Neutron VPN-as-a-Service Documentation

Release 22.0.1.dev4

Neutron development team

Sep 19, 2024

CONTENTS

1 Using VPNaaS **2**
- 1.1 Installation . . . 2
- 1.2 Configuration Guide . . . 2
  - 1.2.1 Configuration . . . 2
    - neutron_vpnaas.conf . . . 2
    - vpn_agent.ini . . . 2
    - Sample neutron_vpnaas.conf . . . 5
    - Sample vpn_agent.ini . . . 6
  - 1.2.2 Policy . . . 8
    - neutron-vpnaas policies . . . 8
    - Sample Neutron VPNaaS Policy File . . . 11
- 1.3 User Guide . . . 13
  - 1.3.1 Basic Usage . . . 13
- 1.4 Administration Guide . . . 13
  - 1.4.1 VPNaaS Flavors . . . 13

2 For Contributors **14**
- 2.1 Contributor Guide . . . 14
  - 2.1.1 VPNaaS Team . . . 14
    - Core reviewers and Driver maintainers . . . 14
  - 2.1.2 VPNaaS Internals . . . 15
    - Multiple Local Subnets for VPNaaS . . . 15
  - 2.1.3 VPNaaS Tests . . . 20
    - VPNaaS Tempest Tests . . . 20
    - VPNaaS Rally Tests . . . 22
  - 2.1.4 Testing . . . 23
    - Configuring VPNaaS for DevStack . . . 23
    - Testing VPNaaS with devstack . . . 24
  - 2.1.5 Module Reference . . . 30
  - 2.1.6 About This Documentation . . . 30

Neutron VPNaaS provides Virtual Private Network as a Service (VPNaaS) capabilities to Neutron. Maintained as a separate repo, this works in conjunction with the Neutron repo to provide VPN services for OpenStack. The VPNaaS API is implementation as an extension to the OpenStack networking API.

Enjoy!

CHAPTER

ONE

USING VPNAAS

1.1 Installation

The detail instruction to enable neutron VPNaaS is described in the Networking Guide. Follow the instruction after installing `neutron-vpnaas` from distributor packages or PyPI.

1.2 Configuration Guide

1.2.1 Configuration

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

Neutron VPNaaS uses the following configuration files for its various services.

neutron_vpnaas.conf

service_providers

`service_provider`

Type multi-valued

Default `''`

Defines providers for advanced services using the format: <service_type>:<name>:<driver>[:default]

vpn_agent.ini

This is a configuration file for the VPNaaS L3 agent extension of the neutron l3-agent.

ipsec

config_base_dir

Type string

Default `$state_path/ipsec`

Location to store ipsec server config files

ipsec_status_check_interval

Type integer

Default `60`

Interval for checking ipsec status

enable_detailed_logging

Type boolean

Default `False`

Enable detail logging for ipsec pluto process. If the flag set to True, the detailed logging will be written into config_base_dir/<pid>/log. Note: This setting applies to OpenSwan and LibreSwan only. StrongSwan logs to syslog.

pluto

shutdown_check_timeout

Type integer

Default `1`

Initial interval in seconds for checking if pluto daemon is shutdown

Table 1: Deprecated Variations

Group	Name
libreswan	shutdown_check_timeout

shutdown_check_retries

Type integer

Default `5`

The maximum number of retries for checking for pluto daemon shutdown

Table 2: Deprecated Variations

Group	Name
libreswan	shutdown_check_retries

shutdown_check_back_off

Type floating point

Default `1.5`

A factor to increase the retry interval for each retry

Table 3: Deprecated Variations

Group	Name
libreswan	shutdown_check_back_off

restart_check_config

Type boolean

Default `False`

Enable this flag to avoid from unnecessary restart

Table 4: Deprecated Variations

Group	Name
libreswan	restart_check_config

strongswan

ipsec_config_template

Type string

Default `/home/zuul/src/opendev.org/openstack/neutron-vpnaas/neutron_vpnaas/services/vpn/device_drivers/template/strongswan/ipsec.conf.template`

Template file for ipsec configuration.

strongswan_config_template

Type string

Default `/home/zuul/src/opendev.org/openstack/neutron-vpnaas/neutron_vpnaas/services/vpn/device_drivers/template/strongswan/strongswan.conf.template`

Template file for strongswan configuration.

ipsec_secret_template

Type string

Default `/home/zuul/src/opendev.org/openstack/neutron-vpnaas/neutron_vpnaas/services/vpn/device_drivers/template/strongswan/ipsec.secret.template`

Template file for ipsec secret configuration.

default_config_area

Type string

Default `/etc/strongswan.d`

The area where default StrongSwan configuration files are located.

vpnagent

vpn_device_driver

Type multi-valued

Default `neutron_vpnaas.services.vpn.device_drivers.ipsec.OpenSwanDriver, neutron_vpnaas.services.vpn.device_drivers.strongswan_ipsec.StrongSwanDriver, neutron_vpnaas.services.vpn.device_drivers.libreswan_ipsec.LibreSwanDriver`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

The vpn device drivers Neutron will use

The following are sample configuration files for Neutron VPNaaS services and utilities. These are generated from code and reflect the current state of code in the neutron-vpnaas repository.

Sample neutron_vpnaas.conf

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[service_providers]

#
# From neutron.vpnaas
#

# Defines providers for advanced services using the format:
# <service_type>:<name>:<driver>[:default] (multi valued)
#service_provider =
```

Sample vpn_agent.ini

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[ipsec]

#
# From neutron.vpnaas.agent
#

# Location to store ipsec server config files (string value)
#config_base_dir = $state_path/ipsec

# Interval for checking ipsec status (integer value)
#ipsec_status_check_interval = 60

# Enable detail logging for ipsec pluto process. If the flag set to True, the
# detailed logging will be written into config_base_dir/<pid>/log. Note: This
# setting applies to OpenSwan and LibreSwan only. StrongSwan logs to syslog.
# (boolean value)
#enable_detailed_logging = false


[pluto]

#
# From neutron.vpnaas.agent
#

# Initial interval in seconds for checking if pluto daemon is shutdown␣
↪(integer
# value)
#shutdown_check_timeout = 1

# The maximum number of retries for checking for pluto daemon shutdown␣
↪(integer
# value)
#shutdown_check_retries = 5

# A factor to increase the retry interval for each retry (floating point␣
↪value)
#shutdown_check_back_off = 1.5

# Enable this flag to avoid from unnecessary restart (boolean value)
#restart_check_config = false


[strongswan]
```

(continues on next page)

(continued from previous page)

```
#
# From neutron.vpnaas.agent
#

# Template file for ipsec configuration. (string value)
#ipsec_config_template = /home/zuul/src/opendev.org/openstack/neutron-vpnaas/
↪neutron_vpnaas/services/vpn/device_drivers/template/strongswan/ipsec.conf.
↪template

# Template file for strongswan configuration. (string value)
#strongswan_config_template = /home/zuul/src/opendev.org/openstack/neutron-
↪vpnaas/neutron_vpnaas/services/vpn/device_drivers/template/strongswan/
↪strongswan.conf.template

# Template file for ipsec secret configuration. (string value)
#ipsec_secret_template = /home/zuul/src/opendev.org/openstack/neutron-vpnaas/
↪neutron_vpnaas/services/vpn/device_drivers/template/strongswan/ipsec.secret.
↪template

# The area where default StrongSwan configuration files are located. (string
# value)
#default_config_area = /etc/strongswan.d


[vpnagent]

#
# From neutron.vpnaas.agent
#

# The vpn device drivers Neutron will use (multi valued)
#
# This option has a sample default set, which means that
# its actual default value may vary from the one documented
# below.
#vpn_device_driver = neutron_vpnaas.services.vpn.device_drivers.ipsec.
↪OpenSwanDriver, neutron_vpnaas.services.vpn.device_drivers.strongswan_ipsec.
↪StrongSwanDriver, neutron_vpnaas.services.vpn.device_drivers.libreswan_
↪ipsec.LibreSwanDriver
```

1.2.2 Policy

Neutron VPNaaS, like most OpenStack projects, uses a policy language to restrict permissions on REST API actions.

neutron-vpnaas policies

The following is an overview of all available policies in neutron-vpnaas. For a sample configuration file, refer to *Sample Neutron VPNaaS Policy File*.

neutron-vpnaas

`create_endpoint_group`

Default `rule:regular_user`

Operations

- **POST** `/vpn/endpoint-groups`

Create a VPN endpoint group

`update_endpoint_group`

Default `rule:admin_or_owner`

Operations

- **PUT** `/vpn/endpoint-groups/{id}`

Update a VPN endpoint group

`delete_endpoint_group`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/vpn/endpoint-groups/{id}`

Delete a VPN endpoint group

`get_endpoint_group`

Default `rule:admin_or_owner`

Operations

- **GET** `/vpn/endpoint-groups`
- **GET** `/vpn/endpoint-groups/{id}`

Get VPN endpoint groups

`create_ikepolicy`

Default `rule:regular_user`

Operations

- **POST** `/vpn/ikepolicies`

Create an IKE policy

`update_ikepolicy`

Default `rule:admin_or_owner`

Operations

- **PUT** `/vpn/ikepolicies/{id}`

Update an IKE policy

`delete_ikepolicy`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/vpn/ikepolicies/{id}`

Delete an IKE policy

`get_ikepolicy`

Default `rule:admin_or_owner`

Operations

- **GET** `/vpn/ikepolicies`
- **GET** `/vpn/ikepolicies/{id}`

Get IKE policyies

`create_ipsecpolicy`

Default `rule:regular_user`

Operations

- **POST** `/vpn/ipsecpolicies`

Create an IPsec policy

`update_ipsecpolicy`

Default `rule:admin_or_owner`

Operations

- **PUT** `/vpn/ipsecpolicies/{id}`

Update an IPsec policy

`delete_ipsecpolicy`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/vpn/ipsecpolicies/{id}`

Delete an IPsec policy

`get_ipsecpolicy`

Default `rule:admin_or_owner`

Operations

- **GET** /vpn/ipsecpolicies
- **GET** /vpn/ipsecpolicies/{id}

Get IPsec policies

`create_ipsec_site_connection`

Default rule:regular_user

Operations

- **POST** /vpn/ipsec-site-connections

Create an IPsec site connection

`update_ipsec_site_connection`

Default rule:admin_or_owner

Operations

- **PUT** /vpn/ipsec-site-connections/{id}

Update an IPsec site connection

`delete_ipsec_site_connection`

Default rule:admin_or_owner

Operations

- **DELETE** /vpn/ipsec-site-connections/{id}

Delete an IPsec site connection

`get_ipsec_site_connection`

Default rule:admin_or_owner

Operations

- **GET** /vpn/ipsec-site-connections
- **GET** /vpn/ipsec-site-connections/{id}

Get IPsec site connections

`create_vpnservice`

Default rule:regular_user

Operations

- **POST** /vpn/vpnservices

Create a VPN service

`update_vpnservice`

Default rule:admin_or_owner

Operations

- **PUT** /vpn/vpnservices/{id}

Update a VPN service

`delete_vpnservice`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/vpn/vpnservices/{id}`

Delete a VPN service

`get_vpnservice`

Default `rule:admin_or_owner`

Operations

- **GET** `/vpn/vpnservices`
- **GET** `/vpn/vpnservices/{id}`

Get VPN services

Sample Neutron VPNaaS Policy File

The following is a sample neutron-vpnaas 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 neutron-vpnaas when this documentation is built. You must ensure your version of neutron-vpnaas matches the version of this documentation.

```
# Create a VPN endpoint group
# POST  /vpn/endpoint-groups
#"create_endpoint_group": "rule:regular_user"

# Update a VPN endpoint group
# PUT  /vpn/endpoint-groups/{id}
#"update_endpoint_group": "rule:admin_or_owner"

# Delete a VPN endpoint group
# DELETE  /vpn/endpoint-groups/{id}
#"delete_endpoint_group": "rule:admin_or_owner"

# Get VPN endpoint groups
# GET  /vpn/endpoint-groups
# GET  /vpn/endpoint-groups/{id}
#"get_endpoint_group": "rule:admin_or_owner"

# Create an IKE policy
# POST  /vpn/ikepolicies
#"create_ikepolicy": "rule:regular_user"

# Update an IKE policy
```

(continues on next page)

(continued from previous page)

```
# PUT  /vpn/ikepolicies/{id}
#"update_ikepolicy": "rule:admin_or_owner"

# Delete an IKE policy
# DELETE  /vpn/ikepolicies/{id}
#"delete_ikepolicy": "rule:admin_or_owner"

# Get IKE policyies
# GET  /vpn/ikepolicies
# GET  /vpn/ikepolicies/{id}
#"get_ikepolicy": "rule:admin_or_owner"

# Create an IPsec policy
# POST  /vpn/ipsecpolicies
#"create_ipsecpolicy": "rule:regular_user"

# Update an IPsec policy
# PUT  /vpn/ipsecpolicies/{id}
#"update_ipsecpolicy": "rule:admin_or_owner"

# Delete an IPsec policy
# DELETE  /vpn/ipsecpolicies/{id}
#"delete_ipsecpolicy": "rule:admin_or_owner"

# Get IPsec policies
# GET  /vpn/ipsecpolicies
# GET  /vpn/ipsecpolicies/{id}
#"get_ipsecpolicy": "rule:admin_or_owner"

# Create an IPsec site connection
# POST  /vpn/ipsec-site-connections
#"create_ipsec_site_connection": "rule:regular_user"

# Update an IPsec site connection
# PUT  /vpn/ipsec-site-connections/{id}
#"update_ipsec_site_connection": "rule:admin_or_owner"

# Delete an IPsec site connection
# DELETE  /vpn/ipsec-site-connections/{id}
#"delete_ipsec_site_connection": "rule:admin_or_owner"

# Get IPsec site connections
# GET  /vpn/ipsec-site-connections
# GET  /vpn/ipsec-site-connections/{id}
#"get_ipsec_site_connection": "rule:admin_or_owner"

# Create a VPN service
# POST  /vpn/vpnservices
#"create_vpnservice": "rule:regular_user"
```

(continues on next page)

(continued from previous page)

```
# Update a VPN service
# PUT  /vpn/vpnservices/{id}
#"update_vpnservice": "rule:admin_or_owner"

# Delete a VPN service
# DELETE  /vpn/vpnservices/{id}
#"delete_vpnservice": "rule:admin_or_owner"

# Get VPN services
# GET  /vpn/vpnservices
# GET  /vpn/vpnservices/{id}
#"get_vpnservice": "rule:admin_or_owner"
```

1.3 User Guide

1.3.1 Basic Usage

The basic scenarios are explained in the Networking Guide.

1.4 Administration Guide

1.4.1 VPNaaS Flavors

Todo: Info on the different Swan flavors, how they are different, and what Operating Systems support them.

CHAPTER

TWO

FOR CONTRIBUTORS

2.1 Contributor Guide

In the Contributor Guide, you will find information on the design, and architecture of the Neutron Virtual Private Network as a Service repo. This include things like, information on the reference implementation flavors, design details on VPNaaS internals, and testing. Developers will extend this, as needed, in the future to contain more information.

If you would like to contribute to the development of OpenStack, you must follow the steps documented at: https://docs.openstack.org/infra/manual/developers.html

Once those steps have been completed, changes to OpenStack should be submitted for review via the Gerrit tool, following the workflow documented at: https://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad in the neutron project with `vpnaas` tag added.

New features should be filed on Launchpad in the neutron project with `rfe` tag added in order to get decision from neutron drivers team. Before doing that, it is recommended to check Request for Feature Enhancements (RFE) process.

To get in touch with the neutron-vpnaas community, look at the following resource:

- Join the `#openstack-vpnaas` IRC channel on OFTC. This is where the VPNaaS team is available for discussion.
- We will hold for *VPN-as-a-Service (bi-)weekly IRC meeting* when needed in the near further.

These are great places to get recommendations on where to start contributing to neutron-vpnaas.

2.1.1 VPNaaS Team

Core reviewers and Driver maintainers

Core reviewers

The Neutron VPNaaS Core Reviewer Team is responsible for many things that same as Neutron team.

Driver maintainers

The driver maintainers are supposed to try:

- Test the driver
- Fix bugs in the driver
- Keep the driver up-to-date for Neutron
- Keep the driver up-to-date for its backend
- Review relevant patches

The following is a list of drivers and their maintainers. It includes both of in-tree and out-of-tree drivers. (alphabetical order)

Driver	Contact person	IRC nick
LibreSwanDriver	Dongcan Ye	yedongcan
MidonetIPsecVPNDriver[1]	YAMAMOTO Takashi	yamamoto
NSXvIPsecVpnDriver[2]	Roey Chen	roeyc
OpenSwanDriver	Lingxian Kong	kong
StrongSwanDriver	Lingxian Kong	kong
	Cao Xuan Hoang	hoangcx

2.1.2 VPNaaS Internals

Multiple Local Subnets for VPNaaS

As originally implemented, an VPN IPSec connection could have one or more peer subnets specified, but only **one** local subnet. To support multiple local subnets, multiple IPSec connections would be needed.

With the multiple local subnet support, three goals are addressed. First, there can be multiple local and peer endpoints for a single IPSec connection.

Second, validation enforces that the same IP version is used for all endpoints (to reduce complexity and ease testing).

Third, the what is connected is separated from the how to connect, so that other flavors of VPN (as they are developed) can use some of this mechanism.

Design Notes

There were three proposals considered, to support multiple local subnets.

Proposal A was to just add the local subnets to the IPSec connection API. That would be the quickest way, and addresses the first two goals, but not the third.

Proposal B was to create a new API that specifies of the local subnets and peer CIDRs, and reference those in the connection API. This would separate the what is connected from the how to connect, and again addresses the first two goals (only).

[1] networking-midonet: https://docs.openstack.org/networking-midonet/latest/install/installation.html#vpnaas
[2] vmware-nsx: Maintained under the vmware-nsx repository - https://github.com/openstack/vmware-nsx

Proposal C, which was the *selected proposal*, creates a new API that represents the endpoint groups for VPN connections, in the same manner as proposal B. The added flexibility here, though, which meets goal three, is to also include the endpoint group type, thus allowing subnets (local) and CIDRs (peer) to be used for IPSec, but routers, networks, and VLANs to be used for other VPN types (BGP, L2, direct connection). Additional types can be added in the future as needed.

Client CLI API

The originally implemented client CLI APIs (which are still available for backward compatibility) for an IPsec connection are:

```
openstack vpn service create --router ROUTER --subnet SUBNET NAME
openstack vpn ipsec site connection create
    --vpnservice VPNSERVICE
    --ikepolicy IKEPOLICY
    --ipsecpolicy IPSECPOLICY
    --peer-address PEER_ADDRESS
    --peer-id PEER_ID
    --peer-cidr PEER_CIDRS
    --dpd action=ACTION,interval=INTERVAL,timeout=TIMEOUT
    --initiator {bi-directional | response-only}
    --mtu MTU
    --psk PSK
    VPN_IPSEC_SITE_CONNECTION_NAME
```

Changes to the API, to support multiple local subnets, are shown in **highlighted** text:

```
openstack vpn service create --router ROUTER NAME
openstack vpn endpoint group create
    --description OPTIONAL-DESCRIPTION
    --type={subnet,cidr,network,vlan,router}
    --value=ENDPOINT-OF-TYPE[,--value=ENDPOINT-OF-TYPE,...]
    ENDPOINT-GROUP-NAME
openstack vpn ipsec site connection create
    --vpnservice VPNSERVICE
    --ikepolicy IKEPOLICY
    --ipsecpolicy IPSECPOLICY
    --peer-address PEER_ADDRESS
    --peer-id PEER_ID
    --dpd action=ACTION,interval=INTERVAL,timeout=TIMEOUT
    --initiator {bi-directional | response-only}
    --mtu MTU
    --psk PSK
    --local-endpoint-group ENDPOINT-GROUP-UUID
    --peer-endpoint-group ENDPOINT-GROUP-UUID
    VPN_IPSEC_SITE_CONNECTION_NAME
```

The SUBNET in the original service API is optional, and will be used as an indicator of whether or not the multiple local subnets feature is active. See the Backward Compatibility section, below, for details.

For the endpoint groups, the `--type` value is a string, so that other types can be supported in the future.

The endpoint groups API would enforce that the endpoint values are all of the same type, and match the endpoint type specified.

The connection APIs, would then provide additional validation. For example, with IPSec, the endpoint type must be subnet for local, and cidr for peer, all the endpoints should be of the same IP version, and for the local endpoint, all subnets would be on the same router.

For BGP VPN with dynamic routing, only a local endpoint group would be specified, and the type would be network.

The ROUTER may also be able to be removed, in the future, and can be determined, when the connections are created.

Examples

The original APIs to create one side of an IPSec connection with only one local and peer subnet:

```
openstack vpn ike policy create ikepolicy
openstack vpn ipsec policy create ipsecpolicy
openstack vpn service create --router router1 --subnet privateA myvpn
openstack vpn ipsec site connection create
    --vpnservice myvpn
    --ikepolicy ikepolicy
    --ipsecpolicy ipsecpolicy
    --peer-address 172.24.4.13
    --peer-id 172.24.4.13
    --peer-cidr 10.3.0.0/24
    --psk secret
    vpnconnection1
```

The local CIDR is obtained from the subnet, privateA. In this example, that would be 10.1.0.0/24 (because thats how privateA was created).

Using the multiple local subnet feature, the APIs (with changes shown in **highlighted** below:

```
openstack vpn ike policy create ikepolicy
openstack vpn ipsec policy create ipsecpolicy
openstack vpn service create --router router1 myvpn
openstack vpn endpoint group create
    --type=subnet
    --value=privateA
    --value=privateB
    local-eps
openstack vpn endpoint group create
    --type=cidr
    --value=10.3.0.0/24
    peer-eps
openstack vpn ipsec site connection create
    --vpnservice myvpn
    --ikepolicy ikepolicy
    --ipsecpolicy ipsecpolicy
    --peer-address 172.24.4.13
```

(continues on next page)

(continued from previous page)

```
    --peer-id 172.24.4.13
    --psk secret
    --local-endpoint-group local-eps
    --peer-endpoint-group peer-eps
    vpnconnection1
```

The subnets privateA and privateB are used for local endpoints and the 10.3.0.0/24 CIDR is used for the peer endpoint.

Database

The vpn_endpoints table contains single endpoint entries and a reference to the containing endpoint group. The vpn_endpoint_groups table defines the group, specifying the endpoint type.

Database Migration

For an older database, the first subnet, in the subnet entry of the service table can be placed in an endpoint group that will be used for the local endpoints of the connection. The CIDRs from the connection can be placed into another endpoint group for the peer endpoints.

Backwards Compatibility

Operators would like to see this new capability provided, with backward compatibility support. The implication, as I see it, is to provide the ability for end users to be able to switch to the new API at any time, versus being forced to use the new API immediately, upon upgrade to the new release containing this feature. This would apply to both manual API use, and client apps/scripting-tools that would be used to configure VPNaaS.

There are several attributes that are involve here. One is the subnet ID attribute in the VPN service API. The other is the peer CIDR attribute in the IPSec connection API. Both would be specified by endpoint groups in the new API, and these groups would be called out in the IPSec connection API.

A plan to meet the backward compatibility goal of allowing both APIs to be used at once involves taking the following steps.

For VPN service:

- Make the subnet ID attribute optional.
- If subnet ID is specified for create, consider old API mode.
- If subnet ID specified for create, create endpoint group and store ID.
- For delete, if subnet ID exists, delete corresponding endpoint group.
- For show/list, if subnet ID exists, show the ID in output.
- Subnet ID is not mutable, so no change for update API.

For IPSec site to site connection:

- For create, if old API mode, only allow peer-cidr attribute.

- For create, if not old API mode, require local/peer endpoint group IDs attributes.
- For create, if peer-cidr specified, create endpoint group and store ID.
- For create, reject endpoint group ID attributes, if old API mode.
- For create, reject peer-cidr attribute, if not old API mode.
- For create, if old API mode, lookup subnet in service, find containing endpoint group ID and store.
- For delete, if old API mode, delete endpoint group for peer.
- For update of CIDRs (old mode), will delete endpoint group and create new one. (note 1)
- For update of endpoint-group IDs (new mode), will allow different groups to be specified. (note 1,2)
- For show/list, if old API mode, only display the peer CIDR values from peer endpoint group.
- For show/list, if not old API mode, also show local subnets from local endpoint group.

Note 1: Implication is that connection is torn down and re-created (as is done currently).

Note 2: Users would create a new endpoint group, and then select that group, when modifying the IPSec connection.

For endpoint groups:

- For delete, if subnet, and (sole) subnet ID is used in a VPN service (old mode), reject request.
- Updates are not supported, so no action required. (note 2)

Note 2: Allowing updates would require deletion/recreation of connection using endpoint group. Avoiding that complexity.

The thought here is to use endpoint groups under the hood, but if the old API was being used, treat the endpoint groups as if they never existed. Deleting connections and services would remove any endpoint groups, unlike with the new API, where they are independent.

Migration can be used to move any VPNaaS configurations using the old schema to the new schema. This would look at VPN services and for any with a subnet ID, an endpoint group would be created and the group ID stored in any existing IPSec connections for that service. Likewise, any peer CIDRs in a connection would be copied into a new endpoint group and the group ID stored in the connection.

The subnet ID field would then be removed from the VPN service table, and the peer CIDRs table would be removed.

This migration could be done at the time of the new API release, in which case all tenants with existing VPNaaS configurations would use the new API to manage them (but could use old for new configurations).

Alternatively, the migration could be deferred until the old API is removed, to ensure all existing configurations conform to the new schema. Migration tools can then be created to manually migrate individual tenants, as desired.

Stories

For the endpoint groups, stories can cover:

- CRUD API for the endpoint groups.
- Database support for new tables.
- Migration creation of new tables.
- Validation of endpoints for a group (same type).
- Neutron client support for new API.
- Horizon support for new API.
- API documentation update.

For the multiple local subnets, stories can cover:

- create IPsec connection with one local subnet, but using new API.
- create IPSec connection with multiple local subnets.
- Show IPSec connection to display endpoint group IDs (or endpoints?).
- Ensure previous API still works, but uses new tables.
- Validation to ensure old and new APIs are not mixed.
- Modify CLI client.
- Validate multiple local subnets on same router.
- Validate local and peer endpoints are of same IP version.
- Functional tests with multiple local subnets
- API and How-To documentation update

Note: The intent here is to have the initial stories take slices vertically through the process so that we can demonstrate the capability early.

Note: Horizon work to support the changes is not expected to be part of this effort and would be handled by the Horizon team separately, if support is desired.

2.1.3 VPNaaS Tests

VPNaaS Tempest Tests

This contains the tempest test codes for the Neutron VPN as a Service (VPNaaS) service. The tests currently require tempest to be installed via devstack or standalone. It is assumed that you also have Neutron with the Neutron VPNaaS service installed. These tests could also be run against a multinode openstack.

Please see /neutron-vpnaas/devstack/README.md for the required devstack configuration settings for Neutron-VPNaaS.

How to test:

As a tempest plugin, the steps to run tests by hands are:

1. Setup a local working environment for running tempest.

```
tempest init ${your_tempest_dir}
```

2. Enter ${your_tempest_dir}:

```
cd ${your_tempest_dir}
```

3. Check neutron_vpnaas_tests exist in tempest plugins.

```
tempest list-plugins
```

Name	EntryPoint
neutron_tests neutron_vpnaas_tests	neutron_tempest_plugin.plugin:NeutronTempestPlugin neutron_vpnaas.tests.tempest.plugin:VPNTempestPlugin

4. Run neutron_vpnaas tests.

```
tempest run --regex "^neutron_vpnaas.tests.tempest.api\."
```

Usage in gate

In the jenkins gate, devstack-gate/devstack-vm-gate-wrap.sh will invoke tempest with proper configurations, such as:

```
DEVSTACK_GATE_TEMPEST=1
DEVSTACK_GATE_TEMPEST_ALL_PLUGINS=1
DEVSTACK_GATE_TEMPEST_REGEX="^neutron_vpnaas.tests.tempest.api\."
```

The actual raw command in gate running under the tempest code directory is:

```
tox -eall-plugin -- "^neutron_vpnaas.tests.tempest.api\."
```

External Resources

For more information on the tempest, see: https://docs.openstack.org/tempest/latest/.

VPNaaS Rally Tests

This contains the rally test codes for the Neutron VPN as a Service (VPNaaS) service. The tests currently require rally to be installed via devstack or standalone. It is assumed that you also have Neutron with the Neutron VPNaaS service installed.

These tests could also be run against a multinode openstack.

Please see /neutron-vpnaas/devstack/README.md for the required devstack configuration settings for Neutron-VPNaaS.

Structure

1. plugins - Directory where you can add rally plugins. Almost everything in Rally is a plugin. Contains base, common methods and actual scenario tests
2. rally-configs - Contains input configurations for the scenario tests

How to test

Included in the repo are rally tests. For information on rally, please see the rally README.

Create a rally deployment for your cloud and make sure it is active.

```
rally deployment create --file=cloud_cred.json --name=MyCloud
```

You can also create a rally deployment from the environment variables.

```
rally deployment create --fromenv --name=MyCloud
```

Create a folder structure as below

```
sudo mkdir /opt/rally
```

Create a symbolic link to the plugins directory

```
cd /opt/rally
sudo ln -s /opt/stack/neutron-vpnaas/rally-jobs/plugins
```

Run the tests. You can run the tests in various combinations.

- Single Node with DVR with admin credentials
- Single Node with DVR with non admin credentials
- Multi Node with DVR with admin credentials
- Multi Node with DVR with non admin credentials
- Single Node, Non DVR with admin credentials
- Multi Node, Non DVR with admin credentials

Create a `args.json` file with the correct credentials depending on whether it is a single node or multinode cloud. A `args_template.json` file is available at `/opt/stack/neutron-vpnaas/rally-jobs/rally-configs/args_template.json` for your reference.

Update the `rally_config_dvr.yaml` or `rally_config_non_dvr.yaml` file to change the admin/non_admin credentials.

Use the appropriate config files to run either dvr or non_dvr tests.

With DVR:

```
rally task start \
  /opt/stack/neutron-vpnaas/rally-jobs/rally-configs/rally_config_dvr.yaml \
  --task-args-file /opt/stack/neutron-vpnaas/rally-jobs/rally-configs/args.
↪json
```

Non DVR:

```
rally task start \
  /opt/stack/neutron-vpnaas/rally-jobs/rally-configs/rally_config_non_dvr.
↪yaml \
 --task-args-file /opt/stack/neutron-vpnaas/rally-jobs/rally-configs/args.json
```

Note: Non DVR scenario can only be run as admin as you need admin credentials to create a non DVR router.

External Resources

For more information on the rally testing framework see: https://github.com/openstack/rally/.

2.1.4 Testing

Configuring VPNaaS for DevStack

Multinode vs All-In-One

Devstack typically runs in single or All-In-One (AIO) mode. However, it can also be deployed to run on multiple nodes. For VPNaaS, running on an AIO setup is simple, as everything happens on the same node. However, to deploy to a multinode setup requires the following things to happen:

1. Each controller node requires database migrations in support of running VPNaaS.
2. Each network node that would run VPNaaS L3 agent extension.

Therefore, the devstack plugin script needs some extra logic.

How to Configure

To configure VPNaaS, it is only necessary to enable the neutron-vpnaas devstack plugin by adding the following line to the `[[local|localrc]]` section of devstacks local.conf file:

```
enable_plugin neutron-vpnaas <GITURL> [BRANCH]
```

<GITURL> is the URL of a neutron-vpnaas repository `[BRANCH]` is an optional git ref (branch/ref/tag). The default is master.

For example:

```
enable_plugin neutron-vpnaas https://opendev.org/openstack/neutron-vpnaas␣
↪stable/kilo
```

The default implementation for IPSEC package under DevStack is strongswan. However, depending upon the Linux distribution, you may need to override this value. Select libreswan for Fedora/RHEL/CentOS.

For example, install libreswan for CentOS/RHEL 7:

```
IPSEC_PACKAGE=libreswan
```

This VPNaaS devstack plugin code will then

1. Install the common VPNaaS configuration and code,
2. Apply database migrations on nodes that are running the controller (as determined by enabling the q-svc service),

Testing VPNaaS with devstack

Installation

In order to use Neutron-VPNaaS with devstack a single node setup, youll need the following settings in your local.conf.

```
[[local|localrc]]

enable_plugin neutron-vpnaas https://opendev.org/openstack/neutron-vpnaas

disable_service n-net
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
# Optional, to enable tempest configuration as part of devstack
enable_service tempest

# IPsec driver to use. Optional, defaults to strongswan.
IPSEC_PACKAGE="strongswan"
```

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

Quick Test Script

This quick test script creates two sites with a router, a network and a subnet connected with public network. Then, connect both sites via VPN.

You can find an example at tools/test_script.sh in the source tree.

Using Two DevStack Nodes for Testing

You can use two DevStack nodes connected by a common public network to test VPNaaS. The second node can be set up with the same public network as the first node, except it will use a different gateway IP (and hence router IP). In this example, well assume we have two DevStack nodes (`East` and `West`), each running on hardware.

Note:

- You can do the same thing with multiple VM guests, if desired.
- You can also create similar topology using two virtual routers with one devstack.

Example Topology

```
(10.1.0.0/24 - DevStack East)
          |
          |  10.1.0.1
[Neutron Router]
          |  172.24.4.226
          |
          |  172.24.4.225
[Internet GW]
          |
          |
[Internet GW]
          | 172.24.4.232
          |
          | 172.24.4.233
[Neutron Router]
          |  10.2.0.1
          |
(10.2.0.0/24 DevStack West)
```

DevStack Configuration

For East you need to append the following lines to the local.conf, which will give you a private net of 10.1.0.0/24 and public network of 172.24.4.0/24

```
PUBLIC_SUBNET_NAME=yoursubnet
PRIVATE_SUBNET_NAME=mysubnet
FIXED_RANGE=10.1.0.0/24
NETWORK_GATEWAY=10.1.0.1
PUBLIC_NETWORK_GATEWAY=172.24.4.225
Q_FLOATING_ALLOCATION_POOL=start=172.24.4.226,end=172.24.4.231
```

For West you can add the following lines to local.conf to use a different local network, public GW (and implicitly router) IP.

```
PUBLIC_SUBNET_NAME=yoursubnet
PRIVATE_SUBNET_NAME=mysubnet
FIXED_RANGE=10.2.0.0/24
NETWORK_GATEWAY=10.2.0.1
PUBLIC_NETWORK_GATEWAY=172.24.4.232
Q_FLOATING_ALLOCATION_POOL=start=172.24.4.233,end=172.24.4.238
```

VPNaaS Configuration

With DevStack running on East and West and connectivity confirmed (make sure you can ping one router/GW from the other), you can perform these VPNaaS CLI commands.

On East

```
openstack vpn ike policy create ikepolicy1
openstack vpn ipsec policy create ipsecpolicy1
openstack vpn service create --description "My vpn service" \
    --router router1 myvpn
openstack vpn endpoint group create --type subnet --value mysubnet my-locals
openstack vpn endpoint group create --type cidr --value 10.2.0.0/24 my-peers
openstack vpn ipsec site connection create --vpnservice myvpn \
    --ikepolicy ikepolicy1 --ipsecpolicy ipsecpolicy1 \
    --peer-address 172.24.4.233 --peer-id 172.24.4.233 \
    --local-endpoint-group my-locals --peer-endpoint-group my-peers \
    --psk secret vpnconnection1
```

On West

```
openstack vpn ike policy create ikepolicy1
openstack vpn ipsec policy create ipsecpolicy1
openstack vpn service create --description "My vpn service" \
    --router router1 myvpn
openstack vpn endpoint group create --type subnet --value mysubnet my-locals
openstack vpn endpoint group create --type cidr --value 10.1.0.0/24 my-peers
openstack vpn ipsec site connection create --vpnservice myvpn \
```

(continues on next page)

(continued from previous page)

```
    --ikepolicy ikepolicy1 --ipsecpolicy ipsecpolicy1 \
    --peer-address 172.24.4.226 --peer-id 172.24.4.226 \
    --local-endpoint-group my-locals --peer-endpoint-group my-peers \
    --psk secret vpnconnection1
```

Note: Make sure setup security group (open icmp for vpn subnet etc)

Verification

You can spin up VMs on each node, and then from the VM ping to the other one. With tcpdump running on one of the nodes, you can see that pings appear as encrypted packets (ESP). Note that BOOTP, IGMP, and the keepalive packets between the two nodes are not encrypted (nor are pings between the two external IP addresses).

Once stacked, VMs were created for testing, VPN IPsec commands used to establish connections between the nodes, and security group rules added to allow ICMP and SSH.

Using single DevStack and two routers for testing

Simple instructions on how to setup a test environment where a VPNaaS IPsec connection can be established using the reference implementation (StrongSwan). This example uses VirtualBox running on laptop to provide a VM for running DevStack.

The idea here is to have a single OpenStack cloud created using DevStack, two routers (one created automatically), two private networks (one created automatically) 10.1.0.0/24 and 10.2.0.0/24, a VM in each private network, and establish a VPN connection between the two private nets, using the public network (172.24.4.0/24).

Preparation

Create a VM (e.g. 4 GB RAM, 2 CPUs) running Ubuntu 16.04, with NAT I/F for access to the Internet. Clone a DevStack repo with latest.

DevStack Configuration

For single DevStack and two routers case, You can find an example at devstack/local_AIO.conf.sample in the source tree.

Start up the cloud using `./stack.sh` and ensure it completes successfully. Once stacked, you can change `RECLONE` option in local.conf to No.

Cloud Configuration

Once stacking is completed, youll have a private network (10.1.0.0/24), and a router (router1). To prepare for establishing a VPN connection, a second network, subnet, and router needs to be created, and a VM spun up in each private network.

```
# Create second net, subnet, router
source ~/devstack/openrc admin demo
openstack network create privateB
openstack subnet create --network privateB --subnet-range 10.2.0.0/24 --
↪gateway 10.2.0.1 subB
openstack router create routerB
openstack router add subnet routerB subB
openstack router set --external-gateway public routerB

# Start up a VM in the privateA subnet.
PRIVATE_NET=`openstack network show private -c id -f value`
openstack server create --flavor 1 --image cirros-0.3.5-x86_64-uec \
    --nic net-id=$PRIVATE_NET peter

# Start up a VM in the privateB subnet
PRIVATE_NETB=`openstack network show privateB -c id -f value`
openstack server create --flavor 1 --image cirros-0.3.5-x86_64-uec \
    --nic net-id=$PRIVATE_NETB paul
```

At this point, you can verify that you have basic connectivity.

Note: DevStack will create a static route that will allow you to ping the private interface IP of router1 from privateB network. You can remove the route, if desired.

IPsec Site-to-site Connection Creation

The following commands will create the IPsec connection:

```
# Create VPN connections
openstack vpn ike policy create ikepolicy
openstack vpn ipsec policy create ipsecpolicy
openstack vpn service create --router router1 \
    --description "My vpn service" myvpn
openstack vpn endpoint group create --type subnet --value privateA my-localsA
openstack vpn endpoint group create --type cidr --value 10.2.0.0/24 my-peersA
openstack vpn ipsec site connection create --vpnservice myvpn \
    --ikepolicy ikepolicy --ipsecpolicy ipsecpolicy \
    --peer-address 172.24.4.13 --peer-id 172.24.4.13 \
    --local-endpoint-group my-localsA --peer-endpoint-group my-peersA \
    --psk secret vpnconnection1

openstack vpn service create --router routerB \
```

(continues on next page)

(continued from previous page)

```
    --description "My vpn serviceB" myvpnB
openstack vpn endpoint group create --type subnet --value subB my-localsB
openstack vpn endpoint group create --type cidr --value 10.1.0.0/24 my-peersB
openstack vpn ipsec site connection create --vpnservice myvpnB \
    --ikepolicy ikepolicy --ipsecpolicy ipsecpolicy \
    --peer-address 172.24.4.11 --peer-id 172.24.4.11 \
    --local-endpoint-group my-localsB --peer-endpoint-group my-peersB \
    --psk secret vpnconnection2
```

At this point (once the connections become active - which can take up to 30 seconds or so), you should be able to ping from the VM in the privateA network, to the VM in the privateB network. Youll see encrypted packets, if you tcpdump using the qg-# interface from one of the router namespaces. If you delete one of the connections, youll see that the pings fail (if all works out correctly :)).

Note: Because routerB is created manually, its public IP address may change (172.24.4.13 in this case).

Multiple Local Subnets

Early in Mitaka, IPsec site-to-site connections will support multiple local subnets, in addition to the current multiple peer CIDRs. The multiple local subnet feature is triggered by not specifying a local subnet, when creating a VPN service. Backwards compatibility is maintained with single local subnets, by providing the subnet in the VPN service creation.

To support multiple local subnets, a new capability has been provided (since Liberty), called Endpoint Groups. Each endpoint group will define one or more endpoints of a specific type, and can be used to specify both local and peer endpoints for IPsec connections. The Endpoint Groups separate the what gets connected from the how to connect for a VPN service, and can be used for different flavors of VPN, in the future. An example:

```
# Create VPN connections
openstack vpn ike policy create ikepolicy
openstack vpn ipsec policy create ipsecpolicy
openstack vpn service create --router router1 \
    --description "My vpn service" myvpnC
```

To prepare for an IPsec site-to-site, one would create an endpoint group for the local subnets, and an endpoint group for the peer CIDRs, like so:

```
openstack vpn endpoint group create --type subnet --value privateA --value␣
↪privateA2 my-locals
openstack vpn endpoint group create --type cidr --value 10.2.0.0/24 --value␣
↪20.2.0.0/24 my-peers
```

where privateA and privateA2 are two local (private) subnets, and 10.2.0.0/24 and 20.2.0.0/24 are two CIDRs representing peer (private) subnets that will be used by a connection. Then, when creating the IPsec site-to-site connection, these endpoint group IDs would be specified, instead of the peer-cidrs attribute:

```
openstack vpn ipsec site connection create --vpnservice myvpnC \
    --ikepolicy ikepolicy --ipsecpolicy ipsecpolicy \
    --peer-address 172.24.4.11 --peer-id 172.24.4.11 \
    --local-endpoint-group my-locals --peer-endpoint-group my-peers \
    --psk secret vpnconnection3
```

Note:

- The validation logic makes sure that endpoint groups and peer CIDRs are not intermixed.
- Endpoint group types are subnet, cidr, network, router, and vlan. However, only subnet and cidr are implemented (for IPsec use).
- The endpoints in a group must be of the same type, although It can mix IP versions.
- For IPsec connections, validation currently enforces that the local and peer endpoints all use the same IP version.
- IPsec connection validation requires that local endpoints are subnets, and peer endpoints are CIDRs.
- Migration will convert information for any existing VPN services and connections to endpoint groups.
- The original APIs will work for backward compatibility.

Todo: Add notes about functional testing, with info on how different reference drivers are tested.

2.1.5 Module Reference

Todo: Add in all the big modules as automodule indexes.

2.1.6 About This Documentation

This documentation is generated by the Sphinx toolkit and lives in the source tree. Additional documentation on VPNaaS and other components of OpenStack can be found on the OpenStack wiki and the Neutron section of the wiki (see the VPN related pages). The Neutron Development wiki is also a good resource for new contributors.