffic  (cf. bug 1428887)
    // "dst-rewritten" condition here means float dnat was applied in
    // prerouting.  in case of iptables based implementations,
    // "--ctstate DNAT" might be used.
    [if default SNAT is enabled on the router]
    dst-rewritten -> default snat, ACCEPT

    // MidoNet-specific "same subnet" rules
    [per RIF]
    (inport == outport == rif) && dst != 169.254.169.254
        -> snat to rif ip, ACCEPT

    // non-float -> non-float in tenant traffic would come here
```

References

8.3 Ocata specs

8.3.1 QoS implementaiton for networking-midonet

This spec describes how to implement QoS extension for networking-midonet. The backend side is covered by another spec.[5]

[5] https://review.gerrithub.io/#/c/289456/

Proposed Change

QoS driver

Use the Neutron QoS plugin as it is. Implement MidoNet specific notification driver which communicates with the MidoNet API.

```
[DEFAULT]
service_plugins = qos

[qos]
notification_drivers = midonet,message_queue
```

setup.cfg:

```
neutron.qos.notification_drivers =
    midonet = midonet.neutron.services.qos.
↪driver:MidoNetQosServiceNotificationDriver
```

Note: *message_queue* driver[4] is the AMQP RPC[7] based driver for the reference implementation. It isnt necessary for MidoNet-only deployments.

Neutron QoS plugin[1] has notification driver mechansim[2], which can be used for networking-midonet to implement backend notifications.

When Neutron QoS plugin receives API requests, it updates the corresponding DB rows. After committing the DB changes, it calls one of the following methods of the loaded notification drivers:

- create_policy
- update_policy
- delete_policy

Note: a request for a rule (eg. *update_policy_rule*) ends up with a notification for the entire policy the rule belongs to.

Note: a request for a specific rule type (eg. *update_policy_dscp_marking_rule*) are automatically converted to a generic method (eg. *update_policy_rule*) by the QoS extension, namely *QoSPluginBase*.[6]

[4] https://github.com/openstack/neutron/blob/2be2d97d11719db88537a9664c95f1b6b11d3707/neutron/services/qos/notification_drivers/message_queue.py#L40

[7] https://docs.openstack.org/neutron/latest/contributor/internals/quality_of_service.html#rpc-communication

[1] https://github.com/openstack/neutron/blob/2be2d97d11719db88537a9664c95f1b6b11d3707/neutron/services/qos/qos_plugin.py

[2] https://github.com/openstack/neutron/blob/2be2d97d11719db88537a9664c95f1b6b11d3707/neutron/services/qos/notification_drivers/manager.py

[6] https://github.com/openstack/neutron/blob/2be2d97d11719db88537a9664c95f1b6b11d3707/neutron/extensions/qos.py#L225

Error handling

Notification driver methods are considered async and always success.[9] Currently theres no convenient way to report errors from the backend. While its possible for a driver to return an error by raising an exception, if multiple drivers are loaded and one of them fails that way, the rest of drivers are just skipped. Even if we assume the simplest case where only MidoNet QoS driver is loaded, theres no mechanism to mark the resource error or rollback the operation. Theres an ongoing effort in Neutron[8] in that area, which might improve the situation.

Core resource extensions

For ML2, the existing QoS extension driver should work.

If we want to make this feature available for the monolithic plugins, the equivalent needs to be implemented for them.

Alternative

Instead of the QoS driver, we can implement the entire QoS plugin by ourselves.

```
[DEFAULT]
service_plugins = midonet_qos
```

setup.cfg:

```
neutron.service_plugins =
    midonet_qos = midonet.neutron.services.qos.plugin:MidonetQosPlugin
```

This might fit the current backend design[5] better.

We can re-use the reference QoS plugin and its DB models by inheriting its class. Its a rather discouraged pattern these days, though. This way the first implementation might be simpler. But it might be tricky to deal with other backends (consider ML2 heterogeneous deployments) and future enhancements in Neutron.

References

[9] https://bugs.launchpad.net/neutron/+bug/1627749
[8] https://review.openstack.org/#/c/351858/